Force.com IDE Developer Guide

Force.com IDE v35.0, Winter '16

@salesforcedocs

Last updated: December 17, 2015

CONTENTS

CHAPTER 1 Force.com IDE UI Overview

In this chapter ...

The Force.com IDE plug-in adds a Force.com perspective to the Eclipse IDE. An Eclipse perspective is a collection of views, editors, and other user-interface tools that are organized for a specific purpose. The Force.com IDE plug-in includes specialized tools for developing applications for the Force.com platform.

To activate the Force.com perspective, select **Window** > **Open Perspective** > **Other** > **Force.com Perspective**.

SEE ALSO:

Force.com Toolbar Buttons

Package Explorer

Force.com Context (Right-Click) Menu

Force.com IDE Editors

Schema Explorer

Create New Apex Class

Create an Apex Class from a WSDL

Create New Apex Trigger

Create New Custom Application

Create New Custom Object

Create New HomePage Component

Create New HomePage Layout

Create New Letterhead

Create New Profile

Create New Visualforce Component

Create New Visualforce Page

Add Workflow From Server

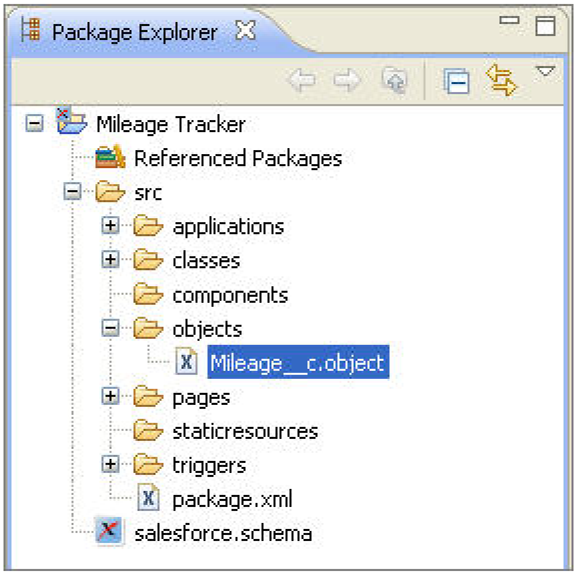
Force.com Toolbar Buttons

The Force.com perspective provides the following buttons in the Eclipse toolbar:

Button	Description
	Saves changes to the active file. If you are working online, the file is saved to the server and refreshed. Note: In offline mode, this button saves the file locally only. To save a file to the server, right-click the file and choose **Force.com** > **Save to Server**. For details, see Working Offline.
	Opens Salesforce in your default browser.
	Opens the Salesforce Developers community in your default browser.
	Launches the Deploy to Server wizard for the active Force.com project. For details, see Deploying Code with the Force.com IDE.

Package Explorer

The Package Explorer provides a hierarchical tree view of the projects and files in your workspace, and works just like the standard Eclipse Project Explorer. The Force.com IDE provides additional menu options; for details, see Force.com Context (Right-Click) Menu.



For information on standard project folders and files, see Force.com Project Basics.

Force.com Context (Right-Click) Menu

To access the Force.com context menu, right-click an object in the Package Explorer and scroll down to **Force.com** (or press the period key (".") to jump to the **Force.com** menu). The options in the menu depend on the item selected. In addition to the Force.com-specific options below, the right-click menu provides access to standard actions, including search and delete.

Menu Item	Description
Add Force.com Nature	Enables Force.com functionality for a project. Use this option when you check out a project from source control to associate the project with Force.com. Available at project level only.
Remove Force.com Nature	Disables Force.com functionality for a project. Available at project level only.
Work Offline	Disables all implicit server communication. In offline mode, saving files and compiling code is done locally by default, but you can still save to the server by using **Save to Server**. Available at project level only.
Work Online	Re-enables implicit server communication. In online mode, saving a file saves the file locally and on the server. Available at project level only.
Execute Anonymous	Executes anonymous blocks of code and commits to the database if successful. For more information, see Execute Anonymous View.
Show in Salesforce Web	Displays the selected component in the default Web browser. This action automatically logs you in based on your project's connection settings, and is a handy way to immediately see the changes you've made.
Upgrade Project	Upgrades the project to the latest version. For more information, see Upgrade Project.
Refresh from Server	Replaces the current project definition with the server definition. You can refresh individual components, folders, or the entire project. For more information, see Refresh from Server.
Save to Server	Saves project files to the server if you are working in offline mode. For more information, see Save to Server.
Synchronize with Server	Synchronizes your project and server. For more information, see Server Synchronization.
Deploy to Server	Deploys components to the server. You can deploy individual files, folders, or an entire project. For more information, see Deploying Code with the Force.com IDE.
Add/Remove Metadata Components	Configures the server metadata components to be synchronized with the project. For more information, see Add/Remove Metadata Components.
Project Properties	Opens the Project Properties dialog. For more information, see Project Properties. Available at project level only.

Force.com IDE Editors

When you open a file in Eclipse, the file opens in the editor area of the IDE. The editor that opens is based on the type of file. The Force.com IDE provides specialized editors for XML, Apex and Visualforce files.

The standard Eclipse editor functionality includes many useful editing features, such as syntax highlighting, unlimited undo and redo, element selection and formatting, and document formatting. The main Eclipse menu bar and toolbar contain operations that are applicable to the active editor.

The tabs at the top of the editor area display the names of files currently open for editing. An asterisk (*) indicates that a file has unsaved changes.

The tabs at the bottom of the editor area allow you to toggle between different views of the active file. For example, when you open an XML file, you can edit it in **Design** or **Source** view.

Force.com IDE editors have built-in content assistance for standard Apex and Force.com metadata types. When editing a custom object, type the less-than sign ("<") to pick from a list of valid elements. The Apex editor also provides code assistance; for details, see Apex Code Assist.

Note: To use content assistance in a third-party XML editor installed as an Eclipse plugin, you must extract the `schema/metadata.xsd` file from `<eclipse_installation>/plugins/com.salesforce.ide.api_<version>.jar` and point the XML editor to this file.

When you save a file, the editor checks for validity. You will not be able to save files to the server until they are formatted and compile correctly.

SEE ALSO:

Apex Editor

Apex Code Assist

Schema Explorer

The Schema Explorer is a tool for browsing the metadata of a Salesforce organization, and for querying data. The Schema Explorer presents the logged-in user's view of the Salesforce data model, including object visibility, permissions, data types, lookup values and other information that is useful in developing applications on the Force.com platform.

The Schema Explorer provides a hierarchical tree view of your organization's schema. Every object that can be accessed by the logged-in user is displayed at the root level of the tree, with additional related properties beneath those roots. Tooltips provide quick summary information.

Note: The Schema Explorer is read-only. You can query data in the schema, but you cannot modify your schema through the Schema Explorer.

To open the Schema Explorer:

1. In the Package Explorer, expand the node for your Force.com project.
2. Double-click **salesforce.schema**.

To create a query:

1. Expand the **Schema** objects in the tree view and select the objects or fields you want to query on.
2. Click **Run Me**.
3. You can also create a query by entering SOQL directly in the **Query Results** window. For example, to search for all the accounts in your organization that start with the letter B, enter the following SOQL query:

```
SELECT Name FROM Account WHERE Name LIKE 'B%'
```

For more information about how to write SOQL queries, see the *Force.com SOQL and SOSL Reference*.

Force.com Wizards

The Force.com IDE plug-in provides a set of wizards for creating new objects and components, accessible from the **File** > **New** menu (or the **New** button in the toolbar).

- Create New Force.com Project
- Create New Apex Class
- Create an Apex Class from a WSDL
- Create New Apex Trigger
- Create New Custom Application
- Create New Custom Object
- Create New HomePage Component
- Create New HomePage Layout
- Create New Letterhead
- Create New Profile
- Create New Visualforce Component
- Create New Visualforce Page
- Add Workflow From Server

Create New Force.com Project

To launch the Create New Force.com Project wizard, select **File** > **New** > **Force.com Project**.

Note: If you do not see **Force.com Project** in the **File** > **New** menu, you are not using the Force.com perspective. You can still create a Force.com project without using the Force.com perspective by selecting **File** > **New** > **Other** > **Force.com** > **Force.com Project**; however, you should use the Force.com perspective because it includes other views and features that aid development. To activate the Force.com perspective, select **Window** > **Open Perspective** > **Other** > **Force.com Perspective**.

The New Force.com Project wizard has two pages:

1. On the first page of the wizard, enter the appropriate properties for the project. For details on these settings, see Force.com Project Properties. Click **Next**.

   Note: When you create a new project, you might be prompted about a new master password. This is a separate password of your choosing required by Eclipse secure storage, and is not associated with your Salesforce credentials. For details on Eclipse secure storage, see the *Eclipse Workbench User Guide*.

2. On the Project Contents page, choose which metadata components are retrieved:

Field	Description
Apex and Visualforce (classes, triggers, pages, components, and static resources)	Select this option to retrieve only Apex and Visualforce components, including classes, triggers, components, pages, and static resources.
Selected metadata components	Select this option and click **Choose...** to open the Choose Metadata Components dialog.
Contents of package	Select this option and click **Choose package...** to retrieve the contents of a particular package. Note that this option is only

Field	Description
	available when there are packages on the organization you connect to.
None	Select this option to retrieve no components. You might want to select this option if you are working in a team-based environment where source files are checked out from a source-control system. When creating a project offline, this is the only option available.

3. Click **Finish** to create the new project.

Note: The selections you make in the New Force.com Project wizard are used to create the `package.xml` file, which defines which components from the server are downloaded into your project. To later modify the project contents, right-click your project and choose **Properties**, and go to **Force.com** > **Project Contents**. In special cases you may want to edit the `package.xml` file by hand. For information on changing the contents of your project manifests after project creation, see About Package.xml.

Create New Apex Class

To launch the Create New Apex Class wizard, select **File** > **New** > **Apex Class** or right-click your project and choose **New** > **Apex Class**. The wizard allows you to choose a template for the new class.

Field	Description
Name	Enter a name for the class. Salesforce recommends following Java standards for naming, that is, classes start with a capital letter, methods start with a lower-case verb, and variables have meaningful names. It is not legal to define a class and interface with the same name in the same class, or an inner class with the same name as its outer class. However, since methods and variables have their own namespaces within the class, it is legal for a variable, method, and a class within a class to have the same name.
Version	Select the version of the API this object conforms to from the dropdown list.
Template	Optional. Select a template for the Apex class from the drop-down list: • **Default** - Creates a class without a template. • **Test Class** - Creates a class with the `@isTest` annotation. Classes defined with the `@isTest` annotation do not count against the organization size limit for Apex scripts.

Field	Description
	• **Inbound Email Client** - Creates a class that processes the contents, headers, and attachments of inbound e-mail.

SEE ALSO:

- Apex Editor
- Apex Code Assist
- Execute Anonymous View
- Create an Apex Class from a WSDL
- Testing Code with the Force.com IDE
- Apex Test Results View
- Debug Logs for Apex Test Results
- Apex Code Settings: Log Category and Log Level

Create an Apex Class from a WSDL

You can generate Apex classes from a WSDL document that is stored on a local hard drive or network. Creating a class by consuming a WSDL document allows you to make callouts to an external Web service in your Apex code by calling the methods in the generated class.

To create an Apex class from a WSDL, click **File** > **New** > **Apex Class from WSDL**, or right-click your project and choose **New** > **Apex Class from WSDL**.

The first step of the WSDL to Apex wizard enables you to specify a WSDL file to import and choose whether to generate asynchronous classes.

Field	Description
WSDL File	Click **Browse** to select a WSDL file from your local hard drive or a network drive.
Add Async Class	Check this box to generate asynchronous classes. The generated classes enable you to make synchronous callouts. If you choose to also generate asynchronous classes by checking this box, you can make asynchronous callouts from a Visualforce page by using those classes.

In the second step of the wizard, you can rename the new classes. The wizard creates a default class name for each namespace in the WSDL. 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. You can rename synchronous classes only. The names of asynchronous classes correspond to the names of their synchronous counterparts and contain a prefix of `Async`.

Note: Open-source code for the WSDL to Apex wizard is available in the WSDL2Apex GitHub repository. Developers in the GitHub community can add enhancements or customizations to WSDL2Apex. The Force.com IDE plug-in is a snapshot of the WSDL2Apex and idecore GitHub repositories at the time of the latest official plug-in release.

Create New Apex Trigger

To launch the Create New Apex Trigger wizard, click **File** > **New** > **Apex Trigger** or right-click your project and choose **New** > **Apex Trigger**. The wizard allows you to select the object that the trigger acts on and specify when the trigger fires.

Field	Description
Name	Enter a name for the trigger.
Version	Select the version of the API this object conforms to from the dropdown list.
Object	Select the object that the trigger acts on. If you do not see the object you are looking for in the dropdown list, click **Refresh Objects**.
Apex Trigger Operations	Use the checkboxes to specify when the trigger fires: • before insert • before update • before delete • after insert • after update • after delete • after undelete

SEE ALSO:

- Apex Editor
- Apex Code Assist
- Execute Anonymous View
- Testing Code with the Force.com IDE
- Apex Test Results View
- Debug Logs for Apex Test Results
- Apex Code Settings: Log Category and Log Level

Create New Custom Application

To launch the Create New Custom Application wizard, click **New** > **Custom Application**. You can edit the generated file to adjust default values, or add new attributes and values.

Field	Description
Label	Enter the label as it should appear in the Salesforce user interface.
Name	Enter a name for the application.

Create New Custom Object

To launch the Create New Custom Object wizard, click **New** > **Custom Object**. You can edit the generated file to adjust default values, or add new attributes and values.

Field	Description
Label	Enter the label as it should appear in the Salesforce user interface.
Plural Label	Enter the plural label of the object. If the default language for this organization does require plural labels, this field may not be available.
Name	Enter a name for the custom object. The name is automatically appended with "`__c`".

Create New HomePage Component

To launch the Create New HomePage Component wizard, click **New** > **HomePage Component**. You can edit the generated file to adjust default values, or add new attributes and values.

Field	Description
Name	Enter a name for the component.

Create New HomePage Layout

To launch the Create New HomePage Layout wizard, click **New** > **HomePage Layout**. You can edit the generated file to adjust default values, or add new attributes and values.

Field	Description
Name	Enter a name for the layout.

Create New Letterhead

To launch the Create New Letterhead wizard, click **New** > **Letterhead**. Use this wizard to create a new letterhead for use in an e-mail template. You can edit the generated file to adjust default values, or add new attributes and values.

Field	Description
Label	Enter the label as it should appear in the Salesforce user interface.
Name	Enter a name for the letterhead.

Create New Profile

To launch the Create New Profile wizard, click **New** > **Profile**. You can edit the generated file to adjust default values, or add new attributes and values.

Field	Description
Name	Enter a name for the profile.

Create New Visualforce Component

To launch the Create New Visualforce Component wizard, click **New** > **Visualforce Component**. Use this wizard to create a component for a Visualforce page.

Field	Description
Label	Enter the label of the component as it should appear in the Visualforce page.
Name	Enter a name for the component.
Version	Select the version of the API this object conforms to from the dropdown list.

Create New Visualforce Page

To launch the Create New Visualforce Page wizard, click **New** > **Visualforce Page**

Field	Description
Label	Enter a label as it should appear on the Visualforce page.
Name	Enter a name for the page.
Version	Select the version of the API this object conforms to from the dropdown list.

Add Workflow From Server

Use the Add Workflow From Server wizard to add the workflow file for a particular object to your Force.com project. To launch the wizard, click **New** > **Workflow**.

Field	Description
Object	Select the object the workflow should be based on. If you do not see the object in the list, click **Refresh Objects**.

The workflow file contains individual workflow components (Alerts, Rules, Tasks) for an object. You cannot create these components from the IDE at this time.

To create new workflow components:

1. In Package Explorer, right-click your project and choose **Force.com** > **Show in Salesforce Web**.
2. In the Web browser, from Setup, enter `Workflow` in the `Quick Find` box, then select the appropriate option.

Force.com Views (Tabs)

The Force.com IDE plug-in includes a collection of views for navigating logs and test output. By default, these views are shown as tabs at the bottom of the Eclipse IDE. You can drag and drop the tabs to rearrange views or move them to a different area of the IDE. To reopen a view after it has been closed, use the **Show View** button in the bottom left corner of the IDE window.

- Problems View
- Apex Test Results View
- Execute Anonymous View
- Synchronize View
- Force.com Log Viewer

Problems View

The Problems view displays system-generated errors and warnings associated with project resources. When you perform an action, such as saving a file to the server, any issues detected by the Force.com IDE or returned from the server will be listed in the Problems view.

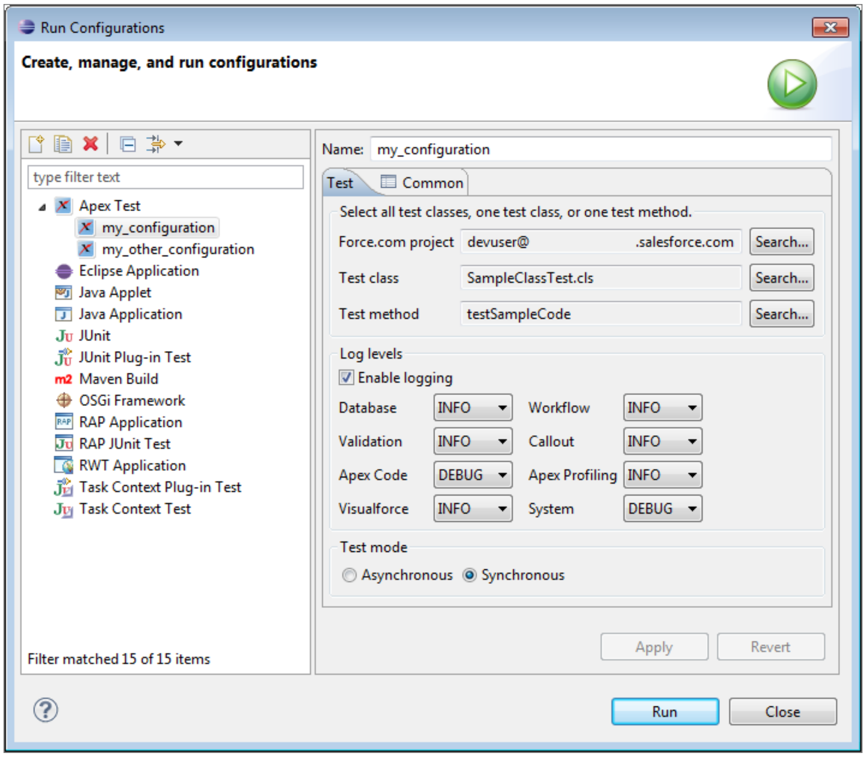
The Problems view is a standard Eclipse view, as described in the *Workbench User Guide* in the Eclipse help.

Apex Test Results View

The Force.com IDE's Apex Test Results view displays the results of your test runs. This view is useful for troubleshooting code, tuning performance, and checking resource usage.

Note: All Apex test execution occurs on the server. Before testing your code, save any changes to the server.

To execute Apex unit tests, select **Run** > **Run Configurations** > **Apex Test**. To create a test run configuration, then select **New launch configuration** (🗋). To execute the selected test run configuration, click **Run**.



Name: my_configuration

Select all test classes, one test class, or one test method.

Force.com project: devuser@ .salesforce.com

Test class: SampleClassTest.cls

Test method: testSampleCode

Log levels

☑ Enable logging

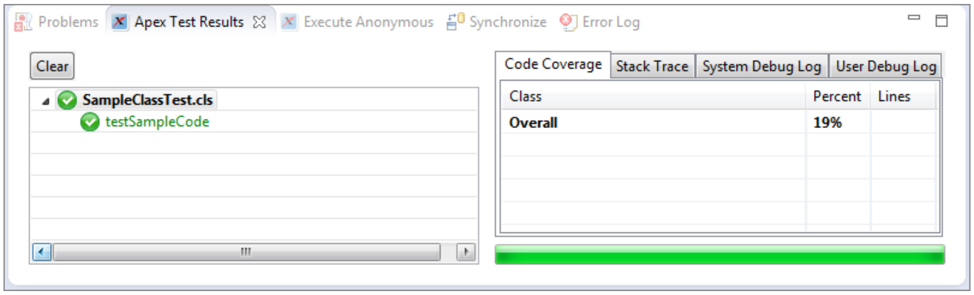
Database	INFO	Workflow	INFO
Validation	INFO	Callout	INFO
Apex Code	DEBUG	Apex Profiling	INFO
Visualforce	INFO	System	DEBUG

Test mode

○ Asynchronous ● Synchronous

> **Note:** To set log levels for your Apex test runs, use run configurations. Logging settings in your project properties apply only when you're deploying code.

After the a test run, the Apex Test Runner view displays results. The left pane displays test results for each class and method in the test run. The right pane displays code coverage, the stack trace, system debug logs, and user debug logs. Make sure that your code has better coverage than the code in this sample org!



SampleClassTest.cls
testSampleCode

Class	Percent	Lines
Overall	19%	

Debug Logs for Apex Test Results

When you run a test, the output is sent to the log. Debug logs display on the right side of the Force.com IDE's Apex Test Results view.

- The first part of the log details the events that occurred during the test run.
- The next few lines give details on how long it took to execute specific lines of code. This information is useful for performance tuning.
- The debug log then lists how many resources a program uses and the total amount available for each resource. The Apex runtime engine tracks the resources that every script uses so that a single script doesn't monopolize the servers. If you write a script that goes over one of the limits, you receive an error message.

To change your logging levels, create or edit an Apex test run configuration. To access your Apex test run configurations, select **Run** > **Run Configurations** > **Apex Test**. To create a test run configuration, then select **New launch configuration** ( ).

Execute Anonymous View

The Force.com IDE Execute Anonymous view allows you to execute an anonymous block of Apex.

Anonymous blocks help you to quickly evaluate Apex on the fly, or to write scripts that change dynamically at runtime. For example, you might write a client Web application that takes input from a user, such as a name and address, and then use an anonymous block to insert a contact with that name and address into the database.

The content of an anonymous block can include user-defined methods and exceptions. It cannot include the keyword `static`.

You do not need to manually commit database changes made by an anonymous block. If your Apex script completes successfully, any database changes are automatically committed. If your Apex script does not complete successfully, any changes made to the database are rolled back.

After your anonymous block is executed on the server, the Results area in the Execute Anonymous view will display the following:

- Status information for the compile and execute phases of the call, including any errors that occur.
- The debug log content, including the output of any calls to the `System.debug()` method.
- The Apex stack trace of any uncaught script execution exceptions, including the class, method, and line number for each call stack element.

Note: Unlike classes and triggers, anonymous blocks are executed as the current user and can fail to compile if the script violates the user's object- and field-level permissions.

Synchronize View

The Force.com IDE Synchronize view allows you to synchronize your project with the server without changing perspectives.

Note: By default, Eclipse prompts you to switch to the Team Synchronization perspective when you synchronize your project with the server. To configure Eclipse to use the Force.com IDE Synchronize view:

1. In the Eclipse menu bar, choose **Window** > **Preferences**.
2. In the Preferences dialog, click **Team** in the navigation tree on the left.
3. In the **Perspectives** drop-down list, choose **None**.

This setting also affects synchronizing with a version control system.

SEE ALSO:

Server Synchronization

Force.com Log Viewer

The Force.com Log Viewer provides an easy way to access the contents of the Force.com IDE system log file. If you encounter exception messages or other problems while using the IDE, this log file may help diagnose and resolve the issue, and can be provided to Salesforce customer support.

To display the Force.com Log Viewer, click **Help** > **Show Force.com IDE Log**.

Note:

- To monitor for events as they occur, the Force.com Log Viewer frequently refreshes itself from the log file on disk. This activity may slow your system.
- We recommend that you keep the Force.com Log Viewer closed during normal use of the IDE.

To include more detail in the system log, run the IDE in debug mode. For information, see Troubleshooting the Force.com IDE: Debug Mode.

CHAPTER 2 Getting Started with the Force.com IDE

In this chapter ...

The Force.com IDE is an integrated development environment for developing applications on the Force.com platform using Apex, Visualforce, and metadata components. Designed for developers and development teams, the IDE provides tools to accelerate Force.com application development. These tools include wizards, source code editors, test execution tools, deployment aids, integrated help, and an interactive debugger.

The Force.com IDE is built on top of the open-source Eclipse Platform and is available as a plug-in. The plug-in is open source—you can find and contribute to its source code on GitHub.

You can use the Force.com IDE to:

- Test and debug Apex classes and triggers using the Apex Test Results view.
- Run anonymous blocks of Apex on the server in the Execute Anonymous view.
- Browse schema objects and fields or assemble and execute SOQL queries in the Schema Explorer.
- Synchronize project contents with changes on the server using Save to Server, Refresh from Server, and Synchronize with Server commands.
- Utilize the Compare Editor to merge changes when conflicts are detected.
- Deploy metadata components from one Salesforce organization to another, or validate a planned deployment without saving changes, using the Deploy to Server wizard.

To get started working in the Force.com IDE, look at the following topics.

- Developing with the Force.com IDE
- Testing Code with the Force.com IDE
- Deploying Code with the Force.com IDE

Developing with the Force.com IDE

The Force.com IDE allows you to create and edit Apex, Visualforce, and XML metadata components using source code editors that provide syntax highlighting, code assistance, and server-based error checking.

Working with Force.com Projects

The first step is to create a Force.com project associated with your Salesforce organization (the home organization) and download metadata components. You can manage projects and files using the standard views, tools, and commands in the Eclipse IDE workbench, and use features from the Force.com IDE.

- Use a wizard to create your first project. For details, see Create New Force.com Project.
- Create or edit the `package.xml` project manifest file using the Choose Metadata Components dialog. For more information, see About Package.xml and Project Properties.
- Develop in your project and on the server at the same time, with multiple developers accessing the same information. For information on how to refresh data and synchronize changes, see Server Synchronization.

Writing Code

Once you have created a project, you can use the Force.com IDE to create, edit and manipulate objects and components.

- Use Force.com Wizards to create objects and components, including Apex classes and triggers, and Visualforce components and pages. Each wizard allows you to define properties for the object and creates it with standard attributes and values.
- Edit your code in the feature-rich Force.com IDE Editors, which include content assistance for built-in Apex types.
- Execute anonymous blocks of Apex and commit them to the database using the Execute Anonymous View.
- Browse schema objects and fields or assemble and execute SOQL queries in the Schema Explorer.

SEE ALSO:

Force.com Project Basics

Create New Apex Class

Create New Apex Trigger

Apex Editor

Apex Code Assist

Execute Anonymous View

Working Offline

Testing Code with the Force.com IDE

To measure your code's quality, track your code coverage and test case coverage. Use the Force.com IDE to create and execute unit tests for your code's actions and behaviors and to adjust the granularity of your test runs' logging.

One of the most important and powerful features of the Force.com platform is its built-in support for automated testing. The Apex language includes the ability to define and execute unit tests, which are pieces of code that verify that your application works the way you intended. Each unit test is defined as a test method, and you can execute your test methods to see which tests are passing or failing. Regularly executing test methods gives you instant insight into the quality of your code and provides an early warning system for detecting regressions.

There are two common ways to measure your code's quality using unit tests.

- *Code coverage* identifies which lines of code a set of unit tests exercises. Code coverage is reported as a percentage. This metric helps you identify the sections of code that are untested and therefore at greatest risk of containing a bug or introducing a regression. Each time you run a set of unit tests on the Force.com platform, a code coverage number is returned. A list of uncovered lines of code in the classes and triggers invoked by the tests is also returned.

  Note: The Force.com platform requires at least 75% of your code to be covered by automated tests before you can deploy it to a production organization. We recommend that you strive for 100% coverage. The code coverage restriction is not enforced for sandbox or Developer Edition organizations.

- *Test case coverage* identifies real-world scenarios in which you expect your code to execute. Even if you have 100% code coverage, bugs can be hiding in your code. To help prevent bugs, ensure that your test values reflect the full set of real-world possibilities, including corner cases. Test cases are not actual unit tests, but are documents that specify what your unit tests are intended to do. High test case coverage means that most or all of the real-world scenarios that you have identified are implemented as unit tests.

Developing a rich set of automatic tests gives you confidence that your code works correctly. Having good test coverage can help you catch bugs when a code change suddenly causes a test to fail. Having a robust set of tests helps us help you, too. Salesforce executes all your tests before each major release to help us avoid regressions. We run each test once in the existing version of our service—the one currently in production—and once in the release-candidate version. We compare the results to identify unexpected functionality changes between releases. To ensure robust test coverage, regularly do the following.

- Create unit tests within the implementation class or in a separate test class. Test classes that are annotated with `@isTest` do not count against your Apex storage limits. Creating a test class is done like any other class. For details, see Create New Apex Class.
- Execute unit tests on the server and view the results using Apex test run configurations and the Apex Test Results view.
- Configure debugging output and system logs using Apex test run configurations.

Salesforce recommends that you write tests for the following:

Single action
Test to verify that a single record produces the correct, expected result.

Bulk actions
Any Apex code, whether a trigger, a class or an extension, may be invoked for 1 to 200 records. You must test not only the single record case, but the bulk cases as well.

Positive behavior
Test to verify that the expected behavior occurs through every expected permutation, that is, that the user filled out everything correctly and did not go past the limits.

Negative behavior
There are likely limits to your applications, such as not being able to add a future date, not being able to specify a negative amount, and so on. You must test for the negative case and verify that the error messages are correctly produced as well as for the positive, within the limits cases.

Restricted user
Test whether a user with restricted access to the sObjects used in your code sees the expected behavior. That is, whether they can run the code or receive error messages.

Note: Conditional and ternary operators are not considered executed unless both the positive and negative branches are executed.

For more information, see “Understanding Testing in Apex” in the *Force.com Apex Code Developer's Guide*.

SEE ALSO:

- Apex Test Results View
- Debug Logs for Apex Test Results
- Apex Code Settings: Log Category and Log Level
- Problems View
- Force.com Log Viewer
- Troubleshooting the Force.com IDE: Debug Mode

Deploying Code with the Force.com IDE

Once you have created application components and tested them in your own development organization, you usually want to migrate them to a different organization for testing, staging, publication, or production use by end users.

Though each Force.com metadata component lives and runs in a particular organization on Force.com servers, the metadata files you work with in the Force.com IDE are portable from one organization to another. Deploying is the process of pushing your local project files into a different Salesforce organization than your project’s home organization. The Force.com IDE provides the **Deploy to Server** wizard to guide you through the deployment process.

Before you deploy, you can streamline your deployment process by creating a new project with only the components you want to deploy. For information, see Create New Force.com Project.

Note: If you deploy to a production organization, you must meet the Apex testing compliance policy: 75% of your Apex must be covered by unit tests, and all of those tests must complete successfully. Additionally, each trigger must have some code coverage.

To open the Deploy to Server wizard, right-click the `src` folder of the project and select **Force.com** > **Deploy to Server**. You can also select individual classes or triggers, multiple classes and triggers, or the `classes` or `triggers` folder. Before the wizard opens, the IDE checks for conflicts between the project and the home organization. If conflicts are found, you are given the option to synchronize before continuing.

The Deploy to Server wizard includes the following pages:

1. On the Destination Details page, enter the connection information for the target organization and click **Next**.
2. On the Archive Options page, optionally create backup files for this deployment and click **Next**.
3. On the Deployment Plan page, review the actions to be performed in this deployment. Click **Next** to execute the deployment.
4. The Deployment Result page displays the result of the deployment. You should see Success displayed at the top of the page. If you do not, review the deployment logs by clicking **View Logs**. If you need assistance with deployment errors, click **Save** to save your log file for future reference.

SEE ALSO:

- Destination Details
- Archive Options
- Deployment Plan

Quickstart: Using the Force.com IDE

This simplified quickstart guides you step-by-step through the following tasks:

1. Create a new project.
2. Create an Apex class and trigger that populates the custom field with text.
3. Add a unit test to automatically test the method.

Note: To complete this exercise, you must have administrative access to two different organizations: either two Developer Edition organizations, or a Developer Edition organization and a sandbox organization. Do not use a production organization for this exercise.

1. Create a Project

The following steps create an Eclipse project and connect it to the home organization (the organization associated with the project):

1. Select **File** > **New** > **Force.com Project**.

   Note: If you do not see **Force.com Project** in the **File** > **New** menu, you are not using the Force.com perspective. You can still create a Force.com project without using the Force.com perspective by selecting **File** > **New** > **Other** > **Force.com** > **Force.com Project**; however, you should use the Force.com perspective because it includes other views and features that aid development. To activate the Force.com perspective, select **Window** > **Open Perspective** > **Other** > **Force.com Perspective**.

2. Enter the following information and click **Next**:

Field	Value
Project Name	Enter a name for your project.
Username	Enter the username you use to log in to the organization associated with this project ("home organization"). The username must have the "Modify All Data" permission.
Password	Enter the password for the specified username.
Security Token	If you are using a security token, enter the value here.
Environment	Choose the appropriate environment for your connection (**Developer Edition** or **Sandbox**).
Do not change endpoint	Leave this option unchecked.
Timeout (sec)	Set to a value between 3 and 600 seconds.
Proxy Settings	If you connect through a proxy, click the Proxy Settings link to open the **Network Connections** dialog.

   For details on these settings, see Force.com Project Properties.

3. On the Project Contents page, choose **Apex and Visualforce**.
4. Click **Finish** to create the project, connect it to the associated organization, and copy components from the home organization into the project in the appropriate folders.

Now that you have created a project, when you edit and save items in the project, the edits are saved to the server. (If the items fail to compile, they are not saved.)

2. Create Apex Components

The following steps create an Apex class and trigger to pre-populate a field on the Accounts tab.

Create a class that populates the `Hello` field on the Accounts tab with the word “World”:

1. In Eclipse, right-click on the project you created in the Package Explorer, and select **New** > **Apex Class**.
2. Enter `MyHelloWorld` as the name for the project, leave the other settings as is, and click **Finish**.
3. The source for the new `MyHelloWorld.cls` class is displayed. Replace the auto-generated text with the following:

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

   Save your changes.

4. If the IDE asks if you want to save the changes to the server, click **Yes**.

Next, create a trigger that calls `MyHelloWorld.cls` whenever a new record is created.

1. Right-click on your project in the Package Explorer and select **File** > **New** > **Apex Trigger**.
2. Enter `helloWorldAccountTrigger` as the name of the trigger.
3. Click the **Object** drop-down list and select **Account**.
4. In the **Apex Trigger Operations** section, check the **before insert** checkbox.
5. Click **Finish**.
6. The source for the new `helloWorldAccountTrigger.trigger` file is displayed. Replace the auto-generated text with the following:

```
trigger helloWorldAccountTrigger on Account
(before insert) {
MyHelloWorld.addHelloWorld(Trigger.new);
}
```

   Save your changes.

To see your new Apex class and trigger working, log in to your Salesforce organization in a browser and create a new account. You should see the **Description** field pre-populated with the value “Hello World”.

At this point, you can make changes to the class or trigger in the project. When you save, the changes are automatically saved to the associated organization, assuming no conflicts exist. If you make changes from the organization itself, you must synchronize those changes to the project. For details, see Server Synchronization.

3. Add 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 are flagged with the `testMethod` keyword in the method definition. A rich set of unit tests gives you confidence that your code works correctly and can help you catch bugs when a code change suddenly causes a test to fail.

Note: While you can develop and execute Apex classes and triggers freely in your Developer Edition or Sandbox organization, at least 75% of your code must be covered by automated unit tests before you can deploy it to a production organization.

The steps below create a simple unit test for the Hello World program:

1. Open `MyHelloWorld.cls` and add the test methods below:

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

2. Create another Apex class with the name `MyHelloWorld.cls`. Replace the auto-generated text with the following test class:

```
@isTest
private class MyHelloWorldTest {
   // Simple test of the method
   // MyHelloWorld.addHelloWorld(Account[])
   static testMethod void test_addHelloWorld()
   {
      // Set up test data set
      Account testAcct1 = new Account();
      Account testAcct2 = new Account(Description = 'Foo');
      Account[] accts = new Account[] { testAcct1, testAcct2 };

      // Execute code with test data
      MyHelloWorld.addHelloWorld(accts);  // call

      // Confirm results
      System.assertEquals('Hello World', accts[0].Description);
      System.assertEquals('Hello World', accts[1].Description);
   }

   // Simple test of the trigger helloWorldAccountTrigger
   static testMethod void test_helloWorldAccountTrigger()
   {
      // Set up test data set
      Account testAcct1 = new Account(Name='One');
      Account testAcct2 = new Account(Name='Two', Description = 'Foo');
      Account[] accts = new Account[] { testAcct1, testAcct2 };

      // Execute trigger with test data set
      insert accts;
      // Confirm results
      Account[] acctQuery = [SELECT Description FROM Account WHERE Id = :accts[0].Id OR
 Id = :accts[1].Id];
      System.assertEquals('Hello World', acctQuery[0].Description);
      System.assertEquals('Hello World', acctQuery[1].Description);
   }
}
```

3. Save your changes.
4. Right-click on the classes folder in Project Explorer and select **Force.com** > **Run Tests**.

For detailed information on testing, see Testing Code with the Force.com IDE.

Working Offline

Increase your productivity by reducing the time required for saving the projects you create using the Force.com IDE in offline mode, which is the default setting.

By default, you'll create your projects in offline mode. When you work in offline mode, you avoid compiling your organization's Apex code each time you save your files. If, however, you have a stable Internet connection and a small- to medium-sized code base, you may wish to toggle your projects to online mode.

Toggle Between Offline and Online Modes

Right-click your project in the Package Explorer and choose **Force.com**, and then select the mode in which you want to work.

Save Files

When working online, the files you save are saved on the server and then retrieved from the server into your project. When working offline, files are saved locally. You can, however, save files to the server when you right-click the file in the Package Explorer and choose **Force.com** > **Save to Server**. This action is the same as when you save files while working online.

SEE ALSO:

Server Synchronization

Updating the Force.com IDE

It is recommended that you always use the latest version of the Force.com IDE so you have the latest features, bug fixes, and documentation. To update the IDE:

1. From Eclipse, select **Help** > **Software and Workspace Center**. Alternatively, you can click the **Check for Updates** link on the Force.com Start Page.
2. The Software and Workspace Center opens, searches automatically, and provides a link to any relevant updates in the Updates Available section.
3. Select the Force.com IDE update and click **Apply 1 change**.
4. In the dialog that opens, confirm that the Install Location is correct and click **Update**.
5. Accept the license agreement and click **Finish**.

6. You are prompted to restart Eclipse; click **OK**.

After you have updated the Force.com IDE, your projects might also need to be upgraded to the latest version. For more information, see Upgrade Project.

Developing with the Force.com IDE

The Force.com IDE allows you to create and edit Apex, Visualforce, and XML metadata components using source code editors that provide syntax highlighting, code assistance, and server-based error checking.

Working with Force.com Projects

The first step is to create a Force.com project associated with your Salesforce organization (the home organization) and download metadata components. You can manage projects and files using the standard views, tools, and commands in the Eclipse IDE workbench, and use features from the Force.com IDE.

- Use a wizard to create your first project. For details, see Create New Force.com Project.
- Create or edit the `package.xml` project manifest file using the Choose Metadata Components dialog. For more information, see About Package.xml and Project Properties.
- Develop in your project and on the server at the same time, with multiple developers accessing the same information. For information on how to refresh data and synchronize changes, see Server Synchronization.

Writing Code

Once you have created a project, you can use the Force.com IDE to create, edit and manipulate objects and components.

- Use Force.com Wizards to create objects and components, including Apex classes and triggers, and Visualforce components and pages. Each wizard allows you to define properties for the object and creates it with standard attributes and values.
- Edit your code in the feature-rich Force.com IDE Editors, which include content assistance for built-in Apex types.
- Execute anonymous blocks of Apex and commit them to the database using the Execute Anonymous View.
- Browse schema objects and fields or assemble and execute SOQL queries in the Schema Explorer.

SEE ALSO:

Force.com Project Basics

Create New Apex Class

Create New Apex Trigger

Apex Editor

Apex Code Assist

Execute Anonymous View

Working Offline

Force.com Project Basics

Like most integrated development environments, the Force.com IDE organizes application resources into containers called projects. Unlike traditional software development projects where source code is compiled to create runnable applications, the resources in a Force.com project live within a Salesforce organization and are copied into the local project for editing.

Each Force.com project is connected to a Salesforce organization, known as its home organization, and contains a set of files which correspond to metadata components stored in the home organization's database. When you save a file in a Force.com project, it is immediately saved to the server, or if the server finds an error that error message is returned to the Force.com IDE and displayed in the Problems View.

Note: Because Force.com projects are connected to a live environment, they may be impacted by components running in the home organization but not downloaded into the project. For example, a workflow rule defined in your organization will run in the same transaction as an Apex trigger if both are defined against the same object and the workflow rule's criteria are met.

To manage Force.com projects and resources, use the Package Explorer view, which displays each project's resources in a hierarchical tree view. Force.com projects are organized into the following folders:

- **src** – This folder contains all of the metadata components in your project. Metadata components are organized into folders, by type. For a list of types and folders, see Metadata Types.
- **src/package.xml** – This control file, known as the project manifest, determines what metadata components are retrieved from the server when synchronizing with the project's home organization. For more information see About Package.xml.
- **salesforce.schema** – Opening this file activates the Schema Explorer for the project's home organization. For more information see Schema Explorer.
- **Referenced Packages** – This folder contains the contents of any managed packages that are installed in the project's home organization. These files are read only; customizing installed managed packages from the Force.com IDE is not supported.

SEE ALSO:

Create New Force.com Project

Add/Remove Metadata Components

Project Properties

Working Offline

About Package.xml

Create New Force.com Project

To launch the Create New Force.com Project wizard, select **File** > **New** > **Force.com Project**.

Note: If you do not see **Force.com Project** in the **File** > **New** menu, you are not using the Force.com perspective. You can still create a Force.com project without using the Force.com perspective by selecting **File** > **New** > **Other** > **Force.com** > **Force.com Project**; however, you should use the Force.com perspective because it includes other views and features that aid development. To activate the Force.com perspective, select **Window** > **Open Perspective** > **Other** > **Force.com Perspective**.

The New Force.com Project wizard has two pages:

1. On the first page of the wizard, enter the appropriate properties for the project. For details on these settings, see Force.com Project Properties. Click **Next**.

   Note: When you create a new project, you might be prompted about a new master password. This is a separate password of your choosing required by Eclipse secure storage, and is not associated with your Salesforce credentials. For details on Eclipse secure storage, see the *Eclipse Workbench User Guide*.

2. On the Project Contents page, choose which metadata components are retrieved:

Field	Description
Apex and Visualforce (classes, triggers, pages, components, and static resources)	Select this option to retrieve only Apex and Visualforce components, including classes, triggers, components, pages, and static resources.
Selected metadata components	Select this option and click **Choose...** to open the Choose Metadata Components dialog.

Field	Description
Contents of package	Select this option and click **Choose package...** to retrieve the contents of a particular package. Note that this option is only available when there are packages on the organization you connect to.
None	Select this option to retrieve no components. You might want to select this option if you are working in a team-based environment where source files are checked out from a source-control system. When creating a project offline, this is the only option available.

3. Click **Finish** to create the new project.

Note: The selections you make in the New Force.com Project wizard are used to create the `package.xml` file, which defines which components from the server are downloaded into your project. To later modify the project contents, right-click your project and choose **Properties**, and go to **Force.com** > **Project Contents**. In special cases you may want to edit the `package.xml` file by hand. For information on changing the contents of your project manifests after project creation, see About Package.xml.

Project Properties

Use the Force.com IDE's project properties dialog to modify an existing project: to update organization credentials and connection settings, change the included metadata components, and adjust view settings. To open the project properties dialog, right-click your top-level project folder and select **Properties**.

The project properties dialog includes standard Eclipse property pages and the following Force.com project property pages.

- **Force.com Project Properties**: The properties on the main page determine a Force.com project's Salesforce connection settings. Use this page to update your password or security token, change timeout values, or associate your project with a different home organization. For details on these properties, see Force.com Project Properties.
- **Apex Code Settings**: This page allows you to define default logging levels for the tests that execute during your deployments and during Execute Anonymous code execution. For details on these settings, see Apex Code Settings: Log Category and Log Level.
- **Force.com Project Contents**: This page shows the contents of your project based on the project manifest file (`package.xml`). To add or remove metadata components from your project, click **Add/Remove Metadata Components**. If this project contains a package, the server determines which files are retrieved. You can change the contents of the package only on the server, not from this page. When you change project contents, you are prompted to refresh files from the server.

Add/Remove Metadata Components

To add existing components in your home organization to a project:

1. In the Package Explorer, right-click the project and choose **Properties**.
2. In the Project Properties dialog, expand the node for Force.com and click **Project Contents**.
3. Click **Add/Remove Metadata Components** to open the Choose Metadata Components dialog.

Because a Force.com project is connected to its home organization, simply deleting a metadata file from a project does not always permanently remove it. Depending on whether your intention is to delete a component from the home organization as well as your project, or merely to exclude it from your project, the steps you will take are different.

Note: If you remove a component from a project without deleting it from the server, you can add it to your project again. The component will also be available in other projects that connect to the same server. If you delete the component on the server, the component will not be accessible from other projects. The only way to retrieve it again is by using the Web interface and viewing your Recycle Bin.

To remove a component from the project only:

1. In the Package Explorer, right-click your project and choose **Properties**.
2. In the Project Properties dialog, expand the Force.com node and click **Project Contents**.
3. On the Project Contents page, click **Add/Remove Metadata Components**.
4. Use the Choose Metadata Components dialog to remove the component.
5. In the Package Explorer, expand the nodes to find the component you want to remove. The project will no longer download the component from your home organization during synchronization, but you still need to delete your local copy of the file.
6. In the Package Explorer, right-click the component and choose **Delete**.
7. You will be prompted whether you want to delete the file. Click **Yes** to remove the local copy.
8. In the Remote Delete Confirmation dialog, you will be prompted whether to also delete the component on the server. Click **No** to leave the server-side component unaffected.

To remove a component from the project and delete it from the server:

1. In the Package Explorer, expand the nodes to find the component you want to delete.
2. Right-click the component and choose **Delete**.
3. You will be prompted whether you want to delete the file. Click **Yes** to remove the local copy.
4. Finally, in the Remote Delete Confirmation dialog, you will be prompted whether to also delete the component on the server. Click **Yes** to delete the component entirely.

SEE ALSO:

Choose Metadata Components

Field-Level Security Warning

Choose Metadata Components

Use the Choose Metadata Components dialog to specify which Force.com metadata components you want to retrieve in a project. Click a top-level folder to select all components of a particular type. This option will also retrieve new components that are added via the Web interface whenever you refresh from server. (Selecting the top-level folder is the equivalent of adding the wildcard character to the `package.xml` file. For more information, see About Package.xml.)

Only fully editable packages are available; those created in your organization or installed unmanaged packages. Installed managed packages are not available in this list.

Note: A Force.com project is defined by a `package.xml` file. While you can edit this file by hand, it's much easier and less error prone to use the Choose Metadata Components dialog to create or edit this file. If you edit the `package.xml` file by hand, your changes may be overwritten if you open the Choose Metadata Components dialog. For more information, see About Package.xml.

The following controls help you navigate the available components:

Field	Description
Filter	Enter text to filter all metadata components that start with the letters you enter in this field. You can use the * character within the filter string as a wildcard.
Show only selected items	Select this checkbox to show only the items that have been selected. This is useful after you make several selections and want to see only what will be retrieved in your project.
Select All	Selects all components.
Deselect All	Deselects all components.
Expand	Expands all nodes.
Collapse	Collapses all nodes.
Refresh	Refreshes selections from server.

Field-Level Security Warning

Including both objects and profiles in your project will cause profiles to contain field-level security for all of that object's fields. The settings in your project will overwrite field-level security settings on the server when the profiles are deployed. If you are developing in a sandbox where field-level security is not set up correctly, and you deploy to a production organization, you will overwrite the field-level security in the production organization. To prevent overwriting production field-level security, do one of the following:

- Do not include profiles in your project
- Only choose object fields for which you want to overwrite the field-level security
- Make sure your sandbox security settings are exactly the same as your production org

Component Properties

You can use the Component Properties dialog to view the server properties of a metadata component, such as its name, type, status, package and namespace membership, and the date the file was created and last modified.

To view Component Properties:

1. In the Package Explorer, locate the component file you want to view.
2. Right-click the component and choose **Properties**.
3. Click **Force.com Component** in the navigation pane, on the left side of the file properties dialog.

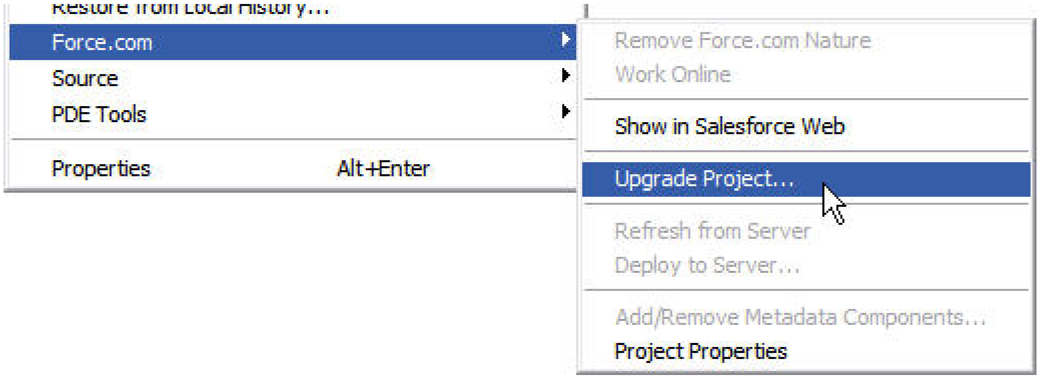
Upgrade Project

When you create a Force.com project, it's designed to work with a specific version of the server. When the server is upgraded to the next version, your projects need to be updated so you can have access to the latest features and metadata.

Note: To upgrade a project, the connection settings for username, password, and proxy settings (if applicable) must be up to date.

To upgrade a Force.com project:

1. Right click a project and choose **Force.com** > **Upgrade Project** to open the Project Upgrade wizard. (This option only appears if an upgrade is available.)



2. On the first page of the wizard, review the information and click **Next** to continue.
3. On the second page of the wizard, review the full details of what will be changed. If you don't want to upgrade all of these components, click **Cancel**. Otherwise click **Finish**.
4. On the final page of the wizard, review your changes.
5. Click **Finish** to retrieve the specified components.

Force.com Wizards

The Force.com IDE plug-in provides a set of wizards for creating new objects and components, accessible from the **File** > **New** menu (or the **New** button in the toolbar).

- Create New Force.com Project
- Create New Apex Class
- Create an Apex Class from a WSDL
- Create New Apex Trigger
- Create New Custom Application
- Create New Custom Object
- Create New HomePage Component
- Create New HomePage Layout
- Create New Letterhead
- Create New Profile
- Create New Visualforce Component
- Create New Visualforce Page
- Add Workflow From Server

Apex Editor

To edit an Apex class or trigger:

1. In the Package Explorer, expand the project node.
2. Apex classes and triggers are in separate folders. Expand the `/classes` or `/triggers` folder and double-click the name of the item you want to edit.

3. Use the **Source** tab of the editor for writing Apex and the **Metadata** tab for editing the associated metadata file.

Note: The editor has content assistance for all built-in types, including both Apex and XML metadata.

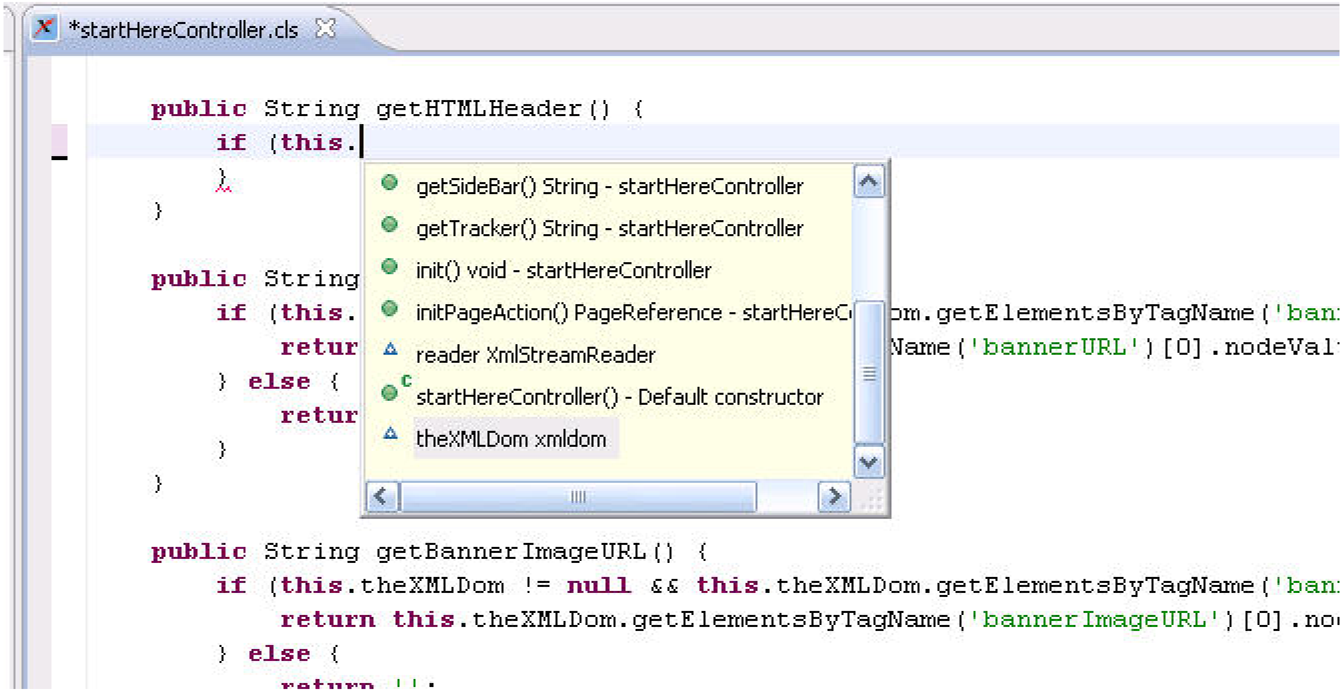
SEE ALSO:

Force.com IDE Editors

Apex Code Assist

Apex Code Assist

Apex code assist is activated when you begin to type a valid line of Apex in the editor. Code assist opens a list of available code completions that aid in both development speed and accuracy.


*startHereController.cls
getSideBar() String - startHereController
getTracker() String - startHereController
init() void - startHereController
initPageAction() PageReference - startHereC
reader XmlStreamReader
startHereController() - Default constructor
theXMLDom xmldom

```
    public String getHTMLHeader() {
        if (this.
        }
    }

    public String
        if (this.                                    om.getElementsByTagName('ban
            retur                                    Name('bannerURL')[0].nodeVal
        } else {
            retur
        }
    }

    public String getBannerImageURL() {
        if (this.theXMLDom != null && this.theXMLDom.getElementsByTagName('ban
            return this.theXMLDom.getElementsByTagName('bannerImageURL')[0].no
        } else {
            return '';
```

To use code assist:

1. Insert the cursor in the editor area.
2. If you are typing a class or variable name, type a dot (".") and pause for a moment, and the proposal window opens. Otherwise press Ctrl + Space on the keyboard to activate the proposal window.
3. Use the mouse or keyboard to select an item from the list.
4. Click or press Enter on a selected line in the list to insert the selection into the editor.

Execute Anonymous View

The Force.com IDE Execute Anonymous view allows you to execute an anonymous block of Apex.

Anonymous blocks help you to quickly evaluate Apex on the fly, or to write scripts that change dynamically at runtime. For example, you might write a client Web application that takes input from a user, such as a name and address, and then use an anonymous block to insert a contact with that name and address into the database.

The content of an anonymous block can include user-defined methods and exceptions. It cannot include the keyword `static`.

You do not need to manually commit database changes made by an anonymous block. If your Apex script completes successfully, any database changes are automatically committed. If your Apex script does not complete successfully, any changes made to the database are rolled back.

After your anonymous block is executed on the server, the Results area in the Execute Anonymous view will display the following:

- Status information for the compile and execute phases of the call, including any errors that occur.
- The debug log content, including the output of any calls to the `System.debug()` method.
- The Apex stack trace of any uncaught script execution exceptions, including the class, method, and line number for each call stack element.

Note: Unlike classes and triggers, anonymous blocks are executed as the current user and can fail to compile if the script violates the user's object- and field-level permissions.

Server Synchronization

When you create a Force.com project, the files you specify in your project manifest are copied from the *server* (your Salesforce organization) and stored locally on your computer.

Note: The term *project* refers to the local files on your computer, while the term *server* refers to the live metadata components in a Salesforce organization.

As you develop in the IDE and make changes to the metadata files in your project, it is possible that you (or another developer or administrator) may make changes to the metadata components directly in the browser from the Salesforce Setup menu. Or if you are working in a team-based development environment, other developers may make changes in the source files in your shared repository. In either case, when metadata is changed outside your project, your project is not immediately aware of those changes and your files may get out of sync.

There are three actions you can take to keep your project and organization files in sync. All of these actions are available by right-clicking the project **src** folder and choosing **Force.com**:

- Save to Server
- Refresh from Server
- Synchronize with Server

Note: If you receive a save error that updates only your local instance (your local project or source control repository), this indicates that your files are not in sync with the server. To replace project files with server definitions, use **Refresh from Server**. To push project files to the server, use **Save to Server**.

SEE ALSO:

Save to Server

Refresh from Server

Synchronize with Server

Synchronize View

Save to Server

Saving files to the server overwrites the metadata components in your organization with the definition in your project files, then refreshes the local files you saved with new copies from your organization. You can save individual files, or all the files in your project.

To save changes in your project:

- Click the Save icon, or right-click the **src** node in the Package Explorer and choose **Force.com** > **Save to Server**. This saves all the files in your project to the server.

To save an individual file:

- Select a component in the Package Explorer and choose **Force.com** > **Save to Server**. This saves only the specified file to the server.

Note: If the metadata components on the server have changed since your last save or refresh and a conflict is detected, you may be asked to enter the Synchronize with Server view.

Refresh from Server

To refresh projects, folders, or individual items:

- In the Package Explorer, right click the item you want to refresh and choose **Force.com** > **Refresh from Server**.

Note: **Refresh from Server** only refreshes the contents specified in the `package.xml` project manifest file. That is, if a file in your project is removed from `package.xml`, you no longer receive new versions of that file when performing a **Refresh from Server**.

Synchronize with Server

When making changes to your home organization in both the IDE and in the browser, your IDE project files may become out of date. When your project is out of sync, you have three options: use your project files, use the server files, or compare the differences between the two. To avoid inadvertently overwriting changes made outside your project, Salesforce recommends using **Synchronize with Server** whenever a conflict is detected. **Synchronize with Server** opens conflicting files in a diff tool so you can compare and merge them.

To synchronize your project with the home organization:

1. In the IDE, right-click the project name in the Package Explorer or other navigation view and select **Force.com** > **Synchronize with Server**. The first time you select this option, the IDE asks if you wish to display the Synchronize view. Click **Yes**. If any project files need to be synchronized, the project is displayed in the Synchronize view with a red arrow and X if there are changes in the project component, and with a blue arrow if there are changes in the home organization. If nothing needs to be synchronized, the Synchronize view displays the message “No changes in 'Latest From Salesforce (Workspace)'”, and the number of conflicts at the bottom of the IDE all show 0.
2. Resolve any reported conflicts. Open the project and the relevant folders until you find the component with a red arrow and X or blue arrow. Right-click the component and select one of the following options, depending on the action you wish to take:
   - **Apply server to project**: Completely replaces the component with the corresponding item in the home organization.
   - **Apply project to server**: Completely replaces the corresponding item in the home organization with the component in the project.
   - **Open in Compare Editor**: Opens the editor so you can inspect each conflict and choose to keep either the home organization change or the component as it is for each conflict. You cannot make any other choice; that is, you cannot overwrite a change in the home organization.
3. Click the save icon when you have resolved all conflicts. You may close the Synchronize view when all conflicts are resolved.

Note: The Synchronize view may not recognize items in the home organization that are not in the project. In these cases, you can right-click the project name and select **Force.com** > **Refresh from server** to refresh the components in a project.

Testing Code with the Force.com IDE

To measure your code's quality, track your code coverage and test case coverage. Use the Force.com IDE to create and execute unit tests for your code's actions and behaviors and to adjust the granularity of your test runs' logging.

One of the most important and powerful features of the Force.com platform is its built-in support for automated testing. The Apex language includes the ability to define and execute unit tests, which are pieces of code that verify that your application works the way you intended. Each unit test is defined as a test method, and you can execute your test methods to see which tests are passing or failing. Regularly executing test methods gives you instant insight into the quality of your code and provides an early warning system for detecting regressions.

There are two common ways to measure your code's quality using unit tests.

- *Code coverage* identifies which lines of code a set of unit tests exercises. Code coverage is reported as a percentage. This metric helps you identify the sections of code that are untested and therefore at greatest risk of containing a bug or introducing a regression. Each time you run a set of unit tests on the Force.com platform, a code coverage number is returned. A list of uncovered lines of code in the classes and triggers invoked by the tests is also returned.

  Note: The Force.com platform requires at least 75% of your code to be covered by automated tests before you can deploy it to a production organization. We recommend that you strive for 100% coverage. The code coverage restriction is not enforced for sandbox or Developer Edition organizations.

- *Test case coverage* identifies real-world scenarios in which you expect your code to execute. Even if you have 100% code coverage, bugs can be hiding in your code. To help prevent bugs, ensure that your test values reflect the full set of real-world possibilities, including corner cases. Test cases are not actual unit tests, but are documents that specify what your unit tests are intended to do. High test case coverage means that most or all of the real-world scenarios that you have identified are implemented as unit tests.

Developing a rich set of automatic tests gives you confidence that your code works correctly. Having good test coverage can help you catch bugs when a code change suddenly causes a test to fail. Having a robust set of tests helps us help you, too. Salesforce executes all your tests before each major release to help us avoid regressions. We run each test once in the existing version of our service—the one currently in production—and once in the release-candidate version. We compare the results to identify unexpected functionality changes between releases. To ensure robust test coverage, regularly do the following.

- Create unit tests within the implementation class or in a separate test class. Test classes that are annotated with `@isTest` do not count against your Apex storage limits. Creating a test class is done like any other class. For details, see Create New Apex Class.
- Execute unit tests on the server and view the results using Apex test run configurations and the Apex Test Results view.
- Configure debugging output and system logs using Apex test run configurations.

Salesforce recommends that you write tests for the following:

Single action
Test to verify that a single record produces the correct, expected result.

Bulk actions
Any Apex code, whether a trigger, a class or an extension, may be invoked for 1 to 200 records. You must test not only the single record case, but the bulk cases as well.

Positive behavior
Test to verify that the expected behavior occurs through every expected permutation, that is, that the user filled out everything correctly and did not go past the limits.

Negative behavior
There are likely limits to your applications, such as not being able to add a future date, not being able to specify a negative amount, and so on. You must test for the negative case and verify that the error messages are correctly produced as well as for the positive, within the limits cases.

Restricted user

Test whether a user with restricted access to the sObjects used in your code sees the expected behavior. That is, whether they can run the code or receive error messages.

Note: Conditional and ternary operators are not considered executed unless both the positive and negative branches are executed.

For more information, see “Understanding Testing in Apex” in the *Force.com Apex Code Developer's Guide.*

SEE ALSO:

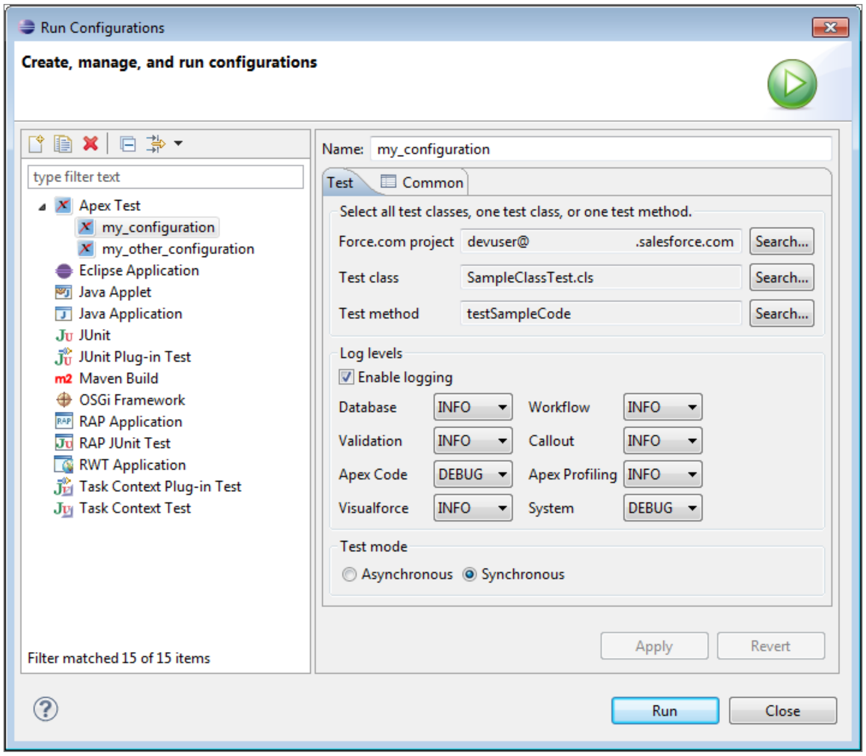
- Apex Test Results View
- Debug Logs for Apex Test Results
- Apex Code Settings: Log Category and Log Level
- Problems View
- Force.com Log Viewer
- Troubleshooting the Force.com IDE: Debug Mode

Apex Test Results View

The Force.com IDE’s Apex Test Results view displays the results of your test runs. This view is useful for troubleshooting code, tuning performance, and checking resource usage.

Note: All Apex test execution occurs on the server. Before testing your code, save any changes to the server.

To execute Apex unit tests, select **Run** > **Run Configurations** > **Apex Test**. To create a test run configuration, then select **New launch configuration** ( ). To execute the selected test run configuration, click **Run**.



Name: my_configuration

Select all test classes, one test class, or one test method.

Force.com project: devuser@ .salesforce.com

Test class: SampleClassTest.cls

Test method: testSampleCode

Log levels

☑ Enable logging

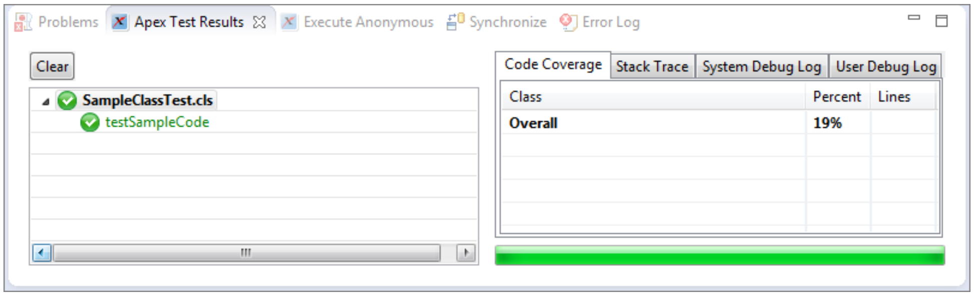
Database	INFO	Workflow	INFO
Validation	INFO	Callout	INFO
Apex Code	DEBUG	Apex Profiling	INFO
Visualforce	INFO	System	DEBUG

Test mode

○ Asynchronous ● Synchronous

Note: To set log levels for your Apex test runs, use run configurations. Logging settings in your project properties apply only when you're deploying code.

After the a test run, the Apex Test Runner view displays results. The left pane displays test results for each class and method in the test run. The right pane displays code coverage, the stack trace, system debug logs, and user debug logs. Make sure that your code has better coverage than the code in this sample org!



Class	Percent	Lines
Overall	**19%**	

Debug Logs for Apex Test Results

When you run a test, the output is sent to the log. Debug logs display on the right side of the Force.com IDE's Apex Test Results view.

- The first part of the log details the events that occurred during the test run.
- The next few lines give details on how long it took to execute specific lines of code. This information is useful for performance tuning.
- The debug log then lists how many resources a program uses and the total amount available for each resource. The Apex runtime engine tracks the resources that every script uses so that a single script doesn't monopolize the servers. If you write a script that goes over one of the limits, you receive an error message.

To change your logging levels, create or edit an Apex test run configuration. To access your Apex test run configurations, select **Run** > **Run Configurations** > **Apex Test**. To create a test run configuration, then select **New launch configuration** ( ).

Deploying Code with the Force.com IDE

Once you have created application components and tested them in your own development organization, you usually want to migrate them to a different organization for testing, staging, publication, or production use by end users.

Though each Force.com metadata component lives and runs in a particular organization on Force.com servers, the metadata files you work with in the Force.com IDE are portable from one organization to another. Deploying is the process of pushing your local project files into a different Salesforce organization than your project's home organization. The Force.com IDE provides the **Deploy to Server** wizard to guide you through the deployment process.

Before you deploy, you can streamline your deployment process by creating a new project with only the components you want to deploy. For information, see Create New Force.com Project.

Note: If you deploy to a production organization, you must meet the Apex testing compliance policy: 75% of your Apex must be covered by unit tests, and all of those tests must complete successfully. Additionally, each trigger must have some code coverage.

To open the Deploy to Server wizard, right-click the `src` folder of the project and select **Force.com** > **Deploy to Server**. You can also select individual classes or triggers, multiple classes and triggers, or the `classes` or `triggers` folder. Before the wizard opens, the IDE checks for conflicts between the project and the home organization. If conflicts are found, you are given the option to synchronize before continuing.

The Deploy to Server wizard includes the following pages:

1. On the Destination Details page, enter the connection information for the target organization and click **Next**.
2. On the Archive Options page, optionally create backup files for this deployment and click **Next**.
3. On the Deployment Plan page, review the actions to be performed in this deployment. Click **Next** to execute the deployment.
4. The Deployment Result page displays the result of the deployment. You should see Success displayed at the top of the page. If you do not, review the deployment logs by clicking **View Logs**. If you need assistance with deployment errors, click **Save** to save your log file for future reference.

SEE ALSO:

Destination Details

Archive Options

Deployment Plan

Destination Details

On the Destination Details page, enter the connection settings for the organization you are deploying to.

Field	Description
Username	Required. The username you use to log into the organization you are deploying to. The username associated with this connection must have the "Modify All Data" permission. Typically, this is only enabled for System Administrator users.
Password	Required. The password for the username specified.
Security Token	The security token is appended to your password as an added security measure. If you are using a security token, enter the value

Field	Description
	here. For more information, see the Salesforce online help topic Setting Login Restrictions.
Environment	Choose the appropriate environment for your connection: • **Production/Developer Edition** - Choose this option if you are connecting to a production or Developer Edition organization. • **Sandbox** - Choose this option if you are connecting to a sandbox organization. Sandbox organizations have an URL that starts with `test`. • **Pre-Release** - Choose this option if you have been given the credentials to connect to a prerelease server. • **Other (Specify)** - Choose this option if you want to connect to a specific instance. If you choose this option, you must enter a value in the `Hostname` field.

Archive Options

The Archive Options page allows you to save a snapshot of the project and the destination organization, respectively, before any changes are made. If you are deploying to a production organization, archive your destination organization to facilitate recovery, in the unlikely event that this is needed. The archive options save a snapshot of the project and the destination organization, respectively, before any changes are made. This facilitates recovery if needed. For deployments to production organizations, you should select both archive options.

Field	Description
Project archive	Select to create an archive of your project (local files).
Destination archive	Select to create an archive of the destination files (files on the server).

Deployment Plan

The Deployment Plan page lists every item available for deployment and the action that will be performed. Each type of action is color coded for quick reference. Select the actions you wish to perform in this deployment.

- Add (green) - Adds the component to the destination organization.
- Delete (red) - Deletes the component from the destination organization.
- Overwrite (yellow) - Overwrites the Deployment Plancomponent on the destination organization. This action is available when no differences between the project and server organization for the component are found. This option is not selected by default when files are identical.
- No Action (grey) - Clear the checkbox next to the component to take no action.

The following controls help navigate the available components:

Button	Description
Select All	Click to select to deploy all items.
Deselect All	Click to remove all items from deployment plan.
Refresh Plan	Click to refresh available objects.

Click **Validate Deployment** to check the likely success or failure of the deployment. The Test Deployment Results View displays either a Success or Failure icon along with the reasons for any failures. Click **View Logs** to view details. Close the window to return to the Deployment Plan page.

CHAPTER 6 Getting Started with the Apex Debugger

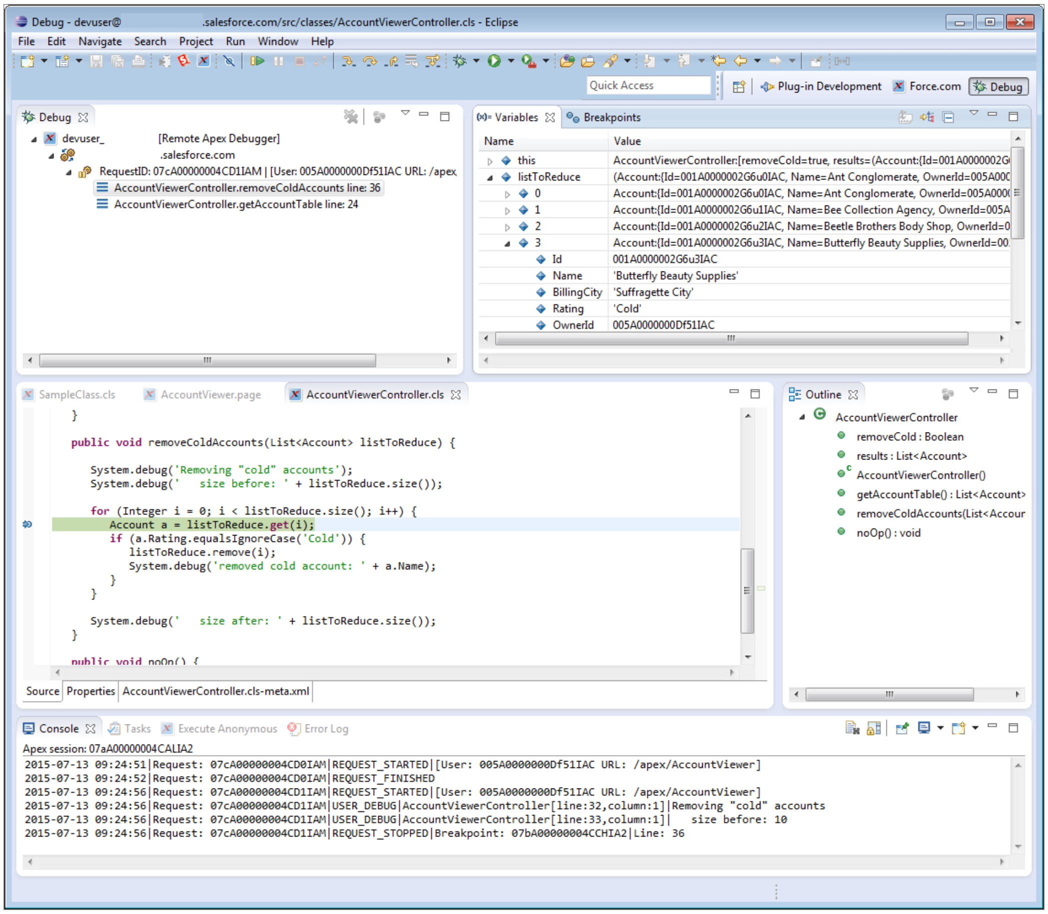
In this chapter ...

Nobody likes the idea of looking for a needle in a haystack—or for a bug in a call stack. We want our tools to facilitate your work and enable your success. And we haven't found the debugging experience at Salesforce any more pleasant than you have. Yes, it's gotten better over time. Years passed. Winter changed into Spring. Spring changed into Summer. Summer changed back into Winter. And gradually the Salesforce debugging experience became less painful. But innovation is in our DNA, and we don't settle for "less bad." So, at Dreamforce '14 we unveiled our gift to you: a real debugger. At Dreamforce '15, we announced that it is generally available. And there was much rejoicing.

Note: Some services and subscriptions include this feature for an extra cost. For pricing details, contact your Salesforce account executive.

The Apex Debugger extends the Force.com IDE plug-in for Eclipse and does most of the things you expect a debugger to do. Use it to:

- Set breakpoints in Apex classes and triggers.
- View variables, including sObject types, collections, and Apex `System` types.
- View the call stack, including triggers activated by Apex Data Manipulation Language (DML), method-to-method calls, and variables.
- Interact with global classes, exceptions, and triggers from your installed managed packages. (When you inspect objects that have managed types that aren't visible to you, only global variables are displayed in the variable inspection pane.)
- Complete standard debugging actions, including step into, over, and out, and run to breakpoint.
- Output your results to the Console window.



Debug - devuser@ .salesforce.com/src/classes/AccountViewerController.cls - Eclipse

Debug

- devuser_ [Remote Apex Debugger]
  - .salesforce.com
    - RequestID: 07cA00000004CD1IAM | [User: 005A0000000Df51IAC URL: /apex,
      - AccountViewerController.removeColdAccounts line: 36
      - AccountViewerController.getAccountTable line: 24

Name	Value
this	AccountViewerController:[removeCold=true, results=(Account:{Id=001A0000002G
listToReduce	(Account:{Id=001A0000002G6u0IAC, Name=Ant Conglomerate, OwnerId=005A000
0	Account:{Id=001A0000002G6u0IAC, Name=Ant Conglomerate, OwnerId=005A000
1	Account:{Id=001A0000002G6u1IAC, Name=Bee Collection Agency, OwnerId=005A
2	Account:{Id=001A0000002G6u2IAC, Name=Beetle Brothers Body Shop, OwnerId=0
3	Account:{Id=001A0000002G6u3IAC, Name=Butterfly Beauty Supplies, OwnerId=00
Id	001A0000002G6u3IAC
Name	'Butterfly Beauty Supplies'
BillingCity	'Suffragette City'
Rating	'Cold'
OwnerId	005A0000000Df51IAC

```
    }

    public void removeColdAccounts(List<Account> listToReduce) {

        System.debug('Removing "cold" accounts');
        System.debug('   size before: ' + listToReduce.size());

        for (Integer i = 0; i < listToReduce.size(); i++) {
            Account a = listToReduce.get(i);
            if (a.Rating.equalsIgnoreCase('Cold')) {
                listToReduce.remove(i);
                System.debug('removed cold account: ' + a.Name);
            }
        }

        System.debug('   size after: ' + listToReduce.size());
    }

    public void noOp() {
```

Outline

- AccountViewerController
  - removeCold : Boolean
  - results : List<Account>
  - AccountViewerController()
  - getAccountTable() : List<Account>
  - removeColdAccounts(List<Accour
  - noOp() : void

Apex session: 07aA00000004CALIA2

```
2015-07-13 09:24:51|Request: 07cA00000004CD0IAM|REQUEST_STARTED|[User: 005A0000000Df51IAC URL: /apex/AccountViewer]
2015-07-13 09:24:52|Request: 07cA00000004CD0IAM|REQUEST_FINISHED
2015-07-13 09:24:56|Request: 07cA00000004CD1IAM|REQUEST_STARTED|[User: 005A0000000Df51IAC URL: /apex/AccountViewer]
2015-07-13 09:24:56|Request: 07cA00000004CD1IAM|USER_DEBUG|AccountViewerController[line:32,column:1]|Removing "cold" accounts
2015-07-13 09:24:56|Request: 07cA00000004CD1IAM|USER_DEBUG|AccountViewerController[line:33,column:1]|   size before: 10
2015-07-13 09:24:56|Request: 07cA00000004CD1IAM|REQUEST_STOPPED|Breakpoint: 07bA00000004CCHIA2|Line: 36
```

Summary of Getting Started with the Apex Debugger

This documentation exists to help you, a Force.com developer, get started with the Apex Debugger for Eclipse. Step through the process of setting up the Debugger. Then, explore a simple debugging puzzle and start thinking about how to debug your projects. Be sure to check out the limits and considerations, too. Finally, learn about some common problems you might encounter and how you can get over those hurdles.

Set Up the Apex Debugger

Complete these tasks to get the Apex Debugger ready for use in your Salesforce org and on your workstation.

IN THIS SECTION:

Contact Salesforce to Enable the Apex Debugger

Some services and subscriptions include this feature for an extra cost. For pricing details, contact your Salesforce account executive.

Install or Update the Force.com IDE Plug-In for Eclipse

The Apex Debugger is part of the Force.com IDE plug-in for Eclipse. Before you set up the Debugger, install or update the Force.com IDE.

Ensure that the Prerequisites listed on the Force.com IDE Installation page are installed on your workstation.

1. In Eclipse, choose **Help** > **Install New Software**.
2. Click **Add**, and add this update site. `https://developer.salesforce.com/media/force-ide/eclipse42/`
3. In the Name and Version list, select all items named Force.com IDE and Force.com IDE Debugger.
4. Click **Next** and proceed with the installation.

Set Up a Permission Set

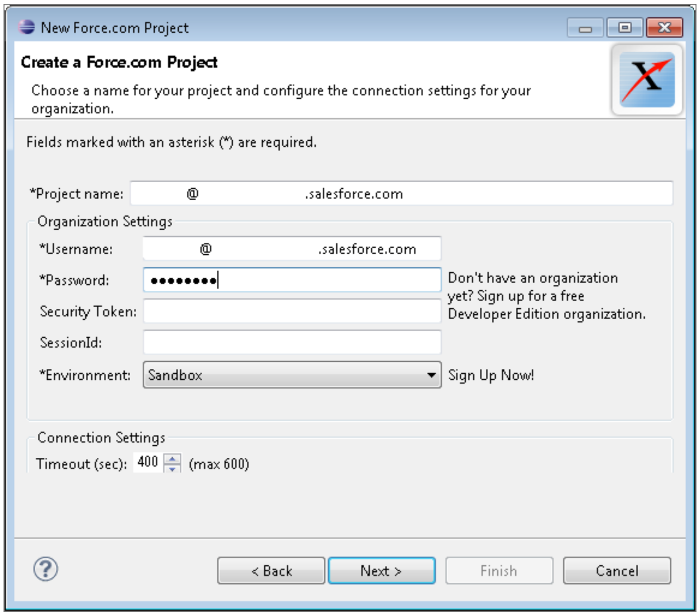
Create a permission set for Apex Debugger users, and assign it to any users in your org who plan to use the Debugger.

1. Log in to your org.
2. From Setup, enter `Permission Sets` in the `Quick Find` box, then click **Permission Sets**.
3. Create a permission set. Give it a name that you can remember, such as Debug Apex.
4. In the “Select the type of users who will use this permission set” section, choose **None** from the User License drop-down list. Choosing None lets you assign the permission set to more than one type of user.
5. Save your changes.
6. Click **System Permissions**.
7. Click **Edit**.
8. Enable the Debug Apex permission.
9. Save your changes.
10. Click **Manage Assignments**.
11. Click **Add Assignments**.
12. Select the users to whom you want to assign the permission set, and then click **Assign**.

Create a Project

If you’re new to using the Force.com IDE, set up a project. If you plan to debug an existing Force.com IDE project, skip these steps.

1. In Eclipse, choose **File** > **New** > **Project** > **Force.com** > **Force.com Project** > **Next**.
2. Enter a descriptive project name and your Salesforce sandbox org’s credentials. Choose **Sandbox** from the Environment drop-down menu.



New Force.com Project

Create a Force.com Project

Choose a name for your project and configure the connection settings for your organization.

Fields marked with an asterisk (*) are required.

*Project name: @ .salesforce.com

Organization Settings

*Username: @ .salesforce.com

*Password: ••••••••

Security Token:

SessionId:

*Environment: Sandbox

Don't have an organization yet? Sign up for a free Developer Edition organization.

Sign Up Now!

Connection Settings

Timeout (sec): 400 (max 600)

3. Click **Next**, and then complete the New Force.com Project wizard.
4. To save changes to the server automatically, right-click the name of the project that you created, then select **Force.com** > **Work Online**. By default, your changes are saved only locally, but Work Online is the recommended setting for using the Apex Debugger.

   When Work Online is enabled, the icon next to your project name looks like this: 📂 , rather than like this: 📂 .

Test Your Debugger Setup

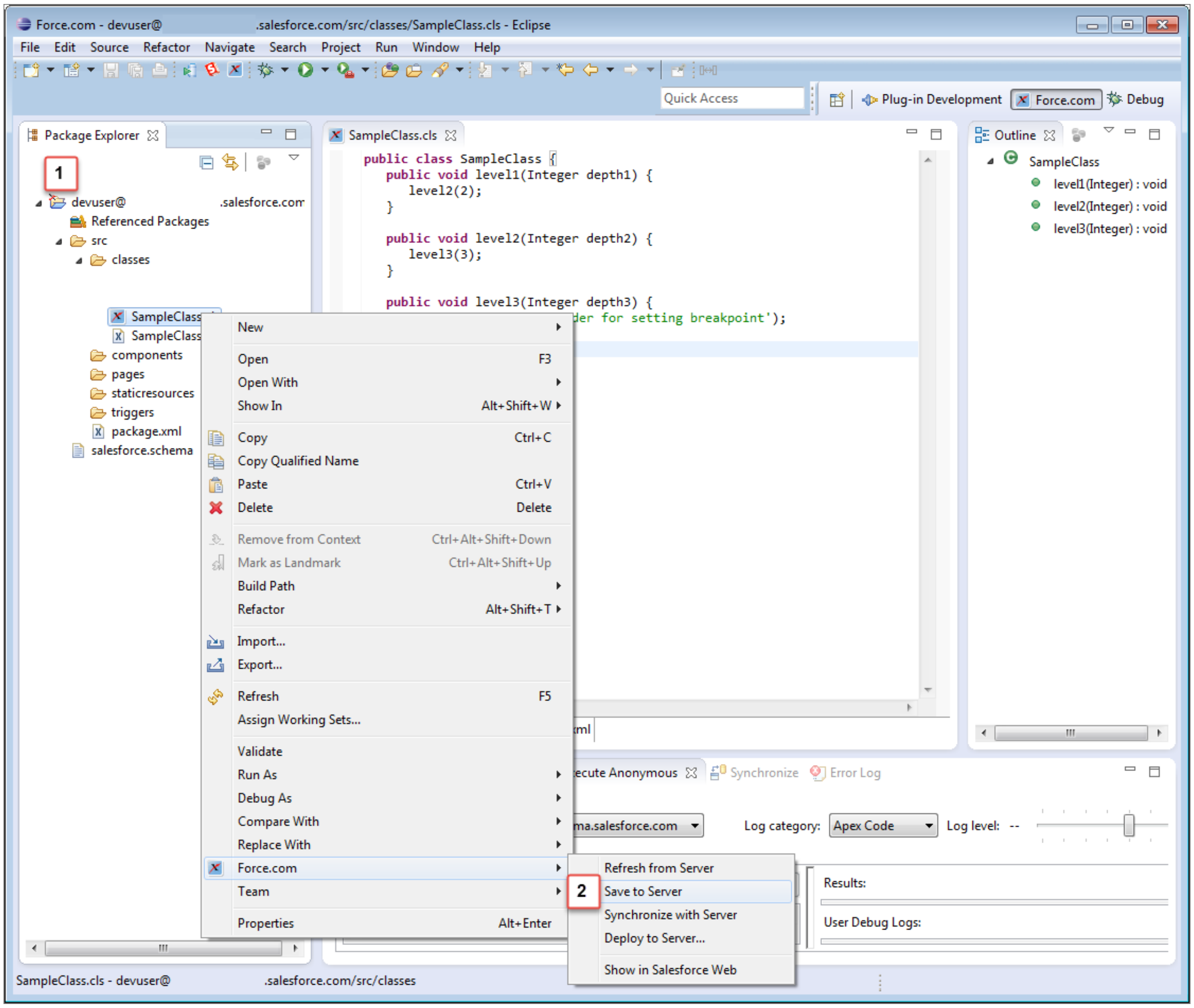
To make sure that everything is functioning properly, create a simple Apex class, set a breakpoint in your code, and then hit the breakpoint using Execute Anonymous.

1. Create an Apex class called `SampleClass` with this content.

```
public class SampleClass {
   public void level1(Integer depth1) {
      level2(2);
   }

   public void level2(Integer depth2) {
      level3(3);
   }

   public void level3(Integer depth3) {
      System.debug('Placeholder for setting breakpoint');
   }
}
```

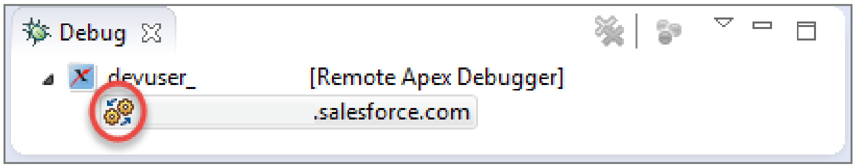
2. Save the class.

3. If you don't have Work Online enabled, save the class to the server. Right-click the class in the Package Explorer, then choose **Force.com** > **Save to Server**. This screenshot shows that Work Online is not enabled (1), and demonstrates how to save to the server (2).

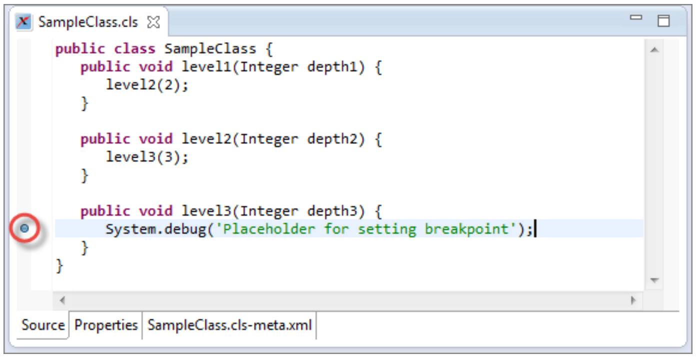


```
public class SampleClass {
    public void level1(Integer depth1) {
        level2(2);
    }

    public void level2(Integer depth2) {
        level3(3);
    }

    public void level3(Integer depth3) {
                          der for setting breakpoint');
```

4. Switch to the Debug perspective by choosing **Window** > **Open Perspective** > **Other** > **Debug**. After you've completed the initial Debugger setup, you can switch between perspectives by clicking the icons in the upper right corner of your Eclipse window.
5. Click the debug icon ( ) in the toolbar and select **Debug Configurations**.
6. Select **Remote Apex Debugger**.
7. To create a configuration for your project, click the New launch configuration icon ( ).
8. Name the configuration.
9. Click **Browse** and select your project.
10. Click **Apply**, and then click **Debug**.
11. Click the debug icon ( ) and launch the new debug configuration.
12. Wait until you see an icon showing turning yellow gears in the Debug pane.



13. To set a breakpoint, double-click in the gray gutter to the left of the `System.debug` statement in `SampleClass.cls`.



```
public class SampleClass {
   public void level1(Integer depth1) {
      level2(2);
   }

   public void level2(Integer depth2) {
      level3(3);
   }

   public void level3(Integer depth3) {
      System.debug('Placeholder for setting breakpoint');
   }
}
```

14. Select the **Execute Anonymous** tab at the bottom of the screen. If the Execute Anonymous tab isn't available, choose **Window** > **Show View** > **Other** > **Force.com** and add it.

15. Make sure that your project is the Active Project.

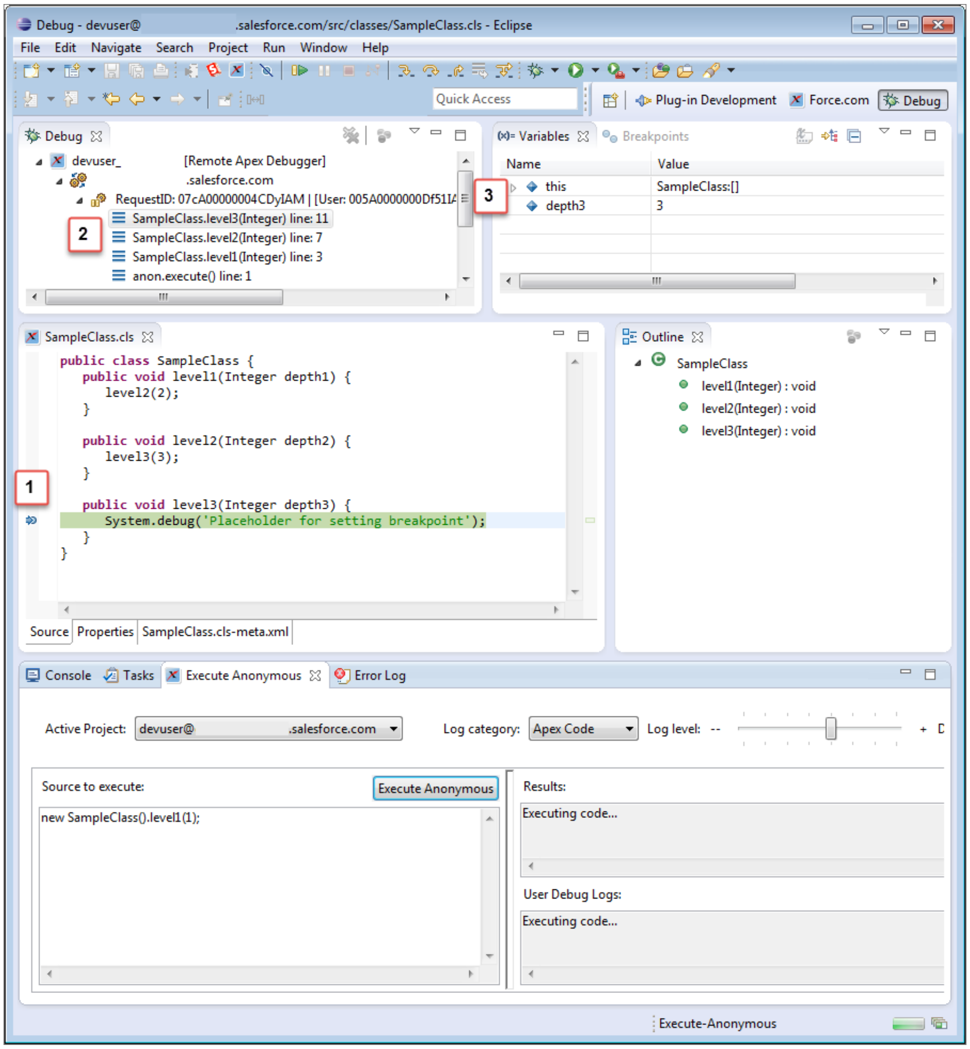
16. In the Source to execute field, enter:

```
new SampleClass().level1(1);
```

17. Click **Execute Anonymous**.

Note: You might get an error message like, "SocketTimeoutException: Read timed out," when using Execute Anonymous to hit a breakpoint during a debugging session. Execute Anonymous expects to receive timely results from the server, but because you've stopped at a breakpoint your results don't arrive in a timely fashion. Dismiss the error message, and carry on with your debugging.

The Debugger stops at the breakpoint (1). The call stack (2) and the values of your variables (3) are displayed. When applicable, a URL showing how each request originated displays next to the request ID in your call stack details.



```
devuser_            [Remote Apex Debugger]
  .salesforce.com
    RequestID: 07cA00000004CDyIAM | [User: 005A0000000Df51IA
      SampleClass.level3(Integer) line: 11
      SampleClass.level2(Integer) line: 7
      SampleClass.level1(Integer) line: 3
      anon.execute() line: 1
```

Name	Value
this	SampleClass:[]
depth3	3

```
public class SampleClass {
    public void level1(Integer depth1) {
        level2(2);
    }

    public void level2(Integer depth2) {
        level3(3);
    }

    public void level3(Integer depth3) {
        System.debug('Placeholder for setting breakpoint');
    }
}
```

SampleClass
- level1(Integer) : void
- level2(Integer) : void
- level3(Integer) : void

Active Project: devuser@ .salesforce.com Log category: Apex Code Log level:

Source to execute:

```
new SampleClass().level1(1);
```

Results:

```
Executing code...
```

User Debug Logs:

```
Executing code...
```

Explore a Simple Debugging Puzzle

Once you've gotten the Apex Debugger up and running, work through this exercise to explore some of the Debugger's capabilities.

IN THIS SECTION:

Create Sample Accounts in Your Org

To complete this exercise, add the following sample data in your org.

Create an Apex Class

Next, let's create an Apex class to debug. This class removes cold accounts from a list so that you can focus on more promising clients.

Create a Visualforce Page

Now we need a Visualforce page to bring the `AccountViewer` controller to life.

Identify a Problem

Because we never write buggy code, our Visualforce page should be working perfectly! Let's make sure.

Debug the Problem
Let’s use the Apex Debugger to see what went wrong.

Fix the Problem
By now you’ve discovered that we need to decrement the iterator `i` when removing an item from `listToReduce`. But we can’t save changes to our code while a debugging session is in process. We first terminate our debugging session, then fix the code.

Delete Your Sample Accounts
Unless you want to keep your sample accounts so that you can play with them in the future, run this code to delete them.

Create Sample Accounts in Your Org

To complete this exercise, add the following sample data in your org.

1. Enter this code in the Source to execute field of the Force.com IDE’s Execute Anonymous pane.

```
List<String> acctNames = new List<String>{
    'Ant Conglomerate',
    'Bee Collection Agency',
    'Beetle Brothers Body Shop',
    'Butterfly Beauty Supplies',
    'Flea LLC',
    'Fly Airlines',
    'Moth Candle Company',
    'Tick Timepieces',
    'Wasp Industrial Products',
    'Weevil Consultancy'
    };


List<Account> newAccts = new List<Account>();


for(Integer i = 0; i < 10; i++) {
    Account newAcct = new Account();
    newAcct.name = acctNames.get(i);
    newAcct.BillingCity = 'Suffragette City';
    newAccts.add(newAcct);
}


newAccts.get(0).rating = 'Warm';
newAccts.get(1).rating = 'Cold';
newAccts.get(2).rating = 'Hot';
newAccts.get(3).rating = 'Cold';
newAccts.get(4).rating = 'Cold';
newAccts.get(5).rating = 'Warm';
newAccts.get(6).rating = 'Hot';
newAccts.get(7).rating = 'Hot';
newAccts.get(8).rating = 'Cold';
newAccts.get(9).rating = 'Warm';


for (Integer i = 0; i < 10; i++) System.debug(newAccts.get(i));
insert newAccts;
```

2. Click **Execute Anonymous**.

   Note: If you get a compile error after executing this code, make sure that `Account.rating` is visible to your user. To give yourself access to the Account object's Rating field, from Setup, enter *`Field`* in the `Quick Find` box, then click **Account** > **Fields** > **Rating** > **Set Field-Level Security** or **Field Accessibility** > **Account** > **View by Fields** > **Rating**.

3. In the Accounts tab of the Salesforce user interface, verify that your accounts have been created.
   Congratulations. Your org now contains all the data that you need to complete the debugging exercise.

Create an Apex Class

Next, let's create an Apex class to debug. This class removes cold accounts from a list so that you can focus on more promising clients.

1. Create an Apex class called `AccountViewerController`.
2. Replace your class's default contents with this code.

```
public class AccountViewerController {


   public Boolean removeCold { get; set; }
   public List<Account> results { get; set; }


   public AccountViewerController() {


      removeCold = false;


      results = [SELECT Id, Name, Owner.Name, Rating, BillingCity, BillingState
         from Account
         WHERE BillingCity = 'Suffragette City'
         Order By Name ASC];
   }


   public List<Account> getAccountTable() {


      List<Account> accountsToReturn;
      accountsToReturn = new List<Account>();
      accountsToReturn.addAll(results);


      if (removeCold==true) {
         removeColdAccounts(accountsToReturn);
      }


      return accountsToReturn;
   }


   public void removeColdAccounts(List<Account> listToReduce) {
      System.debug('Removing "cold" accounts');
      System.debug('   size before: ' + listToReduce.size());


      for (Integer i = 0; i < listToReduce.size(); i++) {
         Account a = listToReduce.get(i);
         if (a.Rating.equalsIgnoreCase('Cold')) {
            listToReduce.remove(i);
            System.debug('removed cold account: ' + a.Name);
         }
      }


      System.debug('   size after: ' + listToReduce.size());
   }


   public void noOp() {
   }


}
```

3. Save `AccountViewerController.cls`.

Create a Visualforce Page

Now we need a Visualforce page to bring the `AccountViewer` controller to life.

1. Create a Visualforce page with the label `Account Viewer` and the name `AccountViewer`.
2. Replace your page's default contents with this code.

```
<apex:page controller="AccountViewerController">
   <apex:form>
      <apex:outputPanel id="resultTable">
         <apex:pageBlock>
            <apex:actionstatus id="status">
               <apex:facet name="start">
                  <div class="waitingSearchDiv" id="el_loading" style=
                        "background-color: #fbfbfb; height: 100%; opacity:0.65;
                        width:100%;">
                     <div class="waitingHolder" style="top: 74.2px; width: 91px;">
                        <img class="waitingImage" src="/img/loading.gif" title=
                           "Please Wait..." />
                        <span class="waitingDescription">Please Wait...</span>
                     </div>
                  </div>
               </apex:facet>
            </apex:actionstatus>
            <apex:pageBlockSection title="Special Accounts" collapsible="false">
               <apex:inputCheckbox value="{!removeCold}" label="Hide Cold Accounts">
                  <apex:actionSupport event="onchange" action="{!noOp}"
```

```
                     status="status" rerender="resultTable"/>
               </apex:inputCheckbox>
            </apex:pageBlockSection>
            <apex:pageBlockSection title="Scheduled Jobs" collapsible="false">
               <apex:pageBlockTable value="{!accountTable}" var="a"
                  id="thePageBlockTable">
                  <apex:column style="vertical-align:top">
                     <apex:outputField value="{!a.name}" />
                     <apex:facet name="header">Name</apex:facet>
                  </apex:column>
                  <apex:column>
                     <apex:outputField value="{!a.BillingCity}" />
                     <apex:facet name="header">City</apex:facet>
                  </apex:column>
                  <apex:column>
                     <apex:outputField value="{!a.BillingState}" />
                     <apex:facet name="header">State</apex:facet>
                  </apex:column>
                  <apex:column>
                     <apex:outputField value="{!a.Rating}" />
                     <apex:facet name="header">Rating</apex:facet>
                  </apex:column>
               </apex:pageBlockTable>
            </apex:pageBlockSection>
         </apex:pageBlock>
      </apex:outputPanel>
   </apex:form>
</apex:page>
```

3. Save `AccountViewer.page`.

Identify a Problem

Because we never write buggy code, our Visualforce page should be working perfectly! Let’s make sure.

1. Make sure that you’re logged in to your org.
2. Navigate to `https://`*`your_salesforce_instance`*`/apex/AccountViewer`.
3. Notice that your Special Accounts list includes cold accounts. Their presence is unacceptable! Select **Hide Cold Accounts**.
4. They're all gone, right? Wrong. What happened? It's time for debugging!

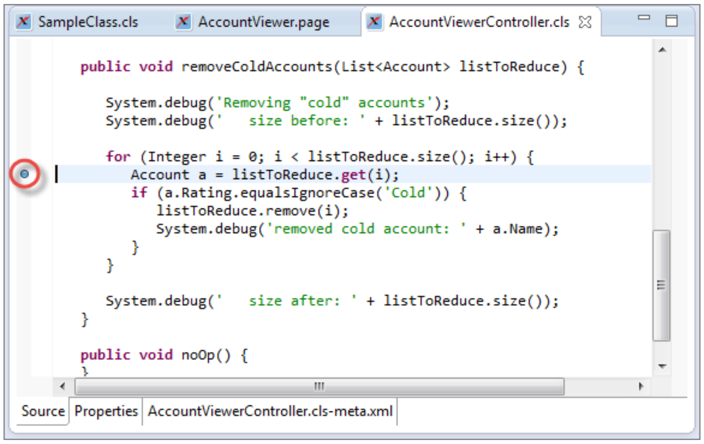
Debug the Problem

Let’s use the Apex Debugger to see what went wrong.

1. Make sure that `AccountViewerController.cls` is open.
2. Switch to the Debug perspective of the Force.com IDE.
3. Make sure that the turning yellow gears (⚙) are visible in the Debug pane.
4. Find the `removeColdAccounts(List<Account> listToReduce)` method.

5. Set a breakpoint on this line:

```
Account a = listToReduce.get(i);
```

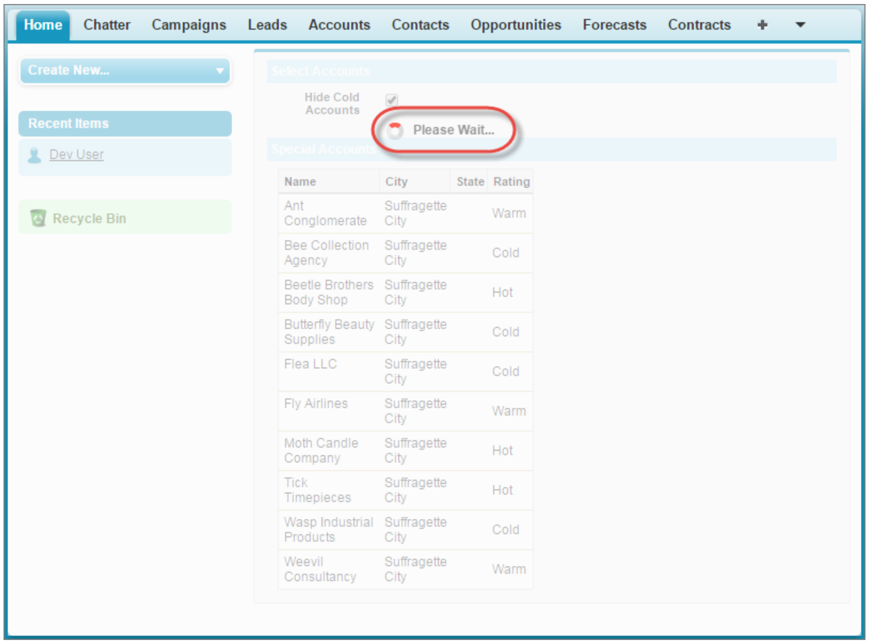

SampleClass.cls
AccountViewer.page
AccountViewerController.cls
Source
Properties
AccountViewerController.cls-meta.xml

```
    public void removeColdAccounts(List<Account> listToReduce) {

       System.debug('Removing "cold" accounts');
       System.debug('   size before: ' + listToReduce.size());

       for (Integer i = 0; i < listToReduce.size(); i++) {
          Account a = listToReduce.get(i);
          if (a.Rating.equalsIgnoreCase('Cold')) {
             listToReduce.remove(i);
             System.debug('removed cold account: ' + a.Name);
          }
       }

       System.debug('   size after: ' + listToReduce.size());
    }

    public void noOp() {
    }
```

6. Navigate to `https://`*`your_salesforce_instance`*`/apex/AccountViewer`.

7. Select **Hide Cold Accounts**.

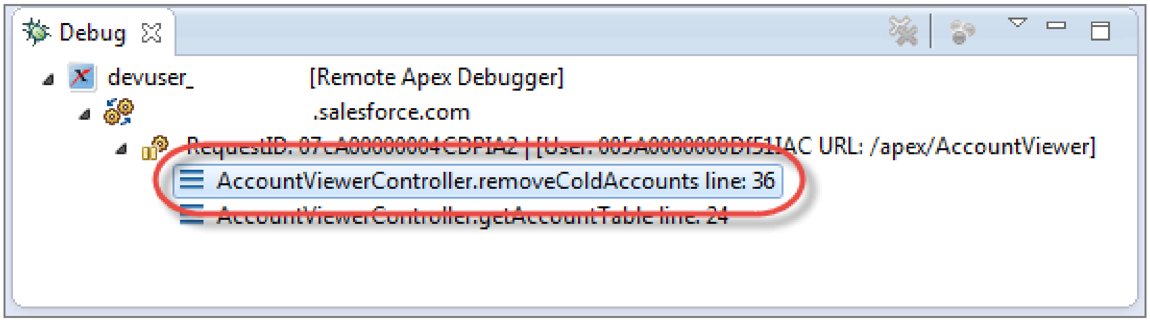
8. Notice that Please Wait has been displaying for awhile. Execution has stopped because you've successfully hit a breakpoint.



Hide Cold Accounts ☑

Name	City	State	Rating
Ant Conglomerate	Suffragette City		Warm
Bee Collection Agency	Suffragette City		Cold
Beetle Brothers Body Shop	Suffragette City		Hot
Butterfly Beauty Supplies	Suffragette City		Cold
Flea LLC	Suffragette City		Cold
Fly Airlines	Suffragette City		Warm
Moth Candle Company	Suffragette City		Hot
Tick Timepieces	Suffragette City		Hot
Wasp Industrial Products	Suffragette City		Cold
Weevil Consultancy	Suffragette City		Warm

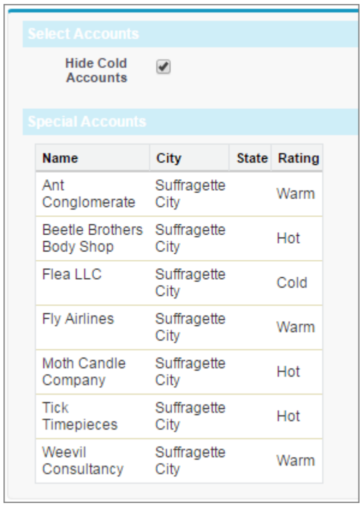
9. In Eclipse, click the line in the Debug pane's stack trace that corresponds to your breakpoint.



devuser_ [Remote Apex Debugger]
.salesforce.com
RequestID: 07cA00000004CDPIA2 | [User: 005A0000000Df51IAC URL: /apex/AccountViewer]
AccountViewerController.removeColdAccounts line: 36
AccountViewerController.getAccountTable line: 24

10. Click Step Over ( ) until you've stepped through all the iterations of your for loop. As you do so, pay attention to the values of the variables in your Variables pane. If you notice the problem, try to correct it. If you manage to correct the problem, go ahead and skip to Fix the Problem on page 55 to confirm your solution.

11. After you've stepped all the way through the for loop, look at your Visualforce page. Some of the cold accounts were hidden, but at least one remains.



Select Accounts

Hide Cold Accounts ☑

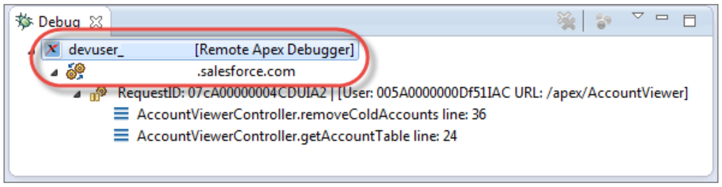
Special Accounts

Name	City	State	Rating
Ant Conglomerate	Suffragette City		Warm
Beetle Brothers Body Shop	Suffragette City		Hot
Flea LLC	Suffragette City		Cold
Fly Airlines	Suffragette City		Warm
Moth Candle Company	Suffragette City		Hot
Tick Timepieces	Suffragette City		Hot
Weevil Consultancy	Suffragette City		Warm

12. In this screenshot, the cold account is Flea LLC. Make a note of this account's name. Then, let's step through our for loop again.

13. Deselect **Hide Cold Accounts**.

14. In Eclipse, click one of the top two levels of the stack trace in the Debug pane.



devuser_ [Remote Apex Debugger]
.salesforce.com
RequestID: 07cA00000004CDUIA2 | [User: 005A0000000Df51IAC URL: /apex/AccountViewer]
AccountViewerController.removeColdAccounts line: 36
AccountViewerController.getAccountTable line: 24

15. To stop your debugging session, click the terminate icon ( ).

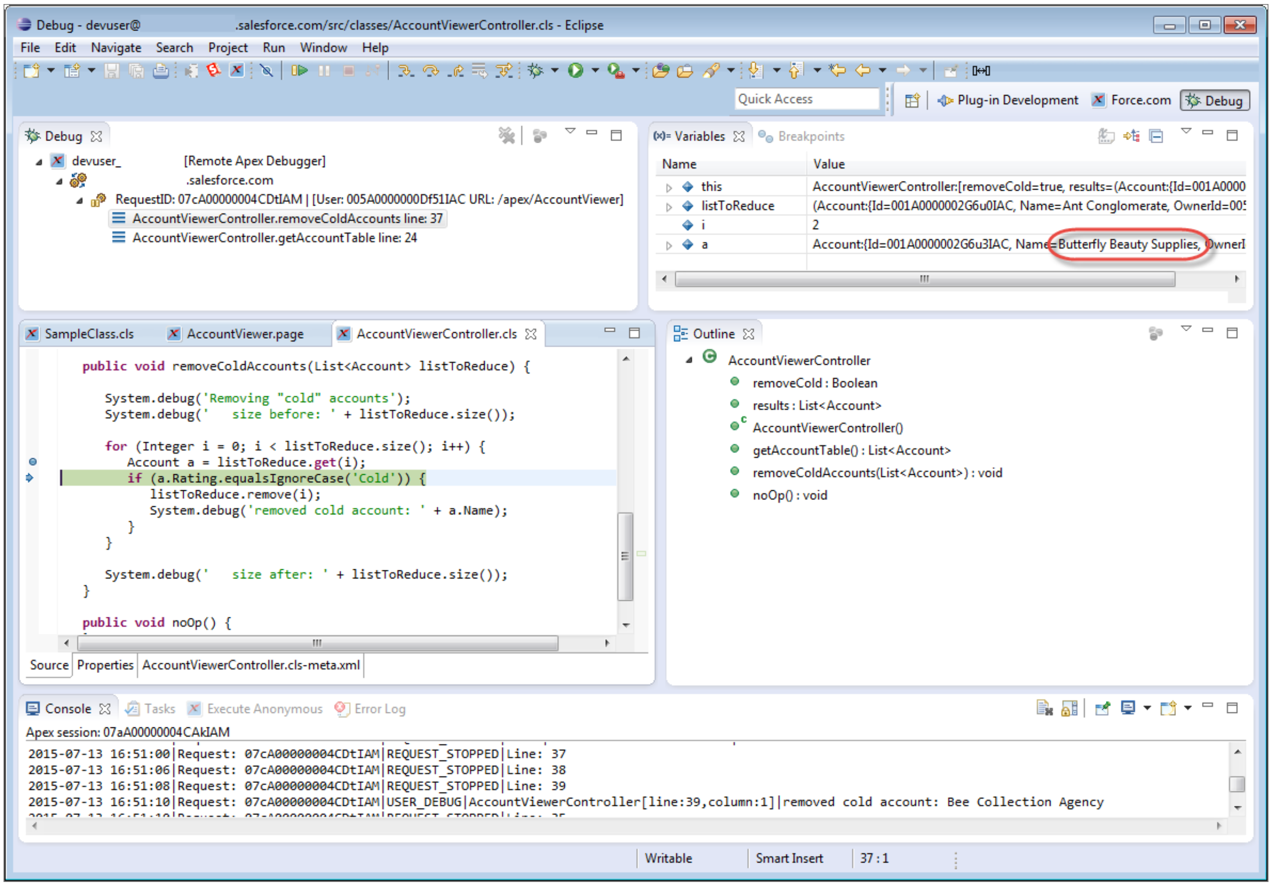
16. To start a new session, click the debug icon ( ).

17. Wait for the turning yellow gears ( ) to reappear in the Debug pane.

18. Reload your Visualforce page, then select **Hide Cold Accounts** again.

19. This time, pay special attention to what happens when you move past the account right before Flea LLC. Make a note of the account that you want to watch out for. In this case, the account we want to watch for is Butterfly Beauty Supplies.

20. Click Step Over ( ) until you see that the value of *`a.Name`* is the account name you're watching for—in this case, `Butterfly Beauty Supplies`.


Debug - devuser@ .salesforce.com/src/classes/AccountViewerController.cls - Eclipse
File Edit Navigate Search Project Run Window Help
Quick Access
Plug-in Development
Force.com
Debug

Debug

- devuser_ [Remote Apex Debugger]
  - .salesforce.com
    - RequestID: 07cA00000004CDtIAM | [User: 005A0000000Df51IAC URL: /apex/AccountViewer]
      - AccountViewerController.removeColdAccounts line: 37
      - AccountViewerController.getAccountTable line: 24

Variables | Breakpoints

Name	Value
this	AccountViewerController:[removeCold=true, results=(Account:{Id=001A0000
listToReduce	(Account:{Id=001A0000002G6u0IAC, Name=Ant Conglomerate, OwnerId=005
i	2
a	Account:{Id=001A0000002G6u3IAC, Name=Butterfly Beauty Supplies, OwnerI

SampleClass.cls | AccountViewer.page | AccountViewerController.cls

```
    public void removeColdAccounts(List<Account> listToReduce) {

       System.debug('Removing "cold" accounts');
       System.debug('   size before: ' + listToReduce.size());

       for (Integer i = 0; i < listToReduce.size(); i++) {
          Account a = listToReduce.get(i);
          if (a.Rating.equalsIgnoreCase('Cold')) {
             listToReduce.remove(i);
             System.debug('removed cold account: ' + a.Name);
          }
       }

       System.debug('   size after: ' + listToReduce.size());
    }

    public void noOp() {
```

Source | Properties | AccountViewerController.cls-meta.xml

Outline

- AccountViewerController
  - removeCold : Boolean
  - results : List<Account>
  - AccountViewerController()
  - getAccountTable() : List<Account>
  - removeColdAccounts(List<Account>) : void
  - noOp() : void

Console | Tasks | Execute Anonymous | Error Log

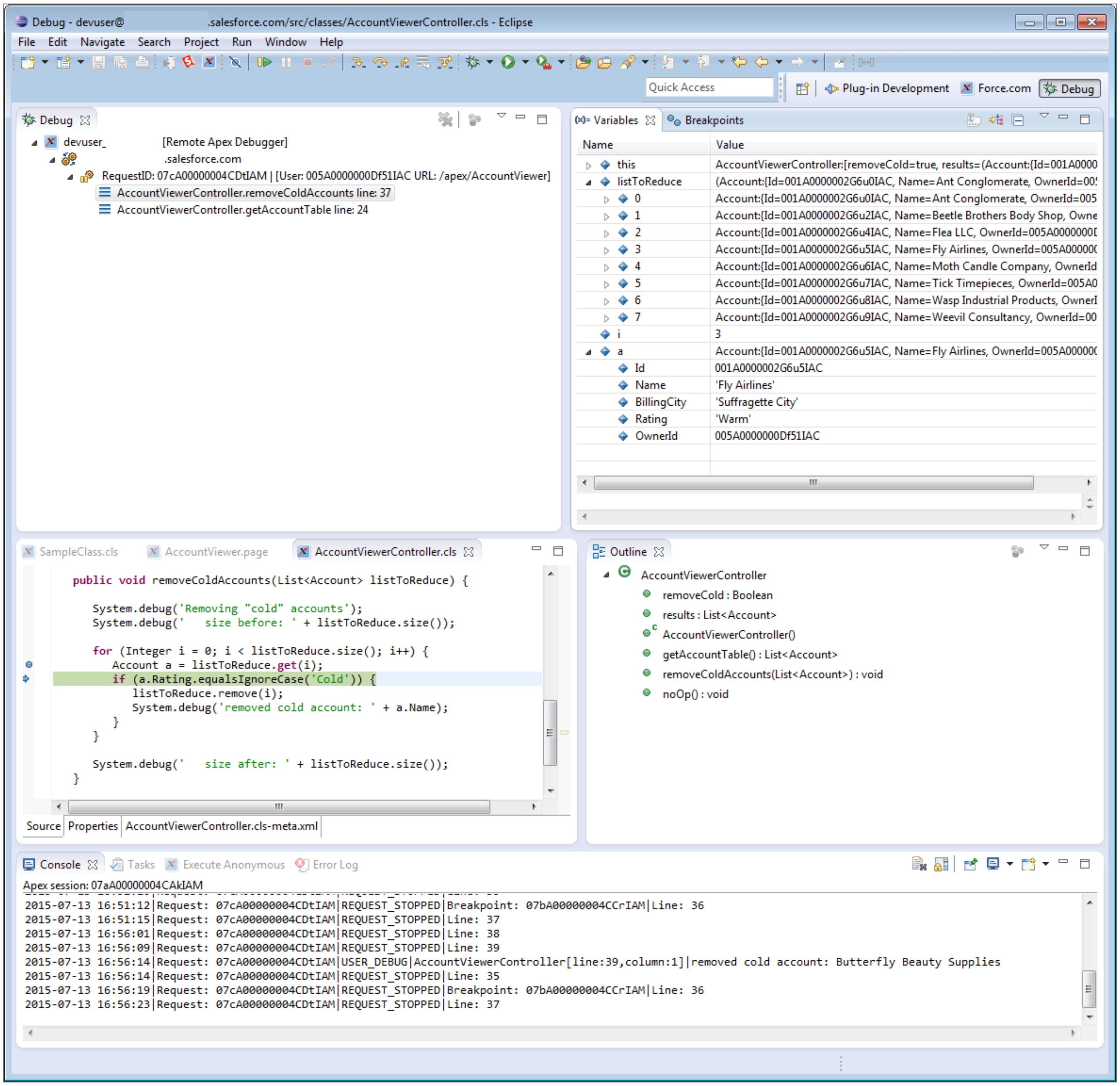
Apex session: 07aA00000004CAkIAM

```
2015-07-13 16:51:00|Request: 07cA00000004CDtIAM|REQUEST_STOPPED|Line: 37
2015-07-13 16:51:06|Request: 07cA00000004CDtIAM|REQUEST_STOPPED|Line: 38
2015-07-13 16:51:08|Request: 07cA00000004CDtIAM|REQUEST_STOPPED|Line: 39
2015-07-13 16:51:10|Request: 07cA00000004CDtIAM|USER_DEBUG|AccountViewerController[line:39,column:1]|removed cold account: Bee Collection Agency
```

Writable | Smart Insert | 37 : 1

21. Paying close attention to the values of your variables, click **Step Over** until *a*'s value changes again.

22. Look at the value of *`listToReduce`*. `'Butterfly Beauty Supplies'` was removed, but we've skipped right over `'Flea LLC'`. Why do you think that could be?


Debug - devuser@ .salesforce.com/src/classes/AccountViewerController.cls - Eclipse
File Edit Navigate Search Project Run Window Help
Quick Access
Plug-in Development
Force.com
Debug
Debug
Variables
Breakpoints
SampleClass.cls
AccountViewer.page
AccountViewerController.cls
Outline
Source
Properties
AccountViewerController.cls-meta.xml
Console
Tasks
Execute Anonymous
Error Log

```
devuser_            [Remote Apex Debugger]
    .salesforce.com
        RequestID: 07cA00000004CDtIAM | [User: 005A0000000Df51IAC URL: /apex/AccountViewer]
            AccountViewerController.removeColdAccounts line: 37
            AccountViewerController.getAccountTable line: 24
```

Name	Value
this	AccountViewerController:[removeCold=true, results=(Account:{Id=001A0000
listToReduce	(Account:{Id=001A0000002G6u0IAC, Name=Ant Conglomerate, OwnerId=00!
0	Account:{Id=001A0000002G6u0IAC, Name=Ant Conglomerate, OwnerId=005
1	Account:{Id=001A0000002G6u2IAC, Name=Beetle Brothers Body Shop, Owne
2	Account:{Id=001A0000002G6u4IAC, Name=Flea LLC, OwnerId=005A0000000I
3	Account:{Id=001A0000002G6u5IAC, Name=Fly Airlines, OwnerId=005A00000(
4	Account:{Id=001A0000002G6u6IAC, Name=Moth Candle Company, OwnerId
5	Account:{Id=001A0000002G6u7IAC, Name=Tick Timepieces, OwnerId=005A0
6	Account:{Id=001A0000002G6u8IAC, Name=Wasp Industrial Products, OwnerI
7	Account:{Id=001A0000002G6u9IAC, Name=Weevil Consultancy, OwnerId=00
i	3
a	Account:{Id=001A0000002G6u5IAC, Name=Fly Airlines, OwnerId=005A00000(
Id	001A0000002G6u5IAC
Name	'Fly Airlines'
BillingCity	'Suffragette City'
Rating	'Warm'
OwnerId	005A0000000Df51IAC

```
public void removeColdAccounts(List<Account> listToReduce) {

   System.debug('Removing "cold" accounts');
   System.debug('   size before: ' + listToReduce.size());

   for (Integer i = 0; i < listToReduce.size(); i++) {
      Account a = listToReduce.get(i);
      if (a.Rating.equalsIgnoreCase('Cold')) {
         listToReduce.remove(i);
         System.debug('removed cold account: ' + a.Name);
      }
   }

   System.debug('   size after: ' + listToReduce.size());
}
```

```
AccountViewerController
    removeCold : Boolean
    results : List<Account>
    AccountViewerController()
    getAccountTable() : List<Account>
    removeColdAccounts(List<Account>) : void
    noOp() : void
```

Apex session: 07aA00000004CAkIAM

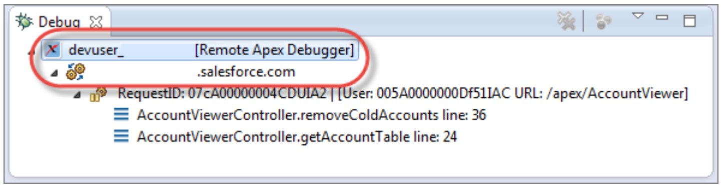
```
2015-07-13 16:51:12|Request: 07cA00000004CDtIAM|REQUEST_STOPPED|Breakpoint: 07bA00000004CCrIAM|Line: 36
2015-07-13 16:51:15|Request: 07cA00000004CDtIAM|REQUEST_STOPPED|Line: 37
2015-07-13 16:56:01|Request: 07cA00000004CDtIAM|REQUEST_STOPPED|Line: 38
2015-07-13 16:56:09|Request: 07cA00000004CDtIAM|REQUEST_STOPPED|Line: 39
2015-07-13 16:56:14|Request: 07cA00000004CDtIAM|USER_DEBUG|AccountViewerController[line:39,column:1]|removed cold account: Butterfly Beauty Supplies
2015-07-13 16:56:14|Request: 07cA00000004CDtIAM|REQUEST_STOPPED|Line: 35
2015-07-13 16:56:19|Request: 07cA00000004CDtIAM|REQUEST_STOPPED|Breakpoint: 07bA00000004CCrIAM|Line: 36
2015-07-13 16:56:23|Request: 07cA00000004CDtIAM|REQUEST_STOPPED|Line: 37
```

The problem lies in the value of the iterator, `i`. When we removed `'Butterfly Beauty Supplies'` from the list, we didn't account for the fact that removing this list item would change the positions of the subsequent items. Flea LLC is now at position `2`, formerly occupied by Butterfly Beauty Supplies, but that position has already been processed. Fortunately, this problem is easily fixed.

Fix the Problem

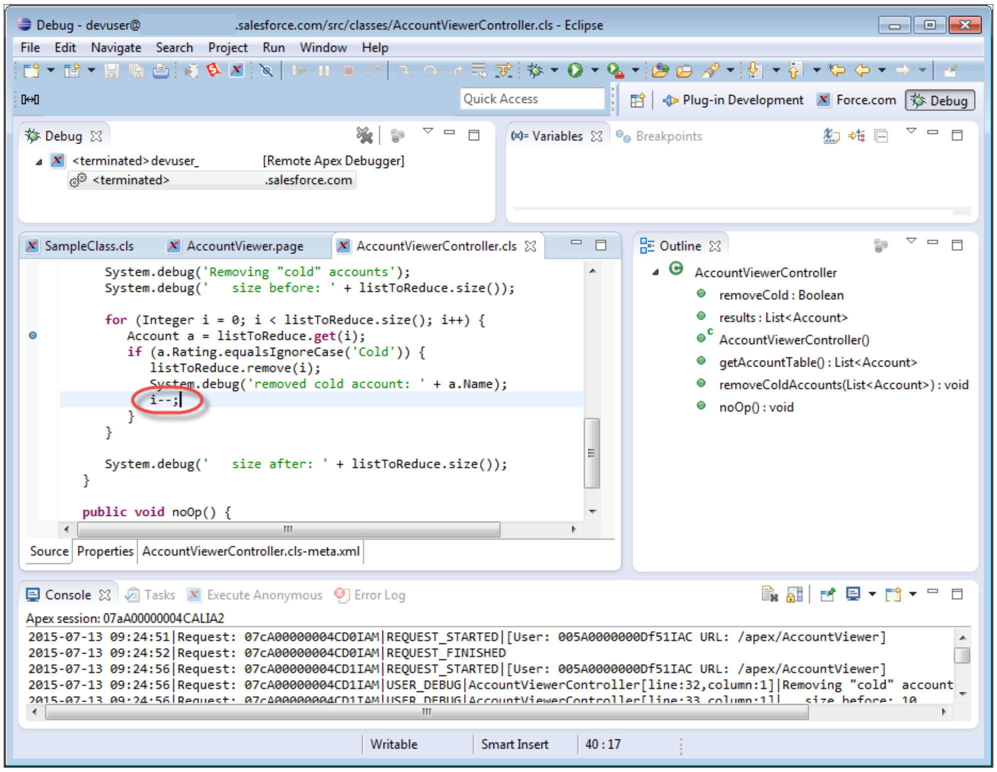
By now you've discovered that we need to decrement the iterator `i` when removing an item from `listToReduce`. But we can't save changes to our code while a debugging session is in process. We first terminate our debugging session, then fix the code.

1. In the Debug perspective of the Force.com IDE, click one of the top two levels of the stack trace in the Debug pane.



```
devuser_            [Remote Apex Debugger]
                    .salesforce.com
  RequestID: 07cA00000004CDUIA2 | [User: 005A0000000Df51IAC URL: /apex/AccountViewer]
    AccountViewerController.removeColdAccounts line: 36
    AccountViewerController.getAccountTable line: 24
```

2. To stop your debugging session, click the terminate icon (🟥).
3. To decrement the value of *`i`* after an account is removed from *`listToReduce`*, modify `AccountViewerController.cls`.



```
<terminated>devuser_            [Remote Apex Debugger]
  <terminated>                  .salesforce.com
```

```
      System.debug('Removing "cold" accounts');
      System.debug('   size before: ' + listToReduce.size());

      for (Integer i = 0; i < listToReduce.size(); i++) {
         Account a = listToReduce.get(i);
         if (a.Rating.equalsIgnoreCase('Cold')) {
            listToReduce.remove(i);
            System.debug('removed cold account: ' + a.Name);
            i--;
         }
      }

      System.debug('   size after: ' + listToReduce.size());
   }

   public void noOp() {
```

```
Apex session: 07aA00000004CALIA2
2015-07-13 09:24:51|Request: 07cA00000004CD0IAM|REQUEST_STARTED|[User: 005A0000000Df51IAC URL: /apex/AccountViewer]
2015-07-13 09:24:52|Request: 07cA00000004CD0IAM|REQUEST_FINISHED
2015-07-13 09:24:56|Request: 07cA00000004CD1IAM|REQUEST_STARTED|[User: 005A0000000Df51IAC URL: /apex/AccountViewer]
2015-07-13 09:24:56|Request: 07cA00000004CD1IAM|USER_DEBUG|AccountViewerController[line:32,column:1]|Removing "cold" account
2015-07-13 09:24:56|Request: 07cA00000004CD1IAM|USER_DEBUG|AccountViewerController[line:33,column:1]|   size before: 10
```

4. Save your changes.
5. Reload your Visualforce page: `https://`*`your_salesforce_instance`*`/apex/AccountViewer`
6. Select **Hide Cold Accounts**.
7. Celebrate! You've successfully debugged your code.

Delete Your Sample Accounts

Unless you want to keep your sample accounts so that you can play with them in the future, run this code to delete them.

1. Enter this code in the Force.com IDE's Execute Anonymous pane, in the Source to execute field.

```
List<Account> buggyAccounts = [SELECT Id
                          FROM Account
                          WHERE BillingCity = 'Suffragette City'];
delete buggyAccounts;
```

2. Click **Execute Anonymous**.

3. In the Accounts tab of the Salesforce user interface, verify that your accounts have been deleted.

Apex Debugger Limits and Considerations

Keep these limits and considerations in mind when working with the Apex Debugger.

IN THIS SECTION:

Apex Debugger Limits

Your use of the Apex Debugger is subject to these restrictions.

- You can use the Apex Debugger only with sandbox orgs.
- You can have only one active debugging session per org across all the org's sandboxes. However, you can purchase more sessions for your parent org. Each sandbox org can have only one active session.
- You can't debug more than two threads at a time.
- Your Apex Debugger session times out if it's left inactive. Under normal conditions, you're allowed 1 hour of inactivity. When your instance's server is experiencing peak loads, the timeout limit can be reduced to 30 minutes.
- Your Apex Debugger session times out after 4 hours, regardless of activity.
- Salesforce can terminate debugging sessions for maintenance purposes, with or without notice. You can't initiate a new debugging session until the maintenance is complete.

Apex Debugger Considerations

Don't be surprised if you encounter these potential "gotchas" when using the Apex Debugger.

- If you edit Apex classes while a debugging session is in progress, your breakpoints might not match your debugging output after you save your changes.
- Your debugging session is orphaned when you close Eclipse before stopping your session. If you have an orphaned session, you can't start a new session.
- `Eval` functionality isn't available.
- Hot swapping isn't permitted. These actions kill your debugging session.
  - Installing or uninstalling a package
  - Saving changes that cause your org's metadata to recompile

    You can't save changes to these items during a debugging session.

    - Apex classes or triggers
    - Visualforce pages or components
    - Lightning resources
    - Permissions or preferences
    - Custom fields or custom objects

- These entry points aren't supported.
  - Asynchronously executed code, including asynchronous tests

    Note: Test code between a pair of `startTest` and `stopTest` methods can run synchronously. To debug your asynchronous functionality, use these methods within your tests.

  - Batch, Queueable, and Scheduled Apex
  - Inbound email
  - Code with the `@future` annotation
- Keep these things in mind when working with breakpoints.
  - You can't set conditional breakpoints.
  - Breakpoints set on a `get` or `set` method must be within the method's body.
  - You can't set breakpoints in or step through Execute Anonymous blocks. However, when you hit a breakpoint using Execute Anonymous, we show your Execute Anonymous frame in the stack. To view your Execute Anonymous code's variables, click this line in the stack.
- Keep these things in mind when working with variables.
  - You can't watch variables.
  - Variable inspection in dynamic Visualforce and Lightning components isn't supported.
  - You can't drill into the instance variables of Apex library objects. To view these objects' contents, use their `toString` methods.
  - Variables declared within a loop are visible outside of the loop.
  - Drill into variables to see their children's values. For example, if you run the query `[SELECT Id, ContactId, Contact.accountId, Contact.Account.ownerId FROM Case]`, your results are nested as follows.

    ```
    Case
    --> Contact
    -----> contactId
    -----> Account
    --------> accountId
    --------> ownerId
    ```

  - When you perform a SOQL query for variables from the EntityDefinition table, your results include the *`durableId`* even if you don't explicitly `SELECT` that variable.

Apex Debugger Troubleshooting

If you encounter problems when using the Apex Debugger, try these troubleshooting techniques.

IN THIS SECTION:

Relaunch Your Debug Configuration

If your code isn't stopping on breakpoints, variables aren't displaying in the Variables pane, or you don't see turning yellow gears in the Debug pane, relaunch your debug configuration.

Kill an Orphaned Session

If Eclipse crashes or is shut down before you end your debugging session, your session is orphaned. You can't start a new session until you remove the orphan. But don't panic! You can kill an orphaned session in Setup or with the Tooling API.

Change Your Session Timeout Preference

If you're on a slow network, you can change the default timeout setting for your Debugger connection. This setting determines how long Eclipse waits for the Debugger to respond when you take an action, such as stepping through your code.

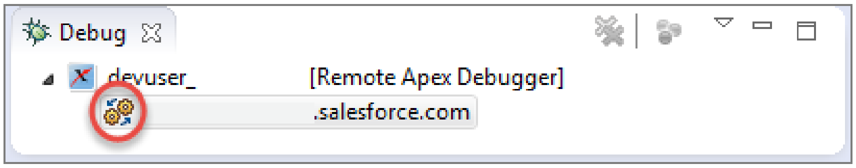
Report Drastic Issues to Customer Support

It's unlikely that you'll encounter drastic issues while working with the Apex Debugger. However, never say never. If you encounter performance issues or are unable to kill your orphaned sessions, contact Salesforce Customer Support for help. If necessary, Support can kill all debugging sessions for the instance on which your org is running. This process prevents you—and all orgs running on your instance—from immediately initiating new debugging sessions.

Relaunch Your Debug Configuration

If your code isn't stopping on breakpoints, variables aren't displaying in the Variables pane, or you don't see turning yellow gears in the Debug pane, relaunch your debug configuration.

1. Click the arrow next to the debug icon (🐞 ▾), and choose your configuration from the list. This action relaunches your debug configuration.
2. Wait for the turning yellow gears to reappear in the Debug pane.



Debug

devuser_ [Remote Apex Debugger]

.salesforce.com

Kill an Orphaned Session

If Eclipse crashes or is shut down before you end your debugging session, your session is orphaned. You can't start a new session until you remove the orphan. But don't panic! You can kill an orphaned session in Setup or with the Tooling API.

IN THIS SECTION:

Kill a Debugging Session in Setup

From the Apex Debugger page in Setup, you can easily kill any debugging session in your org—even sessions belonging to your coworkers. Please use this power compassionately!

Kill a Debugging Session with the Tooling API

If you're too cool for declarative tools, fret not: You can kill a debugging session using the Tooling API, without going anywhere near the Setup tree.

Kill a Debugging Session in Setup

From the Apex Debugger page in Setup, you can easily kill any debugging session in your org—even sessions belonging to your coworkers. Please use this power compassionately!

1. From Setup, enter `Apex Debugger` in the `Quick Find` box, then click **Apex Debugger**.
2. Click **Kill Debugger**.

Kill a Debugging Session with the Tooling API

If you're too cool for declarative tools, fret not: You can kill a debugging session using the Tooling API, without going anywhere near the Setup tree.

1. Open the Developer Console.
2. Open the Query Editor.
3. Select **Use Tooling API**.
4. Run this SOQL query.

```
SELECT Status FROM ApexDebuggerSession WHERE Status = 'Active'
```

5. Change the value of *`Status`* from `Active` to `Kill`.
6. Save the modified row.

Change Your Session Timeout Preference

If you're on a slow network, you can change the default timeout setting for your Debugger connection. This setting determines how long Eclipse waits for the Debugger to respond when you take an action, such as stepping through your code.

1. In Eclipse, select **Window** > **Preferences** > **Force.com** > **Apex Debugger**.
2. Enter a new value for Connection timeout (ms). For slower connections, we recommend a connection timeout of up to 30,000 milliseconds.
3. Click **Apply**.
4. Click **OK**.

Report Drastic Issues to Customer Support

It's unlikely that you'll encounter drastic issues while working with the Apex Debugger. However, never say never. If you encounter performance issues or are unable to kill your orphaned sessions, contact Salesforce Customer Support for help. If necessary, Support can kill all debugging sessions for the instance on which your org is running. This process prevents you—and all orgs running on your instance—from immediately initiating new debugging sessions.

CHAPTER 7 Useful References

In this chapter ...

This section provides additional details on a range of topics:

- Force.com Project Properties
- Apex Code Settings: Log Category and Log Level
- About Package.xml
  - About Metadata Files
  - Metadata Types
- Troubleshooting the Force.com IDE: Debug Mode

Force.com IDE Release Notes

Because the Force.com IDE has an off-cycle release cadence, changes are not typically documented in the Salesforce Release Notes. Read Force.com IDE Release Notes for API versions 31.0 and later here.

Winter ’16 (Force.com IDE v35.0)

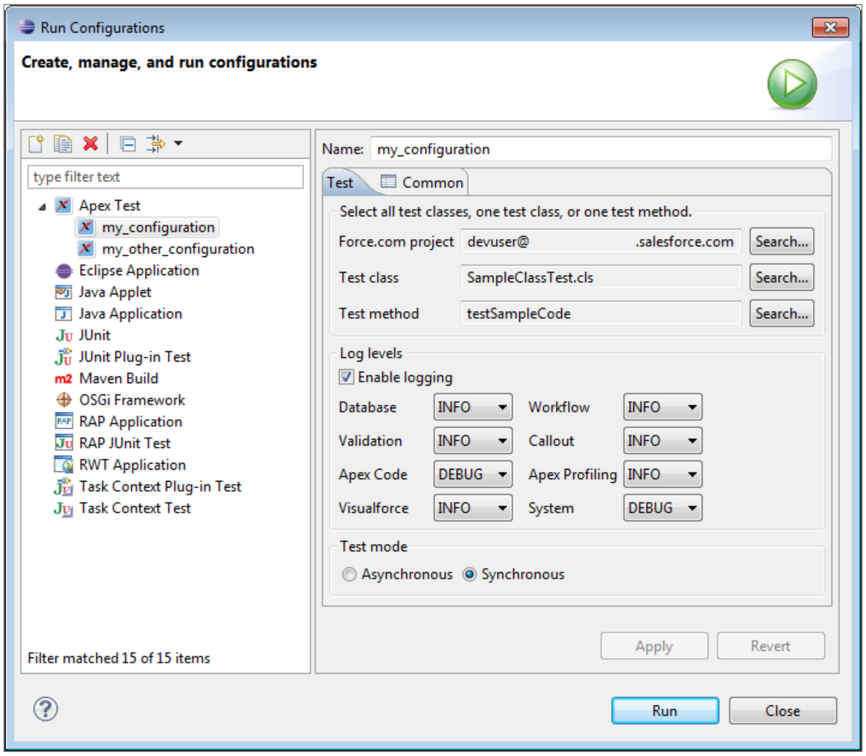
Winter ’16 (Force.com IDE v35.0) contains the following updates.

Enabled auto-completion for `System` types

In Spring ’15 (Force.com IDE v33.0), we removed inline auto-complete tips for Apex built-in objects to allow packaging of a new compiler that features outline view. We’ve been working on bringing auto-completion back. In Winter ’16 (Force.com IDE v35.0), we’re reintroducing auto-completion for the `System` namespace.

Provided run configurations for Apex tests, with configurable logging levels

To execute Apex unit tests, select **Run** > **Run Configurations** > **Apex Test**. To create a test run configuration, then select **New launch configuration** ( ). To execute the selected test run configuration, click **Run**.



Name: my_configuration

Select all test classes, one test class, or one test method.

Force.com project: devuser@ .salesforce.com

Test class: SampleClassTest.cls

Test method: testSampleCode

Log levels

☑ Enable logging

Database	INFO	Workflow	INFO
Validation	INFO	Callout	INFO
Apex Code	DEBUG	Apex Profiling	INFO
Visualforce	INFO	System	DEBUG

Test mode

○ Asynchronous ● Synchronous

Made the interactive Apex Debugger generally available

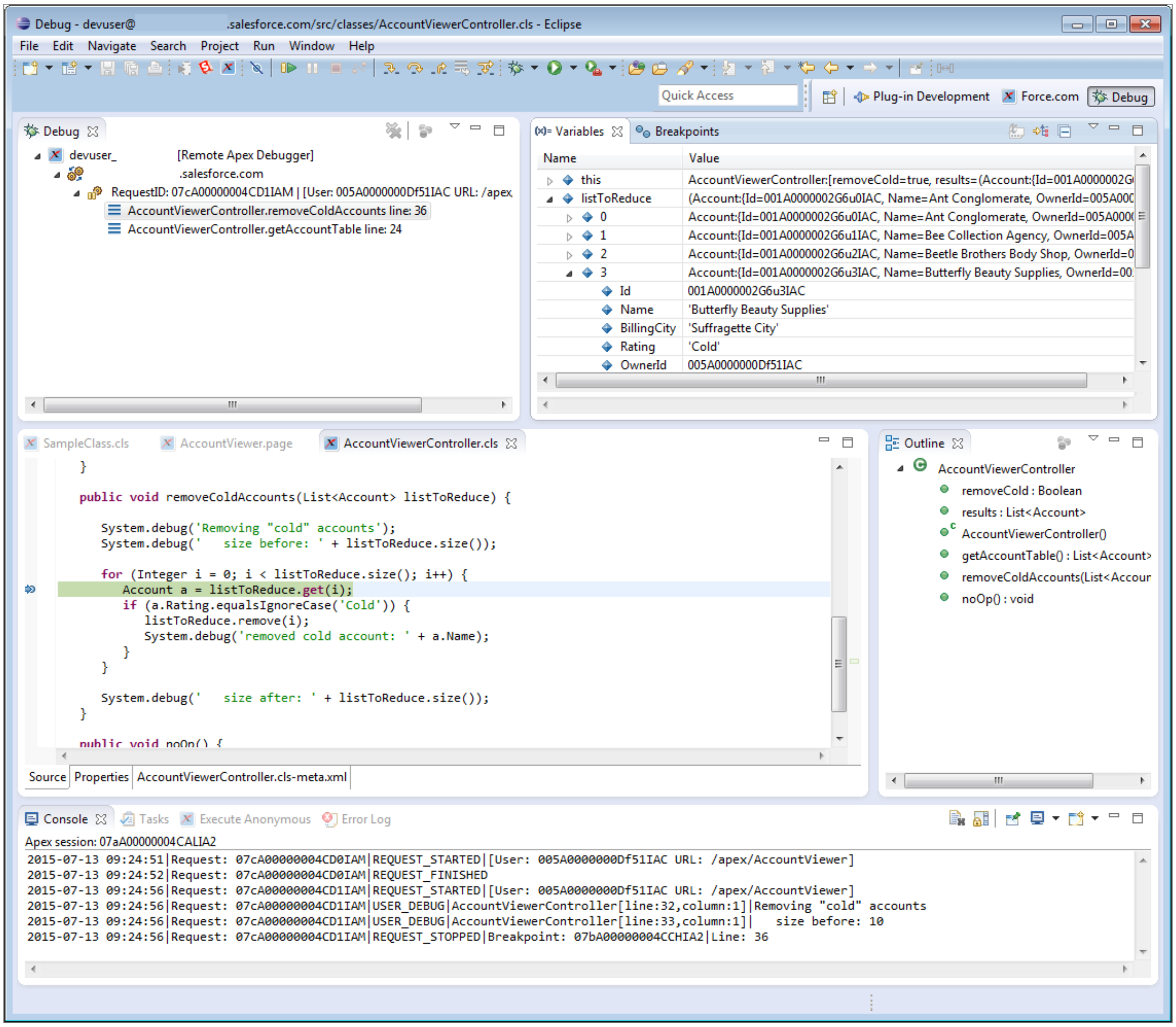
The Apex Debugger extends the Force.com IDE plug-in for Eclipse and behaves similarly to debuggers available for other languages. Use it in sandbox orgs to root out the bugs in your Apex code. After you’ve set a breakpoint and started a debugging session, you can debug actions in your org that cause the line of code to execute.

Note: Some services and subscriptions include the Apex Debugger for an extra cost. For pricing details, contact your Salesforce account executive.

Use the Apex Debugger to complete the following actions.

- Set breakpoints in Apex classes and triggers.

- View variables, including sObject types, collections, and Apex `System` types.
- View the call stack, including triggers activated by Apex Data Manipulation Language (DML), method-to-method calls, and variables.
- Interact with global classes, exceptions, and triggers from your installed managed packages. When you inspect objects that have managed types that aren't visible to you, only global variables are displayed in the variable inspection pane.
- Complete standard debugging actions, including step into, over, and out, and run to breakpoint.
- Output your results to the Console window.


Debug - devuser@ .salesforce.com/src/classes/AccountViewerController.cls - Eclipse
File Edit Navigate Search Project Run Window Help
Quick Access
Plug-in Development
Force.com
Debug
Debug
Variables
Breakpoints
SampleClass.cls
AccountViewer.page
AccountViewerController.cls
Outline
Source
Properties
AccountViewerController.cls-meta.xml
Console
Tasks
Execute Anonymous
Error Log

devuser_ [Remote Apex Debugger]
.salesforce.com
RequestID: 07cA00000004CD1IAM | [User: 005A0000000Df51IAC URL: /apex
AccountViewerController.removeColdAccounts line: 36
AccountViewerController.getAccountTable line: 24

Name	Value
this	AccountViewerController:[removeCold=true, results=(Account:{Id=001A0000002G
listToReduce	(Account:{Id=001A0000002G6u0IAC, Name=Ant Conglomerate, OwnerId=005A000
0	Account:{Id=001A0000002G6u0IAC, Name=Ant Conglomerate, OwnerId=005A000
1	Account:{Id=001A0000002G6u1IAC, Name=Bee Collection Agency, OwnerId=005A
2	Account:{Id=001A0000002G6u2IAC, Name=Beetle Brothers Body Shop, OwnerId=0
3	Account:{Id=001A0000002G6u3IAC, Name=Butterfly Beauty Supplies, OwnerId=00
Id	001A0000002G6u3IAC
Name	'Butterfly Beauty Supplies'
BillingCity	'Suffragette City'
Rating	'Cold'
OwnerId	005A0000000Df51IAC

```
    }

    public void removeColdAccounts(List<Account> listToReduce) {

        System.debug('Removing "cold" accounts');
        System.debug('    size before: ' + listToReduce.size());

        for (Integer i = 0; i < listToReduce.size(); i++) {
            Account a = listToReduce.get(i);
            if (a.Rating.equalsIgnoreCase('Cold')) {
                listToReduce.remove(i);
                System.debug('removed cold account: ' + a.Name);
            }
        }

        System.debug('    size after: ' + listToReduce.size());
    }

    public void noOp() {
```

AccountViewerController
removeCold : Boolean
results : List<Account>
AccountViewerController()
getAccountTable() : List<Account>
removeColdAccounts(List<Accour
noOp() : void

Apex session: 07aA00000004CALIA2

```
2015-07-13 09:24:51|Request: 07cA00000004CD0IAM|REQUEST_STARTED|[User: 005A0000000Df51IAC URL: /apex/AccountViewer]
2015-07-13 09:24:52|Request: 07cA00000004CD0IAM|REQUEST_FINISHED
2015-07-13 09:24:56|Request: 07cA00000004CD1IAM|REQUEST_STARTED|[User: 005A0000000Df51IAC URL: /apex/AccountViewer]
2015-07-13 09:24:56|Request: 07cA00000004CD1IAM|USER_DEBUG|AccountViewerController[line:32,column:1]|Removing "cold" accounts
2015-07-13 09:24:56|Request: 07cA00000004CD1IAM|USER_DEBUG|AccountViewerController[line:33,column:1]|    size before: 10
2015-07-13 09:24:56|Request: 07cA00000004CD1IAM|REQUEST_STOPPED|Breakpoint: 07bA00000004CCHIA2|Line: 36
```

Spring '15 (Force.com IDE v33.0)

Spring '15 (Force.com IDE v33.0) contains the following updates.

Introduced a wizard for generating Apex classes from a WSDL

Previously, you could only generate classes from a WSDL in the Salesforce user interface. The WSDL-based classes enable you to make callouts to external services.

Open-source code for the WSDL to Apex wizard is available in the WSDL2Apex GitHub repository. Developers in the GitHub community can add enhancements or customizations to WSDL2Apex. The Force.com IDE plug-in is a snapshot of the WSDL2Apex and idecore GitHub repositories at the time of the latest official plug-in release.

The Spring '15 (Force.com IDE v33.0) version of the WSDL to Apex tool includes this user-visible contribution from the open-source community.

Update ApexTypeMapper.java—#6

Added `integer` and `boolean` as reserved keywords. These words now pick up the `_x` suffix when used as variable names.

Incorporated pull requests from the open-source community

For complete lists of pull requests, see https://github.com/forcedotcom/idecore/pulls and https://github.com/forcedotcom/WSDL2Apex/pulls. User-visible impacts of community-submitted changes that were incorporated in Spring '15 (Force.com IDE v33.0) include:

Respect spacesForTabs preference for auto-indent—#39

Previously, a hard-coded tab character in `ApexAutoIndentStrategy.java` led to Apex files that contained a mixture of spaces and tabs when the `spacesForTabs` preference was enabled. This change makes the Force.com IDE respect Eclipse's **General** > **Editors** > **Text Editors** > **Insert spaces for tabs** preference.

Speed up unit test postprocessing—#52

This change made a significant improvement in debug-log processing speeds when running unit tests.

Changes to monitor only SF files in `/src/` folder—#57

Popup menus now work only in the `src` folder and on projects.

The Force.com submenu is now easier to see, thanks to the addition of an icon.

Changes for Apex Test Runner: Code coverage results should be alphabetized—#61

Code coverage results are now alphabetized in the Apex Test Runner pane.

Fix display of child and parent records in schema browser—#64

A red "icon missing" image has been removed in the Schema Explorer. This icon was displayed erroneously.

Need to have a different text for debug only—#65

The Execute Anonymous view now displays `System.debug` statements in a more noticeable text style.

Spring '15 (Force.com IDE v33.0)

Spring '15 (Force.com IDE v33.0) contains the following updates.

New wizard for generating Apex classes from a WSDL

Previously, you could only generate classes from a WSDL in the Salesforce user interface. The WSDL-based classes enable you to make callouts to external services.

Open-source code for the WSDL to Apex wizard is available in the WSDL2Apex GitHub repository. Developers in the GitHub community can add enhancements or customizations to WSDL2Apex. The Force.com IDE plug-in is a snapshot of the WSDL2Apex and idecore GitHub repositories at the time of the latest official plug-in release.

The Spring '15 (Force.com IDE v33.0) version of the WSDL to Apex tool includes this user-visible contribution from the open-source community.

Update ApexTypeMapper.java—#6

Added `integer` and `boolean` as reserved keywords. These words now pick up the `_x` suffix when used as variable names.

Incorporated pull requests from the open-source community

For complete lists of pull requests, see https://github.com/forcedotcom/idecore/pulls and https://github.com/forcedotcom/WSDL2Apex/pulls. User-visible impacts of community-submitted changes that were incorporated in Spring '15 (Force.com IDE v33.0) include:

Respect spacesForTabs preference for auto-indent—#39

Previously, a hard-coded tab character in `ApexAutoIndentStrategy.java` led to Apex files that contained a mixture of spaces and tabs when the `spacesForTabs` preference was enabled. This change makes the Force.com IDE respect Eclipse's **General** > **Editors** > **Text Editors** > **Insert spaces for tabs** preference.

Speed up unit test postprocessing—#52

This change made a significant improvement in debug-log processing speeds when running unit tests.

Changes to monitor only SF files in `/src/` folder—#57

Popup menus now work only in the `src` folder and on projects.

The Force.com submenu is now easier to see, thanks to the addition of an icon.

Changes for Apex Test Runner: Code coverage results should be alphabetized—#61

Code coverage results are now alphabetized in the Apex Test Runner pane.

Fix display of child and parent records in schema browser—#64

A red "icon missing" image has been removed in the Schema Explorer. This icon was displayed erroneously.

Need to have a different text for debug only—#65

The Execute Anonymous view now displays `System.debug` statements in a more noticeable text style.

Summer '14 (Force.com IDE v31.0)

Summer '14 (Force.com IDE v31.0) contains the following updates.

- Made the Force.com IDE source code available on GitHub.
- Added new metadata support. See the Salesforce.com Summer '14 Release Notes for a complete list of metadata enhancements.
- Facilitated editing of metadata using the new Properties tab—a form-based GUI.
- Discontinued support for Java 6. We now require Java 7.
- Set offline mode as the default for new projects to shorten save times.
- Enabled syntax checking and outline view using a new parser, which can be toggled through **Preferences** > **Force.com** > **Apex Parser**.
- Set the Tooling API as the default mechanism used for Apex classes and triggers and for Visualforce pages and components.
- Removed inline auto-complete tips for Apex built-in objects to allow packaging of a new compiler that features outline view. Auto-complete tips are still available in Spring '14 (Force.com IDE v30.0) and earlier versions.

Force.com Project Properties

The properties on the main Force.com Project Properties page determine a Force.com project's Salesforce connection settings. Use this page to update your password or security token, change timeout values, or associate your project with a different home organization

Field	Description
Project Name	Required. A project is associated with one organization. Project names must be unique. You can create more than one project that connects to the same organization, but they cannot have the same project name. Note: You can enter any name for the project here, but we recommend that you append the login name used for

Field	Description
	the organization associated with the project. For example, if you logged into an organization as `mary_jones@mycompany.com`, you would use `myProject_mary_jones@mycompany.com` for the project name. Since you are likely to create multiple projects on multiple organizations, it is important to clearly identify the project with the organization it represents.
Username	Required. The username you use to log in to the organization associated with this project ("home organization"). The username associated with this connection must have the "Modify All Data" permission. Typically, this is only enabled for System Administrator users.
Password	Required. The password for the specified username. Note: When you create a new project, you might be prompted about a new master password. This is a separate password of your choosing required by Eclipse secure storage, and is not associated with your Salesforce credentials. For details on Eclipse secure storage, see the *Eclipse Workbench User Guide*.
Security Token	The security token is appended to your password as an added security measure. If you are using a security token, enter the value here. For more information, see the Salesforce online help topic Setting Login Restrictions.
Environment	Required. Choose the appropriate environment for your connection: • Production/Developer Edition - Choose this option if you are connecting to a production or Developer Edition organization. • Sandbox - Choose this option if you are connecting to a sandbox organization. Sandbox organizations have an URL that starts with `test`. • Pre-Release - Choose this option if you are connecting to a prerelease server. • Other (Specify) - Choose this option if you want to connect to a specific instance.
Hostname	This field is only available if you have chosen Other (Specify), in the Environment field above. Enter a specific server instance.
Do not change endpoint	Choose this option if you want to keep the endpoint unchanged for this project. This option is for specialized cases.
Timeout (sec)	If you experience timeouts before all the items in your project can download, you can reset the timeout from a minimum of 3 seconds to a maximum of 600 seconds. This is not usually necessary unless

Field	Description
	your project requires an long-running server communication. For example, if you had a long-running test, more than 50 components, or very large or complex components, you might consider adjusting this value. If you reach the maximum limit, you will be prompted to continue or abort.
Proxy Settings	If you connect through a proxy, use Eclipse's proxy configuration. Click the Proxy Settings link to open the **Window** > **Preferences** > **General** > **Network Connections** dialog.

Apex Code Settings: Log Category and Log Level

The default logging levels for Apex tests executed during deployments and for the Execute Anonymous view are defined on the Apex Code Settings page in the project properties dialog. To open the project properties dialog, right-click your top-level project folder and select **Properties**.

To define logging levels for Apex test execution, select **Run** > **Run Configurations** > **Apex Test**. To create a test run configuration, then select **New launch configuration** (🗋).

You can customize these log categories.

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, the actions taken, and so on.
`Validation`	Includes information about validation rules, such as the name of the rule, whether the rule evaluated true or false, and so on.
`Callout`	Includes the request-response XML that the server is sending and receiving from an external web service. Useful when debugging issues related to using Force.com web service API calls or troubleshooting user access to external objects via an OData adapter for Lightning Connect.
`Apex Code`	Includes information about Apex code and can include information such as log messages generated by DML statements, inline SOQL or SOSL queries, the start and completion of any triggers, and the start and completion of any test method, and so on.
`Apex Profiling`	Includes cumulative profiling information, such as the limits for your namespace, the number of emails sent, and so on.
`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.

You can assign these log levels to your log categories.

Log Level	Description
Error, **Warn**, **Info**	Includes error, warning, and information messages.
Debug	Includes lower-level messages, and messages generated by calls to the `System.debug` method.
Fine, **Finer**	Includes log messages generated by calls to the `System.debug` method, every DML statement or inline SOQL or SOSL query, and the entrance and exit of every user-defined method. In addition, the end of the debug log contains overall profiling information for the portions of the request that used the most resources. These resources include SOQL and SOSL statements, DML operations, and Apex method invocations.
Finest	Includes all messages generated by the **Fine** or **Finer** log levels, as well additional information on Apex scripts, including the following. • Variable declaration statements • Start-of-loop executions • All loop controls, such as `break` and `continue` • Thrown exceptions • Static and class initialization code • Any changes in the `with sharing` context

About Package.xml

The `package.xml` file, also known as the *project manifest*, is a control file that determines the set of metadata components to retrieve or deploy in a project. If you are looking at the Package Explorer, this file is located in the `src` folder.

Note: In practice, it is much easier to use the Project Properties dialog to add or remove components from a project—which modifies `package.xml` for you—but it is also possible to edit `package.xml` by hand.

If your Force.com project is associated with a particular server-side package (as specified in the `<fullName>` element), the server determines what components are listed in `package.xml` and any user changes to the file will be overwritten.

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

Name	Description
`<fullName>`	Optional. The name of a server-side package associated with the project. If the `<fullName>` field is present, all components in the package will be downloaded into the project, and new components created from the IDE will be added to that package.
`<types>`	This element contains one or more `<members>` tags and one `<name>` tag, and is used to list the metadata components of a certain type to retrieve or deploy.
`<members>`	The full name of a component. There is one `<members>` element defined for each component in the directory. You can replace the value in this member with the wildcard character `*` (asterisk) instead of listing each member separately. This is a child element of `<types>`.

Name	Description
`<name>`	Contains the type of the component, for example `CustomObject` or `Profile`. There is one name defined for each component type in the directory. This is a child element of `<types>`.
`<version>`	The Metadata API version number of the files being retrieved or deployed. When deploying, all the files must conform to the same version of the Metadata API.

SEE ALSO:

- About Metadata Files
- Metadata Types
- Add/Remove Metadata Components
- Project Properties

About Metadata Files

The Force.com platform executes applications in its multi-tenant environment through a concept known as *meta-customization*. Application components such as schema definitions, page layouts, workflow rules, even classes and triggers, are stored in a database, just like data in a traditional single-tenant application. These database records that describe the application itself are called *metadata components*. Because they are stored in the database and loaded at runtime, metadata components can be created and modified on the fly by manipulating these records using the Setup pages of your Salesforce organization.

The Force.com IDE brings these metadata components into the context of traditional source code based application development by presenting them as text files. In the IDE, you work with metadata files that define Apex classes and triggers, custom objects, custom fields, Visualforce pages, and other metadata components, and then synchronize these artifacts back to your home organization (a sandbox or Developer Edition environment).

Because the files are text-based, these metadata files can be created or modified in a text editor, compared using text diff tools, managed in a version control system such as CVS or Subversion, and even deployed from one Salesforce organization to another.

Note: The Force.com IDE uses the Metadata API to communicate with the Force.com servers. This API contains a `retrieve()` method to convert metadata components stored in the database into text files, and a `deploy()` method to convert metadata files back into database records. Although most developers will want to use the Eclipse user interface to accomplish their development tasks, the underlying Metadata API calls and components are openly available and documented for your reference. For more information about Metadata calls, see the *Metadata API Developer Guide*.

SEE ALSO:

- Metadata Types

Metadata Types

There are three different kinds of metadata: simple, compound, and complex.

- Simple — These types consist of only a single file, which does not depend on another file for its existence. For example, a custom application is a simple type of metadata. Simple metadata types may be retrieved or deployed by themselves.
- Compound — These types consist of two files, a class file and a metadata file (the name is appended with `-meta.xml.`). For example, Apex classes and triggers, are compound types because there is a class file and a supporting metadata file.

- Complex — These metadata files may contain multiple named components. For example, a `.object` file may contain an entire custom object, or a subset of custom fields on that object, depending on what components were included in the project. The `.workflow` file behaves in a similar manner with respect to individual workflow types.

Custom fields can be considered both simple and complex. Simple because you can retrieve them alone, complex because they are always defined within the context of a standard or custom object.

For a list of the metadata types that can be retrieved or deployed with the Metadata API, and whether or not the component may be retrieved with the wildcard (*) symbol in `package.xml`, see Metadata Types in the Metadata API Developer's Guide.

Troubleshooting the Force.com IDE: Debug Mode

If you experience errors in the Force.com IDE, it can be useful to run Eclipse in Debug Mode. This will cause the IDE to write additional information to its system log.

To turn on Debug Mode, add the following parameter to Eclipse's startup command line or in `eclipse.ini`:

- `-Dforce-ide-debug=true`

You can view the system log within the IDE. For more information, see Force.com Log Viewer.

You can also write the zip file to disk for each save, refresh, synchronize, or deploy action, which can be helpful in diagnosing errors. Add the following parameter to Eclipse's startup command line or in `eclipse.ini`:

- `-Dforce-ide-temp=<full-path-to-directory>`

Note: If you need help that is not provided in the Force.com IDE documentation, you can contact the developer.salesforce.com discussion forums. Be sure to include as much information as possible, including any relevant error log entries.

Additional Resources

The Force.com technical library is available online at `developer.salesforce.com/docs`. Among the many documents you'll find useful are:

Developer Guides for the Force.com Programming Languages and Controllers:

- *Force.com Apex Code Developer Guide*
- *Visualforce Developer Guide*

API Developer Guides:

- *SOAP API Developer Guide*
- *Force.com REST API Developer Guide*
- *Metadata API Developer Guide*
- *Force.com Streaming API Developer Guide*
- *Chatter REST API Developer Guide*

Reference Materials:

- *Object Reference for Salesforce and Force.com*
- *Force.com SOQL and SOSL Reference*
- *Development Lifecycle Guide*
- *AppExchange Publishing Guide*
- *Force.com Migration Tool Guide*

- *Force.com Sites Implementation Guide*
- *Security Implementation Guide*

INDEX