Force.com IDE Developer Guide

Force.com IDE v38.0, Spring '18

@salesforcedocs

Last updated: April 26, 2018

CONTENTS

CHAPTER 1 Get Started with the Force.com IDE

In this chapter ...

The Force.com IDE is an integrated development environment for developing applications on the Lightning Platform using Apex, Visualforce, and metadata components. Designed for developers and development teams, the IDE provides tools to accelerate Salesforce 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.

- Develop with the Force.com IDE
- Test Code with the Force.com IDE
- Deploy Code with the Force.com IDE

Install the Force.com IDE Plug-In

You can install the Force.com IDE into your existing Eclipse distribution or upgrade from a previous version.

Before installing the Force.com IDE, ensure that you have these items installed on your workstation.

- A supported operating system
  - Windows 7, 8, or 10
  - macOS 10.7, 10.8, 10.9, 10.10, or 10.11
  - Ubuntu 12.04 LTS or 14.04 LTS
- Java SE Development Kit (JDK), Runtime Environment 8, or later (Java download page)

  Note: Be sure to download the full JDK—without it, the plug-in doesn't load. On macOS, the full JDK is not the default JRE installation.
- Eclipse Mars or Neon (Eclipse download site)

  The Eclipse IDE for Java Developers distribution is recommended.

1. Launch Eclipse and select **Help** > **Install New Software**.
2. Click **Add**.
3. In the Add Repository dialog, set the name to `Force.com IDE` and the location to `https://developer.salesforce.com/media/force-ide/eclipse45`. For Spring '16 (Force.com IDE v36.0) and earlier Force.com IDE versions, use `https://developer.salesforce.com/media/force-ide/eclipse42`.
4. Click **OK**.
5. To install an older version of the plug-in (for example, if you don't have Java 8), deselect **Show only the latest versions of available software**.
   Eclipse downloads the list of available plug-ins and displays them in the Available Software dialog.
6. Select **Force.com IDE**.
7. If you want to install the Apex Debugger or our tools for working with Lightning components, select **Force.com Debugger** or **Force.com Lightning Support**.
8. Click **Next**.
9. In the Install Details dialog, click **Next**.
10. In the Review Licenses dialog, accept the terms and click **Finish**.
11. If you chose to install support for Lightning components, Eclipse displays a warning dialog about installing software that contains unsigned content. We are bundling third-party plug-ins to support Lightning components. Salesforce doesn't own these third-party plug-ins; hence, we don't sign them. Click **OK** to proceed.
12. Eclipse downloads and installs the Force.com IDE and the required dependencies. When the installation is complete, you are prompted to restart. Click **Yes**.
13. When Eclipse restarts, select **Window** > **Open Perspective** > **Other**. Select **Force.com** and then click **OK**.

    You are now ready to develop and customize Salesforce applications in Eclipse!

Note: When you create a project, you are sometimes prompted for a new master password. This master password is a separate password of your choosing. It's required by Eclipse secure storage but not associated with your Force.com IDE credentials. For information on Eclipse secure storage, see the *Eclipse Workbench User Guide*.

Quickstart: Explore the Force.com IDE

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

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

Note: To complete this exercise, you need administrative access to two different orgs: either two Developer Edition orgs, or a Developer Edition org and a sandbox org. Do not use a production org for this exercise.

1. Create a Project

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

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

   Note: If you don't see Force.com Project, you're 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, we suggest that you 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 org associated with this project (home org). 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. To create the project, click **Finish**, connect the project to the associated org, and copy components from the home org 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 class, 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 record is created.

1. Right-click 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 org in a browser and create an account. The Description field is pre-populated with the value "Hello World".

You can edit the class or trigger in the project. When you save, the changes are also saved to the associated org, assuming that no conflicts exist. If you edit the class or trigger in the org itself, 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 org, at least 75% of your code must be covered by automated unit tests before you can deploy it to a production org.

These steps create a simple unit test for the Hello World program.

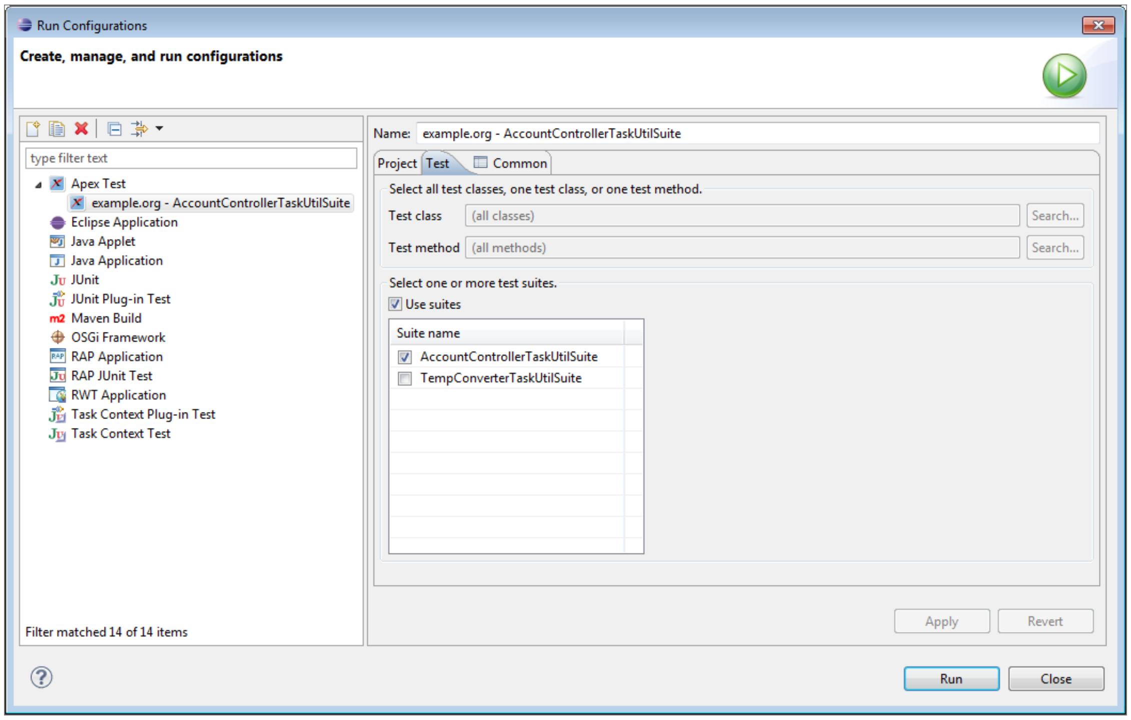
1. Open `MyHelloWorld.cls` and add these test methods.

```
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. To execute Apex unit tests:
   a. Select **Run** > **Run Configurations** > **Apex Test** > **New launch configuration** ( ).
   b. Select the **Test** tab.
   c. Add tests to your run configuration.
      - To add a test class or test method, click one of the Search buttons. To select a method, first select the class that contains that method.
      - To create a run of test suites:
        a. Select **Use suites**.
        b. Select one or more test suites.
   d. Click **Apply**.
   e. To execute the selected test run configuration, click **Run**.



Name: example.org - AccountControllerTaskUtilSuite

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

Test class (all classes)

Test method (all methods)

Select one or more test suites.

[✓] Use suites

Suite name
[✓] AccountControllerTaskUtilSuite
[ ] TempConverterTaskUtilSuite

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

Offline and Online Modes

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, the Force.com IDE creates projects in offline mode. When you work in offline mode, you avoid compiling your org'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, consider toggling 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

Update the Force.com IDE

It is recommended that you always use the latest version of the Force.com IDE so that 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, upgrade your projects to the latest version. For more information, see Upgrade Project.

CHAPTER 2 Develop with the Force.com IDE

In this chapter ...

The Force.com IDE allows you to create and edit Apex, Visualforce, and XML metadata components. The Force.com IDE's source code editors 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 org (the home org) 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 a 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

After 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

Apex Editor

Apex Code Assist

Execute Anonymous View

Offline and Online Modes

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.

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

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**.

Add or Remove Metadata Components

Component Properties

Upgrade Project

SEE ALSO:

Create a Force.com Project

Add or Remove Metadata Components

Project Properties

Offline and Online Modes

About Package.xml

Create a Force.com Project

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

Note: If you don't see Force.com Project, you're 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, we suggest that you 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 properties for the project. For details on these settings, see Force.com Project Properties. Click **Next**.

   Note: When you create a project, you are sometimes prompted for a new master password. This password 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, Lightning, and Visualforce (classes, triggers, Lightning component bundles, pages, Visualforce components, and static resources)	Select this option to retrieve only Apex, Lightning, and Visualforce components, including classes, triggers, components, pages, and static resources.
Selected metadata components	To open the Choose Metadata Components dialog, select this option and click **Choose**.
Contents of package	To retrieve the contents of a particular package, select this option and then choose a package from the **Choose package** drop-down menu. The Contents of package option is available only when packages are installed in the org you're connected to.
None	Select this option to retrieve no components. For example, select this option if you're working in a team-based environment where source files are checked out from a source-control system. When creating a project offline, None is the only option available.

3. To create the project, click **Finish**.

Note: The selections you make in the New Force.com Project wizard determine the contents of the `package.xml` file, which defines the components downloaded from the server into your project. To modify the project contents later, right-click your project and select **Properties** > **Force.com** > **Project Contents**. In special cases, you can also edit the `package.xml` file manually. 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 or 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 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. When you're prompted whether you want to delete the file, click **Yes** to remove the local copy.
8. In the Remote Delete Confirmation dialog, you're 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. To delete files both locally and on the server, make sure that you're working online. Right-click your project in the Package Explorer, and then select **Force.com** > **Work Online**.
2. In the Package Explorer, expand the nodes to find the component you want to delete.
3. Right-click the component and choose **Delete**.
4. When you're prompted whether you want to delete the file, click **OK** to remove the local copy.

5. Finally, in the Remote Delete Confirmation dialog, you're prompted whether to also delete the component on the server. Click **Yes** to delete the component entirely.

   Choose Metadata Components

   Field-Level Security Warning

SEE ALSO:

Choose Metadata Components

Field-Level Security Warning

Choose Metadata Components

Use the Choose Metadata Components dialog to specify which Lightning Platform 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.

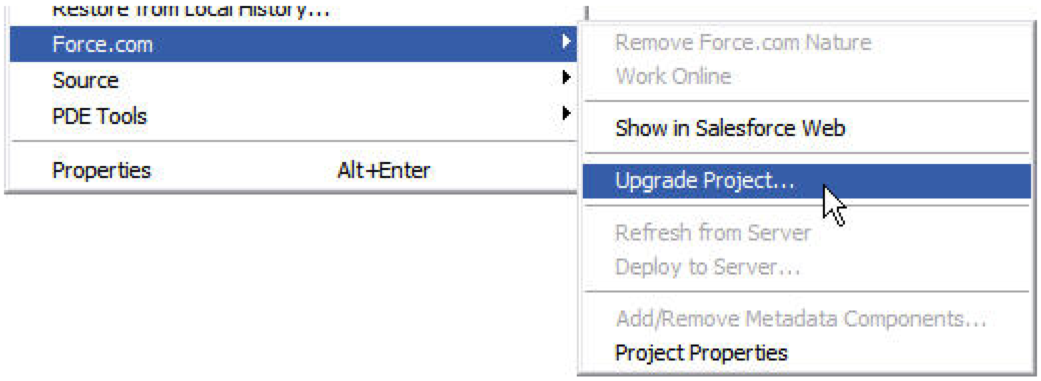
Upgrade Project

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.

Apex Code in the Force.com IDE

The Force.com IDE includes useful features for writing and executing Apex code.

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.

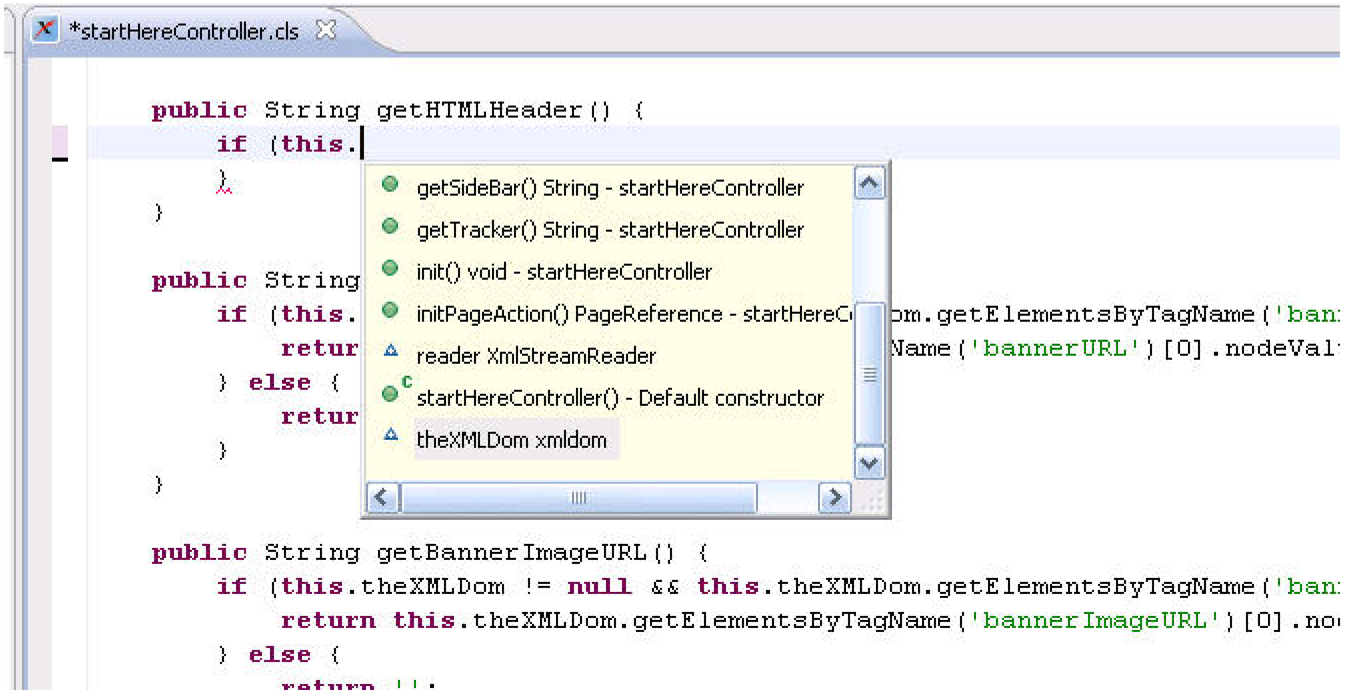
Note: The editor has content assistance for all built-in types, including both Apex and XML metadata.

SEE ALSO:

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.



```
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 '';
```

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

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.

Lightning Components in the Force.com IDE

Summer '16 (Force.com IDE v37.0) and later versions of the Force.com IDE include support for the Lightning Component Framework.

What is the Lightning Component Framework?

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

Notable Lightning Components Features in the Force.com IDE

The Force.com IDE includes lots of cool features to make working with Lightning components easier. Here are a few of our favorites.

Set Up the Force.com IDE with Lightning Components Support

Support for the Lightning Component Framework is generally available in Summer '16 (Force.com IDE v37.0) and later versions of the Force.com IDE. Be sure to select **Force.com Lightning Support** when you install or update the Force.com IDE plug-in.

Explore Lightning Components in the Force.com IDE

Learn how to create a simple Lightning component that renders a list of contacts from your org. Start by creating an Apex controller class, and then create a Lightning component and an event handler. Finally, render the list of contacts in the component.

What is the Lightning Component Framework?

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

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

What is Salesforce Lightning?

Lightning includes the Lightning Component Framework and some exciting tools for developers. Lightning makes it easier to build responsive applications for any device.

Why Use the Lightning Component Framework?

The benefits include an out-of-the-box set of components, event-driven architecture, and a framework optimized for performance.

Open Source Aura Framework

The Lightning Component framework is built on the open source Aura framework. The Aura framework enables you to build apps completely independent of your data in Salesforce.

Components

Components are the self-contained and reusable units of an app. They represent a reusable section of the UI, and can range in granularity from a single line of text to an entire app.

Events

Event-driven programming is used in many languages and frameworks, such as JavaScript and Java Swing. The idea is that you write handlers that respond to interface events as they occur.

What is Salesforce Lightning?

Lightning includes the Lightning Component Framework and some exciting tools for developers. Lightning makes it easier to build responsive applications for any device.

Lightning includes these technologies:

- Lightning components give you a client-server framework that accelerates development, as well as app performance, and is ideal for use with the Salesforce mobile app and Salesforce Lightning Experience.
- The Lightning App Builder empowers you to build apps visually, without code, quicker than ever before using off-the-shelf and custom-built Lightning components. You can make your Lightning components available in the Lightning App Builder so administrators can build custom user interfaces without code.

Using these technologies, you can seamlessly customize and easily deploy new apps to mobile devices running Salesforce. In fact, the Salesforce mobile app and Salesforce Lightning Experience are built with Lightning components.

This guide provides you with an in-depth resource to help you create your own standalone Lightning apps, as well as custom Lightning components that can be used in the Salesforce mobile app. You will also learn how to package applications and components and distribute them in the AppExchange.

Why Use the Lightning Component Framework?

The benefits include an out-of-the-box set of components, event-driven architecture, and a framework optimized for performance.

Out-of-the-Box Component Set

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

Rich component ecosystem

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

Performance

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

Event-driven architecture

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

Faster development

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

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

Device-aware and cross browser compatibility

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

Open Source Aura Framework

The Lightning Component framework is built on the open source Aura framework. The Aura framework enables you to build apps completely independent of your data in Salesforce.

The Aura framework is available at `https://github.com/forcedotcom/aura`. Note that the open source Aura framework has features and components that are not currently available in the Lightning Component framework. We are working to surface more of these features and components for Salesforce developers.

The sample code in this guide uses out-of-the-box components from the Aura framework, such as `aura:iteration` and `ui:button`. The `aura` namespace contains components to simplify your app logic, and the `ui` namespace contains components for user interface elements like buttons and input fields. The `force` namespace contains components specific to Salesforce.

Components

Components are the self-contained and reusable units of an app. They represent a reusable section of the UI, and can range in granularity from a single line of text to an entire app.

The framework includes a set of prebuilt components. For example, components that come with the Lightning Design System styling are available in the `lightning` namespace. These components are also known as the base Lightning components. You can assemble and configure components to form new components in an app. Components are rendered to produce HTML DOM elements within the browser.

A component can contain other components, as well as HTML, CSS, JavaScript, or any other Web-enabled code. This enables you to build apps with sophisticated UIs.

The details of a component's implementation are encapsulated. This allows the consumer of a component to focus on building their app, while the component author can innovate and make changes without breaking consumers. You configure components by setting the named attributes that they expose in their definition. Components interact with their environment by listening to or publishing events.

Events

Event-driven programming is used in many languages and frameworks, such as JavaScript and Java Swing. The idea is that you write handlers that respond to interface events as they occur.

A component registers that it may fire an event in its markup. Events are fired from JavaScript controller actions that are typically triggered by a user interacting with the user interface.

There are two types of events in the framework:

- **Component events** are handled by the component itself or a component that instantiates or contains the component.
- **Application events** are handled by all components that are listening to the event. These events are essentially a traditional publish-subscribe model.

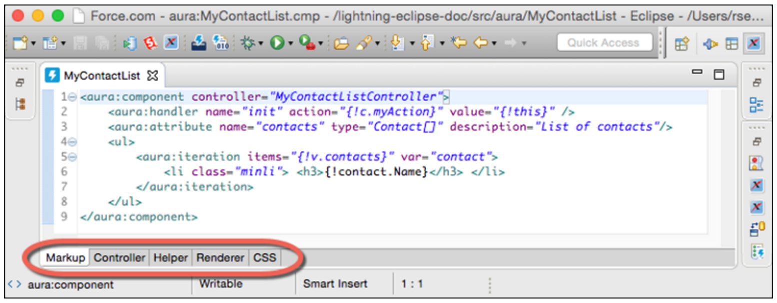
You write the handlers in JavaScript controller actions.

Notable Lightning Components Features in the Force.com IDE

The Force.com IDE includes lots of cool features to make working with Lightning components easier. Here are a few of our favorites.

Multi-Page Editor

Easily access and edit all the resources in your Lightning component bundle. The multi-page editor opens by default when you open a component bundle.



```
<aura:component controller="MyContactListController">
    <aura:handler name="init" action="{!c.myAction}" value="{!this}" />
    <aura:attribute name="contacts" type="Contact[]" description="List of contacts"/>
    <ul>
        <aura:iteration items="{!v.contacts}" var="contact">
            <li class="minli"> <h3>{!contact.Name}</h3> </li>
        </aura:iteration>
    </ul>
</aura:component>
```

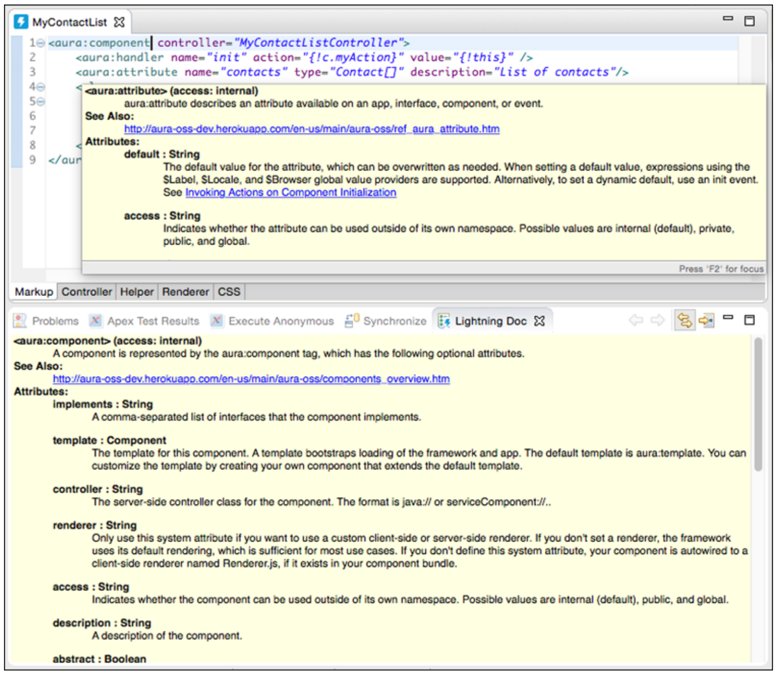
Force.com - aura:MyContactList.cmp - /lightning-eclipse-doc/src/aura/MyContactList - Eclipse - /Users/rse...
Quick Access
MyContactList
Markup
Controller
Helper
Renderer
CSS
aura:component
Writable
Smart Insert
1 : 1

Documentation Browser

Everyone loves documentation! In the Force.com IDE, we give you two handy ways to peruse it.

Access documentation for built-in tags and functions without leaving the IDE. To open the Lightning Doc view, select **Window** > **Show View** > **Other** > **Other** > **Lightning Doc**.

You can also view an element's documentation by hovering over the element in your code.



```
<aura:component controller="MyContactListController">
    <aura:handler name="init" action="{!c.myAction}" value="{!this}" />
    <aura:attribute name="contacts" type="Contact[]" description="List of contacts"/>
```

<aura:attribute> (access: internal)
aura:attribute describes an attribute available on an app, interface, component, or event.
See Also:
http://aura-oss-dev.herokuapp.com/en-us/main/aura-oss/ref_aura_attribute.htm
Attributes:
default : String
The default value for the attribute, which can be overwritten as needed. When setting a default value, expressions using the $Label, $Locale, and $Browser global value providers are supported. Alternatively, to set a dynamic default, use an init event. See Invoking Actions on Component Initialization

access : String
Indicates whether the attribute can be used outside of its own namespace. Possible values are internal (default), private, public, and global.

Press 'F2' for focus

<aura:component> (access: internal)
A component is represented by the aura:component tag, which has the following optional attributes.
See Also:
http://aura-oss-dev.herokuapp.com/en-us/main/aura-oss/components_overview.htm
Attributes:
implements : String
A comma-separated list of interfaces that the component implements.

template : Component
The template for this component. A template bootstraps loading of the framework and app. The default template is aura:template. You can customize the template by creating your own component that extends the default template.

controller : String
The server-side controller class for the component. The format is java:// or serviceComponent://..

renderer : String
Only use this system attribute if you want to use a custom client-side or server-side renderer. If you don't set a renderer, the framework uses its default rendering, which is sufficient for most use cases. If you don't define this system attribute, your component is autowired to a client-side renderer named Renderer.js, if it exists in your component bundle.

access : String
Indicates whether the component can be used outside of its own namespace. Possible values are internal (default), public, and global.

description : String
A description of the component.

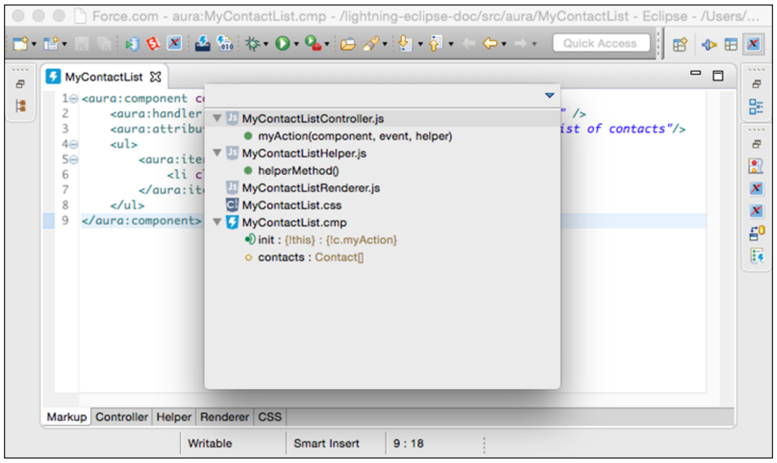
abstract : Boolean

Add a `description` attribute to each `aura:component` that you create, so that you can take full advantage of these useful documentation-related features.

Navigation

Outline View

Quickly locate and navigate between the elements in your Lightning component bundle. To open Outline View, press Ctrl+O (Cmd+O on macOS).

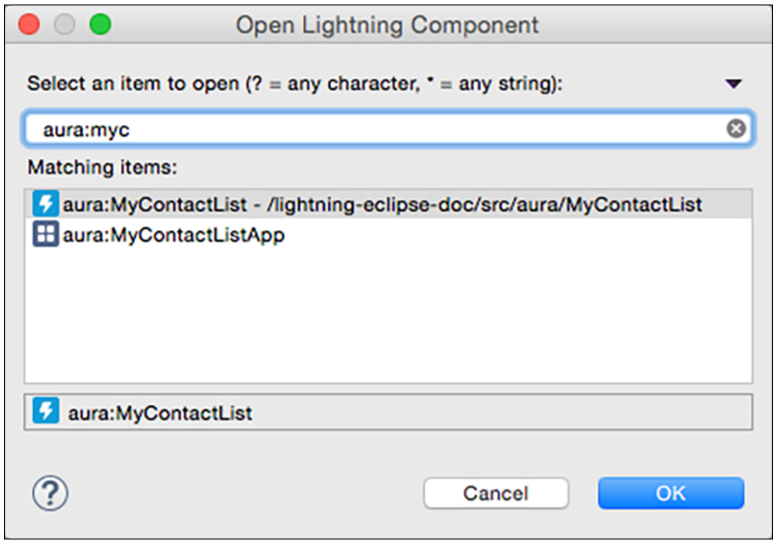

Force.com - aura:MyContactList.cmp - /lightning-eclipse-doc/src/aura/MyContactList - Eclipse - /Users/...
Quick Access
MyContactList
Markup
Controller
Helper
Renderer
CSS
Writable
Smart Insert
9 : 18

```
<aura:component c
    <aura:handler                                  " />
    <aura:attribu                                  ist of contacts"/>
    <ul>
        <aura:ite
            <li c
        </aura:it
    </ul>
</aura:component>
```

```
▼ MyContactListController.js
    ● myAction(component, event, helper)
▼ MyContactListHelper.js
    ● helperMethod()
  MyContactListRenderer.js
  MyContactList.css
▼ MyContactList.cmp
    init : {!this} : {!c.myAction}
    contacts : Contact[]
```

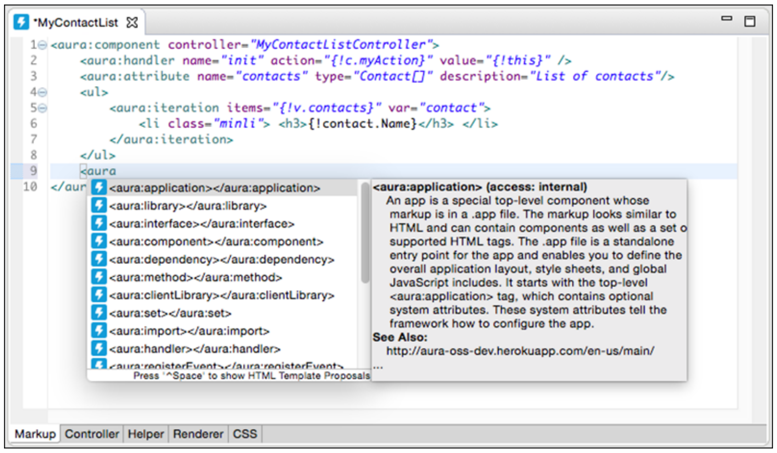
Open Dialog

Quickly locate and navigate between your Lightning component bundles. To display the Open Dialog, press Ctrl+Shift+A (Cmd+Shift+A on macOS).


Open Lightning Component
Select an item to open (? = any character, * = any string):
aura:myc
Matching items:
aura:MyContactList - /lightning-eclipse-doc/src/aura/MyContactList
aura:MyContactListApp
aura:MyContactList
Cancel
OK

Auto-Completion

Save time and avoid typos with the auto-completion feature for markup, JavaScript, and CSS files. Some auto-completion is provided by default as you type. To invoke more auto-completion options, press Ctrl+Space when you're editing a file.



```
<aura:component controller="MyContactListController">
    <aura:handler name="init" action="{!c.myAction}" value="{!this}" />
    <aura:attribute name="contacts" type="Contact[]" description="List of contacts"/>
    <ul>
        <aura:iteration items="{!v.contacts}" var="contact">
            <li class="minli"> <h3>{!contact.Name}</h3> </li>
        </aura:iteration>
    </ul>
    <aura
</aur
```

```
<aura:application></aura:application>
<aura:library></aura:library>
<aura:interface></aura:interface>
<aura:component></aura:component>
<aura:dependency></aura:dependency>
<aura:method></aura:method>
<aura:clientLibrary></aura:clientLibrary>
<aura:set></aura:set>
<aura:import></aura:import>
<aura:handler></aura:handler>
<aura:registerEvent></aura:registerEvent>
```

<aura:application> (access: internal)
An app is a special top-level component whose markup is in a .app file. The markup looks similar to HTML and can contain components as well as a set o supported HTML tags. The .app file is a standalone entry point for the app and enables you to define the overall application layout, style sheets, and global JavaScript includes. It starts with the top-level <aura:application> tag, which contains optional system attributes. These system attributes tell the framework how to configure the app.
See Also:
http://aura-oss-dev.herokuapp.com/en-us/main/
...

Set Up the Force.com IDE with Lightning Components Support

Support for the Lightning Component Framework is generally available in Summer '16 (Force.com IDE v37.0) and later versions of the Force.com IDE. Be sure to select **Force.com Lightning Support** when you install or update the Force.com IDE plug-in.

For Force.com IDE installation instructions, see Install the Force.com IDE Plug-In.

Explore Lightning Components in the Force.com IDE

Learn how to create a simple Lightning component that renders a list of contacts from your org. Start by creating an Apex controller class, and then create a Lightning component and an event handler. Finally, render the list of contacts in the component.

Enable My Domain in Your Development Org
For security purposes, Lightning components require you to define a custom Salesforce domain name for your org. Setting up a new domain is simple, but it takes a few minutes before it's available for use. When you enable My Domain, references and links to Lightning resources are in the format `https://`*`yourDomain`*`.lightning.force.com`. Use a Developer Edition org for this exercise.

Roll Out My Domain to Your Org
Did you get your activation email? From the email, click the link to get back to the My Domain wizard. It takes you to Step 3, where you test the links to your subdomain URLs before rolling out the subdomain to your org. Even though you don't have users to deploy it to in your DE org, you must still roll out My Domain to make your custom Lightning components available in Lightning pages, in the Lightning App Builder, and for standalone apps.

Create an Apex Controller Class in the Force.com IDE
Create a class to access data from contacts.

Create a Lightning Component Bundle in the Force.com IDE
A Lightning component is a combination of markup, JavaScript, and CSS. Create a Lightning component bundle that contains these files.

Add an Event Handler in the Force.com IDE
The event handler calls your `myAction` function when the handler is initialized and retrieves a list of contacts.

Create a Lightning Application in the Force.com IDE
Render and preview your contact list in a Lightning application.

Customize Your Style Sheet in the Force.com IDE

A Lightning component can specify its own cascading style sheet (CSS). The .css file is included when you create a component bundle in the Force.com IDE.

Wrap Up Your Exploration of Lightning Components in the Force.com IDE

You've now learned how to create a Lightning component that renders a list of contacts from your org. You created an Apex controller class, and then added a Lightning component and an event handler. You rendered the list of contacts in the component, both with and without pizazz. What's next?

Enable My Domain in Your Development Org

For security purposes, Lightning components require you to define a custom Salesforce domain name for your org. Setting up a new domain is simple, but it takes a few minutes before it's available for use. When you enable My Domain, references and links to Lightning resources are in the format `https://`*`yourDomain`*`.lightning.force.com`. Use a Developer Edition org for this exercise.

Enable My Domain in Your Org

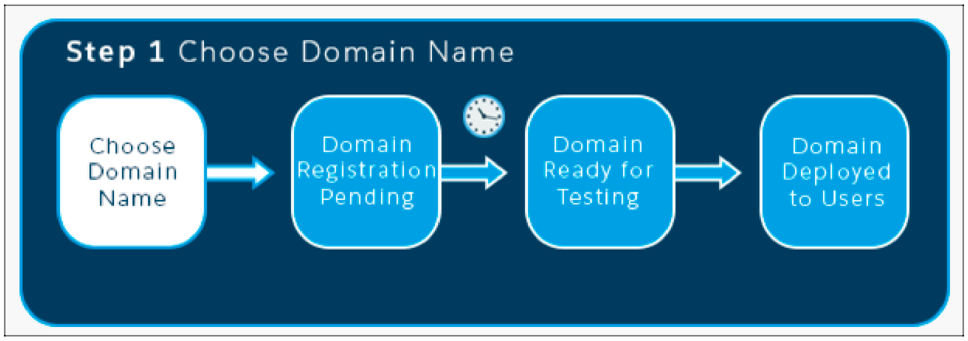
Before we get to the heart of creating Lightning components, let's use Salesforce My Domain to set up a subdomain. Is setting a My Domain a requirement? Yes, if you want to use Lightning components in Lightning tabs, Lightning pages, or as standalone apps. Salesforce requires My Domain as a security measure to help prevent malicious attacks—just in case a security hole lies hidden deep within a third-party or custom component.

If you already have My Domain enabled in your DE org or use a Trailhead Playground org, skip this section and the next one. You already have My Domain set up.

If you don't have a subdomain yet, it's easy to set one up.

Every Salesforce org is set up within the `salesforce.com` domain with a URL like `https://na30.salesforce.com`. With My Domain, you define your own domain, or a subdomain, within the `salesforce.com` domain. Your new URL looks something like: `https://`*`yourDomain`*`.my.salesforce.com`.

Use the My Domain wizard to create a subdomain.



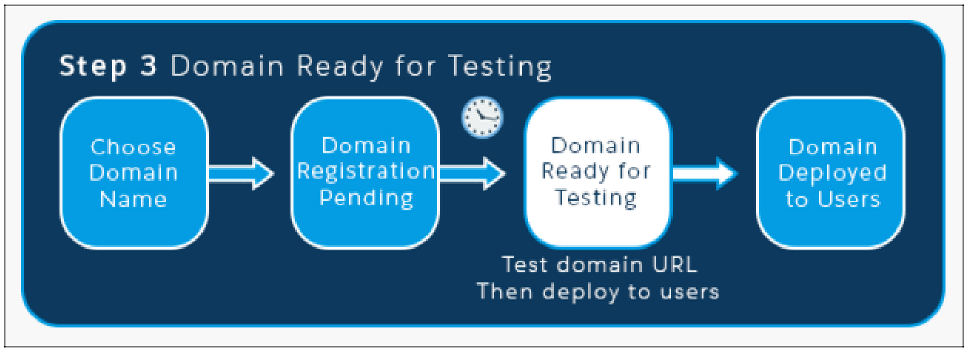
1. From Setup, enter *`My Domain`* in the `Quick Find` box, then select **My Domain**.
2. Enter the name for your subdomain after `https://` and click **Check Availability**. Typically, a subdomain is your company name, but you can use any name as long as it's unique. If this name is already taken, choose another one.
3. Click **Register Domain**.

Salesforce updates its domain registries with your new subdomain. When it's done, you receive an email with a subject like, "Your Developer Edition domain ready for testing." It takes just a few minutes.

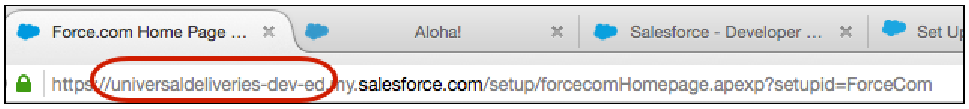
Important: Did you catch that last part? It can take a few minutes before your domain is available. You can't move to the next step until you get the activation email.

Roll Out My Domain to Your Org

Did you get your activation email? From the email, click the link to get back to the My Domain wizard. It takes you to Step 3, where you test the links to your subdomain URLs before rolling out the subdomain to your org. Even though you don't have users to deploy it to in your DE org, you must still roll out My Domain to make your custom Lightning components available in Lightning pages, in the Lightning App Builder, and for standalone apps.



1. Click the link in the activation email to log in to your Salesforce subdomain. It takes you to your Salesforce org.
   Notice that the URL in the browser address bar shows your new subdomain name. Right now, you're the only one who has this URL.



https://universaldeliveries-dev-ed.my.salesforce.com/setup/forcecomHomepage.apexp?setupid=ForceCom

2. Click around your org to make sure that links point to your new domain. You probably haven't created links in your DE org, so we can go on. (When creating a domain in a production org, this important step is easily overlooked.)
3. From the My Domain page, click **Deploy to Users**, and then click **OK**. Deploying a subdomain rolls out the new subdomain URL throughout your org. Now all your users see the subdomain URL in the browser address bar.
4. Step 4 of the wizard displays configuration options, which we can ignore for now.

Congratulations, you've set up My Domain! When setting up My Domain in a production org, you have a few more steps. Learn more by completing the My Domain unit of the User Authentication module. Now that you've protected—and branded—your org with a subdomain, let's go on.

Create an Apex Controller Class in the Force.com IDE

Create a class to access data from contacts.

1. If you don't already have an Eclipse project for this org, complete the steps in Create a Force.com Project.
2. To save changes to the server automatically, right-click the name of the project that you created, then select **Force.com** > **Work Online**. When Work Online is enabled, the icon next to your project name looks like this: , rather than like this: . Working online can lead to long save times, but your Developer Edition org probably has a small codebase, so save times aren't a concern.
3. Right-click your project in the Package Explorer, then select **New** > **Apex Class**.

4. Enter *`MyContactListController`* in the Name field, and then click **Finish**.
5. In the body of the class, between the {} braces, enter this code.

```
    @AuraEnabled
    public static List<Contact> getContacts() {
        return [SELECT Id, Name, Email, Title, Phone FROM Contact];
    }
```

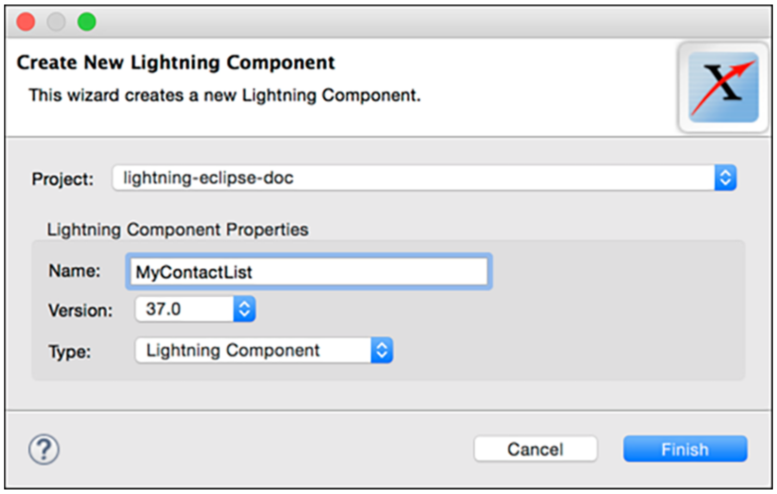
   `@AuraEnabled` enables client- and server-side access to the controller method.

6. To save your updated class, press Ctrl+S or (Cmd+S on macOS).

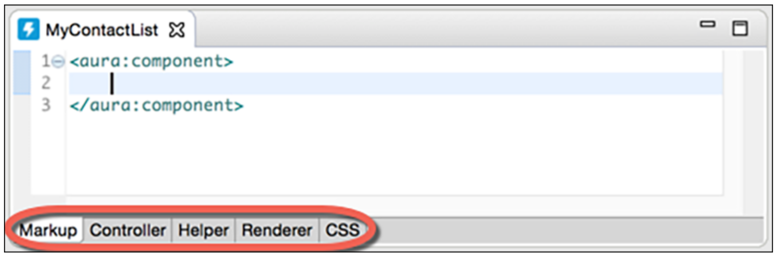
Create a Lightning Component Bundle in the Force.com IDE

A Lightning component is a combination of markup, JavaScript, and CSS. Create a Lightning component bundle that contains these files.

1. In the Force.com IDE, select **File** > **New** > **Lightning Bundle**.
2. Enter *`MyContactList`* in the Name field, and then click **Finish**.


Create New Lightning Component
This wizard creates a new Lightning Component.
Project: lightning-eclipse-doc
Lightning Component Properties
Name: MyContactList
Version: 37.0
Type: Lightning Component

3. In the Package Explorer, expand the `src`, `aura`, and `MyContactList` directories.
4. Double-click **MyContactList.cmp**.
5. Note the Markup, Controller, Helper, Renderer, and CSS tabs at the bottom of your source panel. When you created the MyContactList component bundle, the bundle's controller, helper, renderer, and CSS files were created for you. You can easily navigate among the items in your component bundle using these tabs.



```
<aura:component>

</aura:component>
```

Note: If you don't need all these files, you can delete them when you're ready to deploy your component. To delete a file, right-click the file in the Package Explorer and then choose **Delete**.

6. Replace the `MyContactList.cmp` line that contains `<aura:component>` with the following line.

```
<aura:component controller="MyContactListController">
```

7. To save your updated component, press Ctrl+S or (Cmd+S on macOS).
8. Click the **Controller** tab.
9. In the body of the `myAction` JavaScript function, add this code.

```
var action = component.get("c.getContacts");
action.setCallback(this, function(data) {
component.set("v.contacts", data.getReturnValue());
});
$A.enqueueAction(action);
```

10. To save your updated controller, press Ctrl+S or (Cmd+S on macOS).

Your component can now access data from the `MyContactListController` Apex controller.

Add an Event Handler in the Force.com IDE

The event handler calls your `myAction` function when the handler is initialized and retrieves a list of contacts.

1. Use the outline view to navigate to the markup file in the `MyContactList` component bundle: Press Ctrl+O (Cmd+O on macOS). An outline of the contents of the component bundle that you're working on appears. Double-click `MyContactList.cmp`.
2. On the line after the opening `aura:component` tag, add this markup.

```
<aura:handler name="init" action="{!c.myAction}" value="{!this}" />
```

   This event handler calls the `myAction` client-side controller to handle initialization.

3. Let's check out the documentation for the `aura:handler` component. Hover your cursor over `aura:handler` in the source panel. This action brings up information about `aura:handler` and its attributes. If you want to keep that documentation in view while you work on your code, select **Window** > **Show View** > **Other** > **Other** > **Lightning Doc**. Everyone knows that good documentation crucial for successful development, and the Force.com IDE provides documentation for the component you're editing right in your development environment.
4. Now, right underneath the `aura:handler`, add an `aura:attribute` component to make the list of contacts accessible to your Lightning component. Let's use autocompletion to help with this task. Type *`<aura:a`*, and examine the options that appear. We want `<aura:attribute>`, so press the down arrow on your keyboard until it's highlighted, then press Enter.
5. Between the quote marks for the `aura:attribute` component's `name` attribute, type *`contacts`*. Between the quote marks for the `aura:attribute` component's `type` attribute, type *`Contact[]`*. You can delete the `description=""` attribute, or you can add a description of your choice. The `aura:attribute` component now looks something like this.

```
<aura:attribute name="contacts" type="Contact[]" description="List of contacts" />
```

6. To save your updated component, press Ctrl+S or (Cmd+S on macOS).

Create a Lightning Application in the Force.com IDE

Render and preview your contact list in a Lightning application.

1. In the Force.com IDE, make sure that you're on the Markup tab of the `MyContactList` component.

2. Add this markup below the handler and attribute tags that you created earlier.

```
<ul>
<aura:iteration items="{!v.contacts}" var="contact">
<li class="minli"> <h3>{!contact.Name}</h3> </li>
</aura:iteration>
</ul>
```

   `aura:iteration` iterates over the list of contacts and displays their names.

3. To save your updated component, press Ctrl+S or (Cmd+S on macOS).
4. Select **File** > **New** > **Lightning Bundle**.
5. Enter *`MyContactListApp`* in the File field.
6. Select **Lightning Application** from the Type drop-down menu.
7. Click **Finish**.
8. In the Package Explorer, expand the `src`, `aura`, and `MyContactListApp` directories.
9. Double-click **MyContactListApp.app**.
10. In the body of `MyContactListApp.app`, between the `<aura:application>` tags, add this markup.

```
<c:MyContactList />
```

11. To save your updated application, press Ctrl+S or (Cmd+S on macOS).
12. If you're working in Offline mode, right-click **MyContactListApp.app** in the Package Explorer and select **Force.com** > **Save to Server**.
13. In your web browser, navigate to `https://`*`yourDomain`*`.lightning.force.com/c/MyContactListApp.app`. Behold, a list of the contacts in your org!

Customize Your Style Sheet in the Force.com IDE

A Lightning component can specify its own cascading style sheet (CSS). The .css file is included when you create a component bundle in the Force.com IDE.

The contact list we created was okay, but it could use more pizazz. Let's spice it up a bit.

1. Make sure that the active file in the source panel is one of your `MyContactList` files (not a `MyContactListApp` file).
2. Click the **CSS** tab at the bottom of the source panel.
3. Replace the generated CSS with this CSS.

```
.THIS {
background:url("https://bit.ly/1Dy6Eji") no-repeat center center;
min-height:100%;
background-size:cover;
color: darkblue;
font: Arial;
margin: 0;
padding-top:10px;
padding-bottom:10px;
}
.THIS ul {
list-style-type: none;
width: 90%;
padding: 5px;
position: relative;
margin: 0 auto;
}
.THIS h3 {
font: bold 20px/1.5 Arial;
}
.THIS .minli {
padding: 10px;
margin-bottom: 10px;
overflow: auto;
background: rgba(255,255,255,.4);
border-radius: 3px;
width:95%;
}
.THIS li:hover {
cursor: pointer;
}
```

4. To save your updated CSS, press Ctrl+S or (Cmd+S on macOS).
5. Refresh the browser page that's displaying the preview of your application. If you've closed that page, you can find it again at `https://`*`yourDomain`*`.lightning.force.com/c/MyContactListApp.app`.

Isn't that better? Now *that* is what we call pizazz.

Wrap Up Your Exploration of Lightning Components in the Force.com IDE

You've now learned how to create a Lightning component that renders a list of contacts from your org. You created an Apex controller class, and then added a Lightning component and an event handler. You rendered the list of contacts in the component, both with and without pizazz. What's next?

Lightning components aren't just for building web pages. Use them to build responsive apps for phones, tablets, and even watches to help your team move Lightning fast.

You can build components from scratch, but often it's not necessary. Lots of other developers have made components that they've built available in the Components section of the AppExchange, and many of them are free. Why reinvent the wheel when plenty of wheels are already in circulation?

For a tour of Lightning components in the Force.com IDE, watch Platform Developer Tools engineer Nick Chen's presentation at TrailheaDX '16.

Ready to dive into learning about Lightning components? Check out the Lightning Components Basics module on Trailhead. When you're ready to start sorting out your nuts and bolts, peruse the *Lightning Components Developer Guide*.

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**.

Save to Server

Refresh from Server

Synchronize with Server

SEE ALSO:

Save to Server

Refresh from Server

Synchronize with Server

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** refreshes only 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 you update your home org in both the IDE and in the browser, your IDE project files might 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 the Synchronize with Server option 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 org:

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 org. 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 org.
   - **Apply project to server**: Completely replaces the corresponding item in the home org 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 org 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 org.
3. When you’ve resolved all conflicts, click the save icon. Then close the Synchronize view.

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

CHAPTER 3 Test Code with the Force.com IDE

In this chapter ...

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 Lightning 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 Lightning 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 Lightning 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 toward your Apex storage limits. Creating a test class is done like any other class.
- Create test suites to manage your tests.
  - To create a test suite, select **File** > **New** > **Apex Test Suite**, or right-click your project in the Package Explorer and then select **New** > **Apex Test Suite**.
  - To edit a test suite, edit its XML file: `/src/testSuites/YourTestSuite.testSuite`. Apex test suites are represented in Metadata API as `ApexTestSuite` components.

- 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 *Apex Code Developer Guide*.

SEE ALSO:

Apex Test Results View

Debug Logs for Apex Test Results

Apex Code Settings: Log Category and Log Level

Troubleshoot the Force.com IDE: Debug Mode

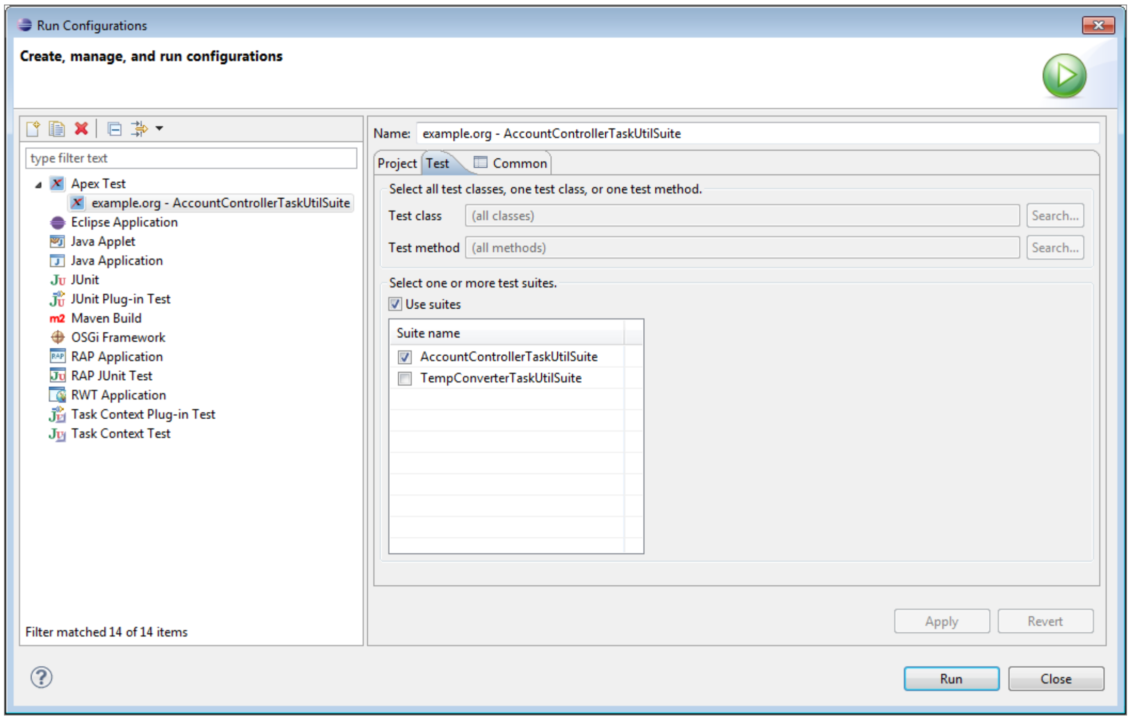
Apex Test Results View

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:

1. Select **Run** > **Run Configurations** > **Apex Test** > **New launch configuration** ( ).
2. Select the **Test** tab.
3. Add tests to your run configuration.
   - To add a test class or test method, click one of the Search buttons. To select a method, first select the class that contains that method.
   - To create a run of test suites:
     a. Select **Use suites**.
     b. Select one or more test suites.
4. Click **Apply**.
5. To execute the selected test run configuration, click **Run**.



Name: example.org - AccountControllerTaskUtilSuite

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

Test class (all classes)

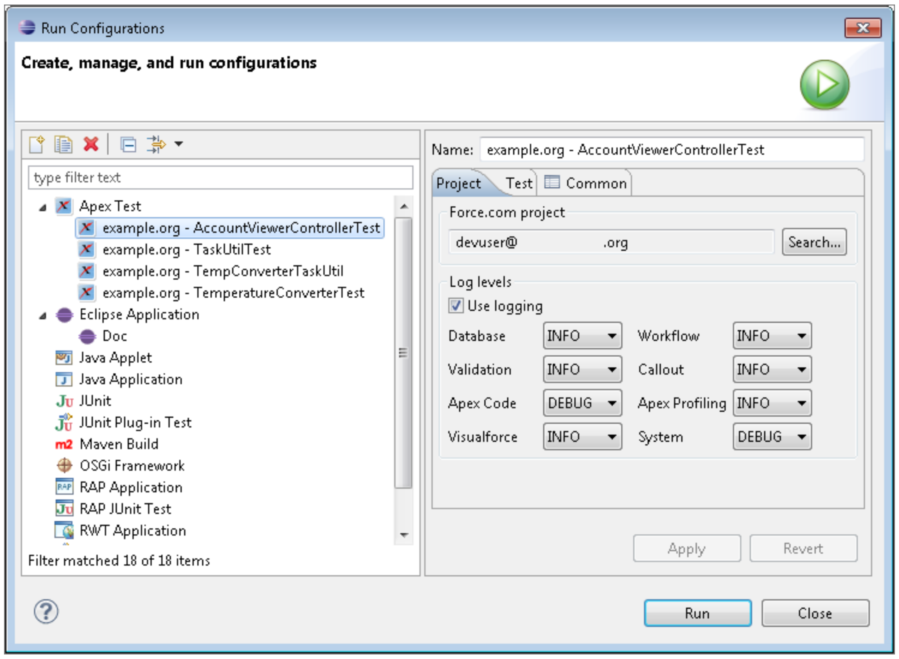
Test method (all methods)

Select one or more test suites.

[✓] Use suites

Suite name
[✓] AccountControllerTaskUtilSuite
[ ] TempConverterTaskUtilSuite

To set log levels for an Apex test run, use your run configuration's Project tab. Logging settings in your project's properties apply only when you're deploying code.



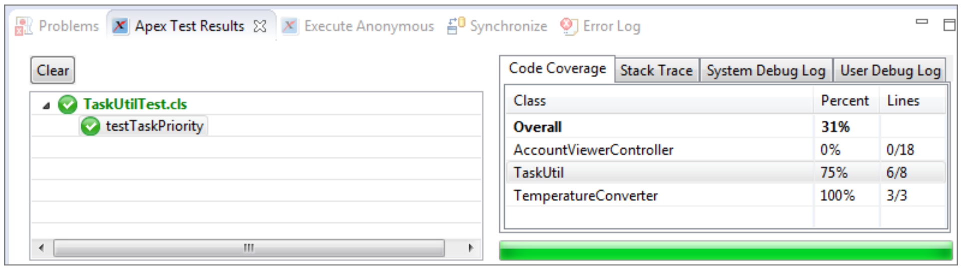
Name: example.org - AccountViewerControllerTest

Force.com project: devuser@ .org

[✓] Use logging

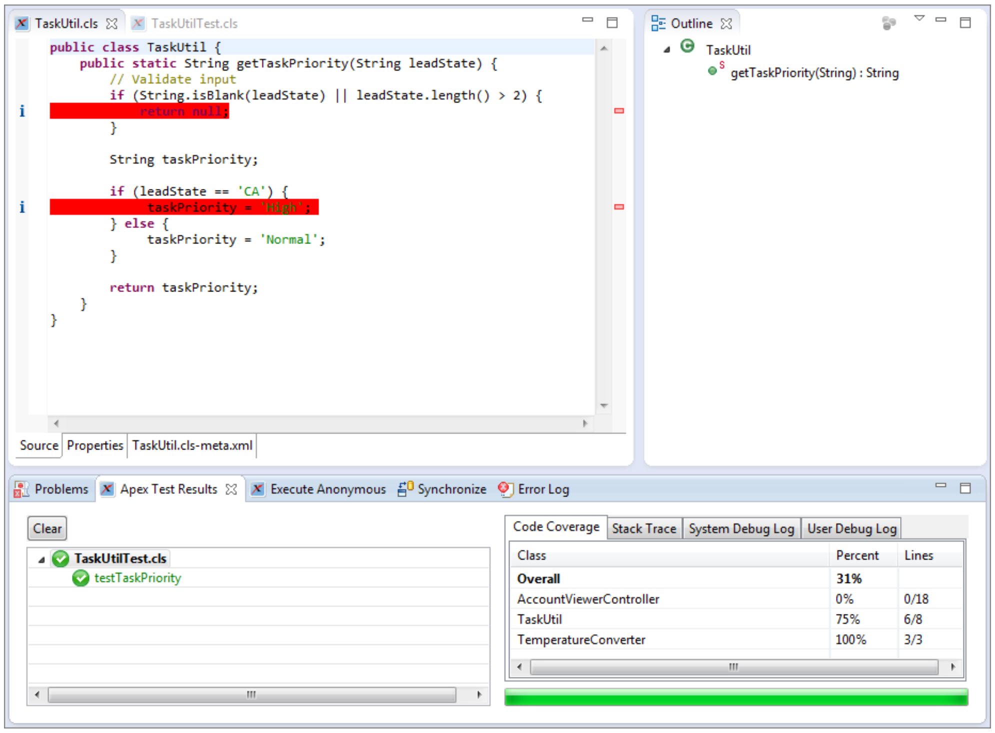
Database	INFO	Workflow	INFO
Validation	INFO	Callout	INFO
Apex Code	DEBUG	Apex Profiling	INFO
Visualforce	INFO	System	DEBUG

After 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	**31%**	
AccountViewerController	0%	0/18
TaskUtil	75%	6/8
TemperatureConverter	100%	3/3

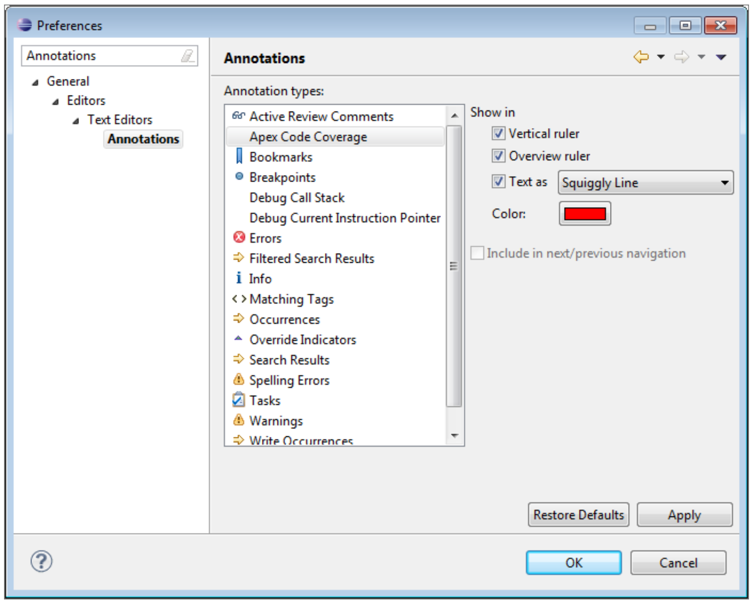
To view code coverage highlighting, double-click a class name in the Apex Test Results pane's Code Coverage list. Lines that aren't covered by your unit tests are highlighted. Unlike the Developer Console, which color-codes both covered and uncovered lines, the Force.com IDE highlights only the lines of code that still need coverage.



```
public class TaskUtil {
    public static String getTaskPriority(String leadState) {
        // Validate input
        if (String.isBlank(leadState) || leadState.length() > 2) {
            return null;
        }

        String taskPriority;

        if (leadState == 'CA') {
             taskPriority = 'High';
        } else {
             taskPriority = 'Normal';
        }

        return taskPriority;
    }
}
```

Class	Percent	Lines
Overall	**31%**	
AccountViewerController	0%	0/18
TaskUtil	75%	6/8
TemperatureConverter	100%	3/3

To change the color for your uncovered code or to replace the highlighting with squiggly underlining, select **Window** > **Preferences**. Enter `Annotations` in the filter text box, then select **Annotations** > **Apex Code Coverage**. Make your desired changes, and then click **Apply**.



Show in
[✓] Vertical ruler
[✓] Overview ruler
[✓] Text as Squiggly Line
Color: 🟥
[ ] Include in next/previous navigation

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.

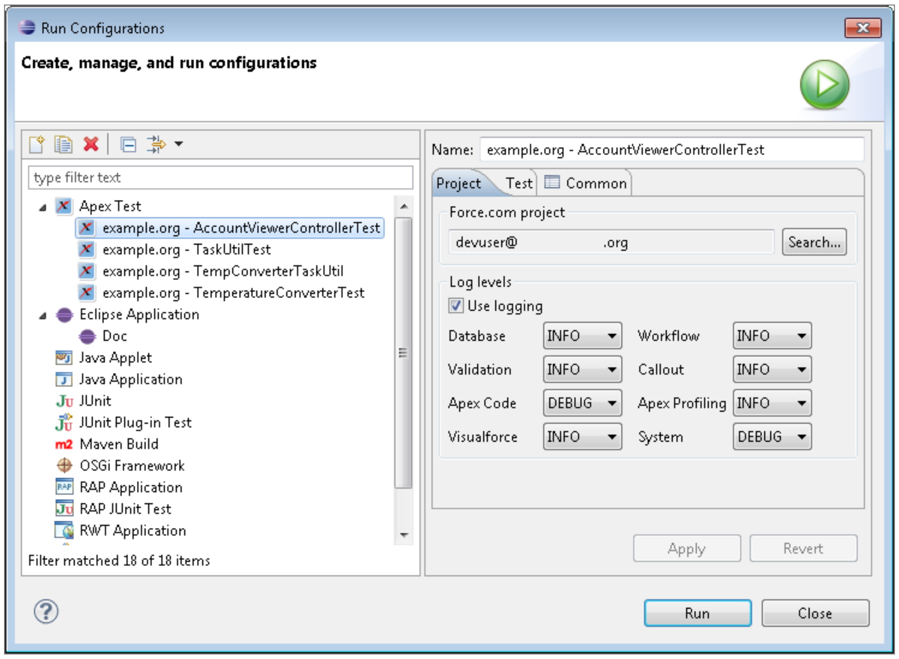
Debug Logs for Apex Test Results

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 define logging levels for Apex test execution:

1. Select **Run** > **Run Configurations** > **Apex Test**.
2. To create a test run configuration, select **New launch configuration** ( ).
3. To customize your log levels, select the Project tab, and then select **Use logging**.



Name: example.org - AccountViewerControllerTest

Force.com project: devuser@ .org

Log levels

☑ Use logging

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

CHAPTER 4 Deploy Code with the Force.com IDE

In this chapter ...

After you have created application components and tested them in your own development org, you usually want to migrate them to a different org. The second org is used for testing, staging, publication, or production use by end users.

Although each metadata component lives and runs in a particular org, the metadata files you work with in the Force.com IDE are portable from one org to another. Deploying is the process of pushing your local project files into a different Salesforce org than your project's home org. 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 project with only the components you want to deploy. For information, see Create a Force.com Project.

Note: If you deploy to a production org, you must meet the Apex testing compliance policy: 75% of your Apex must be covered by unit tests, and all those tests must complete successfully. Also, 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 org. 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 org 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. To execute the deployment, click **Next**.
4. The Deployment Result page displays the result of the deployment. If you don't see Success displayed at the top of the page, 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 here. For more information, see the Salesforce online help topic Restrict Where and When Users Can Log In to Salesforce.
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 5 Get 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.

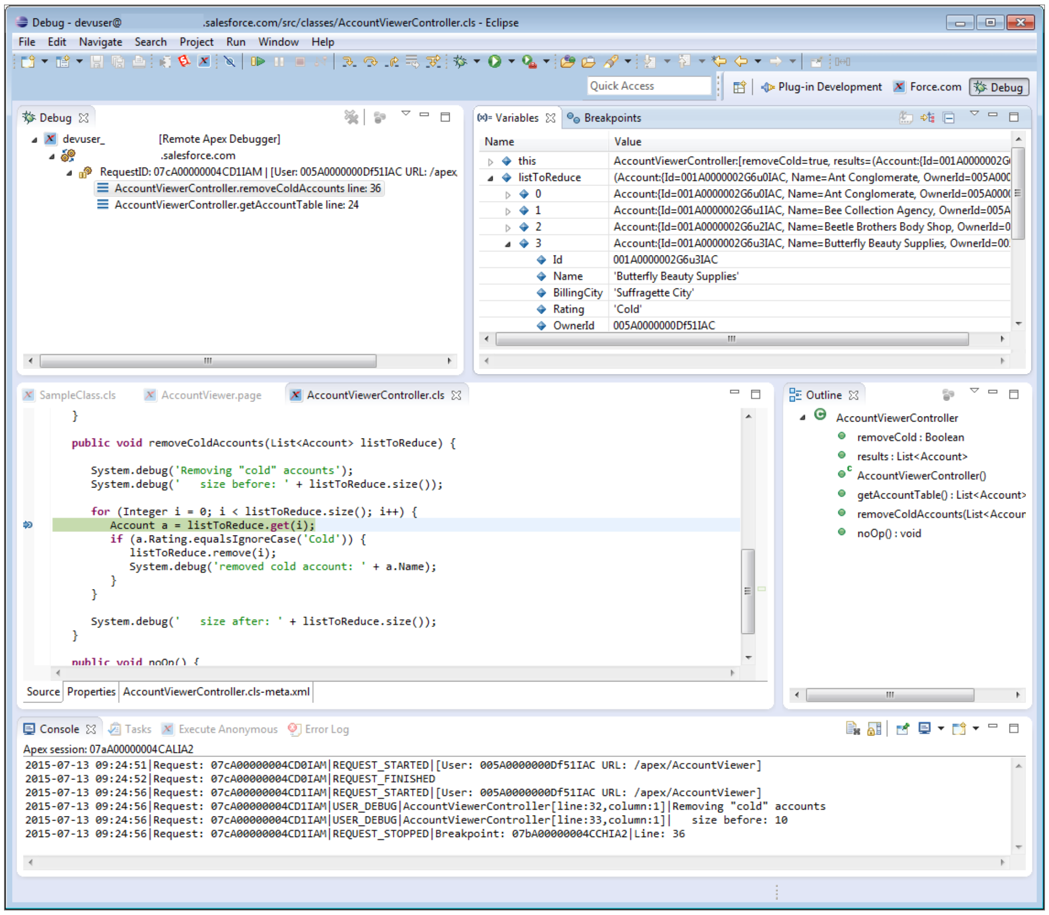
EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Enterprise**, **Performance** (one session included), and **Unlimited** (one session included) Editions

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
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
```

Get Started with the Apex Debugger

This documentation helps you, a Lightning Platform 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 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.

Contact Salesforce to Enable the Apex Debugger
Some services and subscriptions include the debugging feature for an extra cost. For pricing details, contact your Salesforce account executive. After Salesforce grants licenses for the Apex Debugger to your production org, match your production license counts to your sandbox.

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.

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

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.

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.

Whitelist Users and Request Types
Filter which requests are debugged by setting up whitelisting. If you don't use whitelisting, all events in your org trigger debugging during a debugging session. Whitelist users or request types to focus only on the events that are relevant to the problem you're debugging.

Contact Salesforce to Enable the Apex Debugger

Some services and subscriptions include the debugging feature for an extra cost. For pricing details, contact your Salesforce account executive. After Salesforce grants licenses for the Apex Debugger to your production org, match your production license counts to your sandbox.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Enterprise**, **Performance** (one session included), and **Unlimited** (one session included) Editions

User Permissions Needed

To match licenses to sandbox:	Modify All Data

The matching process updates the license counts in your sandbox so that they match production. The process adds licenses to your sandbox that are in production but not in your sandbox, and deletes licenses from your sandbox that aren't in production.

To match licenses to a sandbox org, log in to your sandbox org. From Setup, enter `Company Information` in the Quick Find box, then select **Company Information** > **Match Production Licenses**. This process triggers an alert informing you that the matching process has started. After all licenses have been matched in sandbox, a confirmation email is sent to you. The confirmation email is always sent to the user who initiates the license copy.

Matching production licenses requires that your sandbox and production orgs are on the same Salesforce release. If your sandbox has already been upgraded to the next release but production hasn't been upgraded (for example, during sandbox preview), you can't match production licenses in sandbox.

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

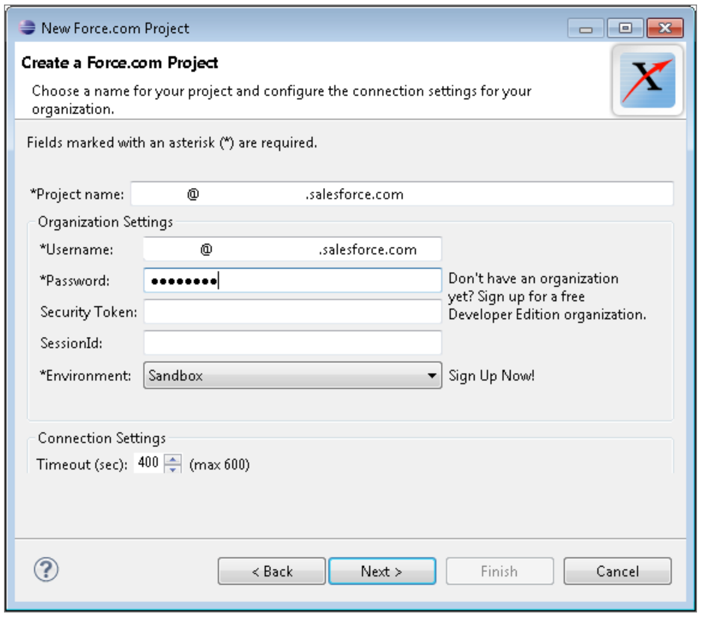
Create a permission set for Apex Debugger users, and assign it to users in your org who plan to use the Debugger.

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, select **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.
< Back
Next >
Finish
Cancel

Fields marked with an asterisk (*) are required.

*Project name:	@ .salesforce.com
Organization Settings	
*Username:	@ .salesforce.com
*Password:	••••••••
Security Token:	
SessionId:	
*Environment:	Sandbox
Connection Settings	
Timeout (sec):	400 (max 600)

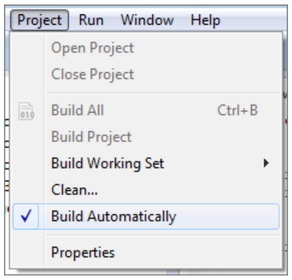
Don't have an organization yet? Sign up for a free Developer Edition organization.

Sign Up Now!

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: .

   Work Online saves your changes to the server if Build Automatically is enabled. If you've enabled Work Online but your changes aren't saving to the server, make sure that you've selected **Project** > **Build Automatically**.



Test Your Debugger Setup

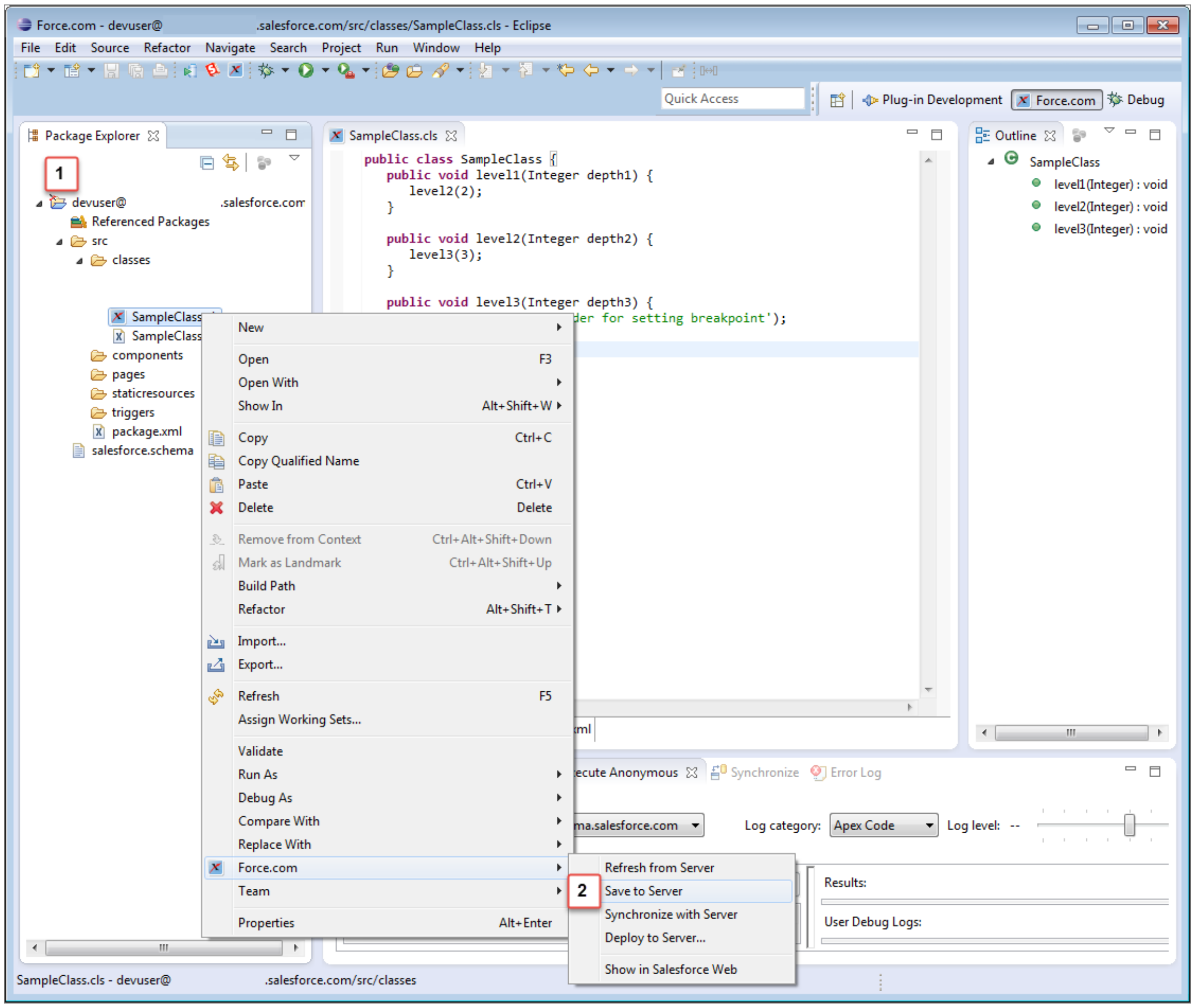
To make sure that everything is functioning properly, create a simple Apex class, set a breakpoint in your code, and then hit the breakpoint using Execute Anonymous.

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).



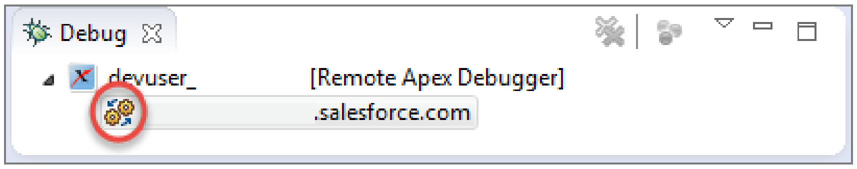
```
public class SampleClass {
   public void level1(Integer depth1) {
      level2(2);
   }

   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**.

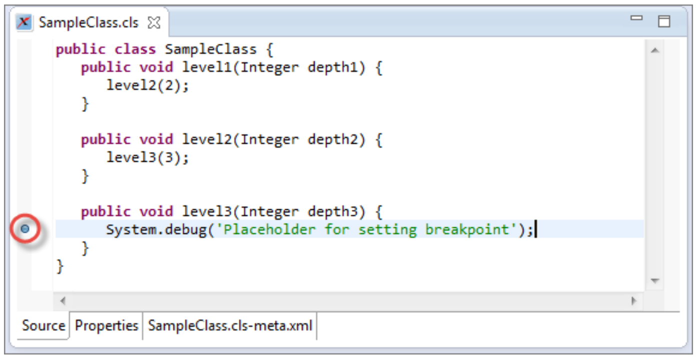
   Tip: After you've completed the initial Debugger setup, you can switch between perspectives by clicking the icons in the upper right corner of your Eclipse window.

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');
   }
}
```

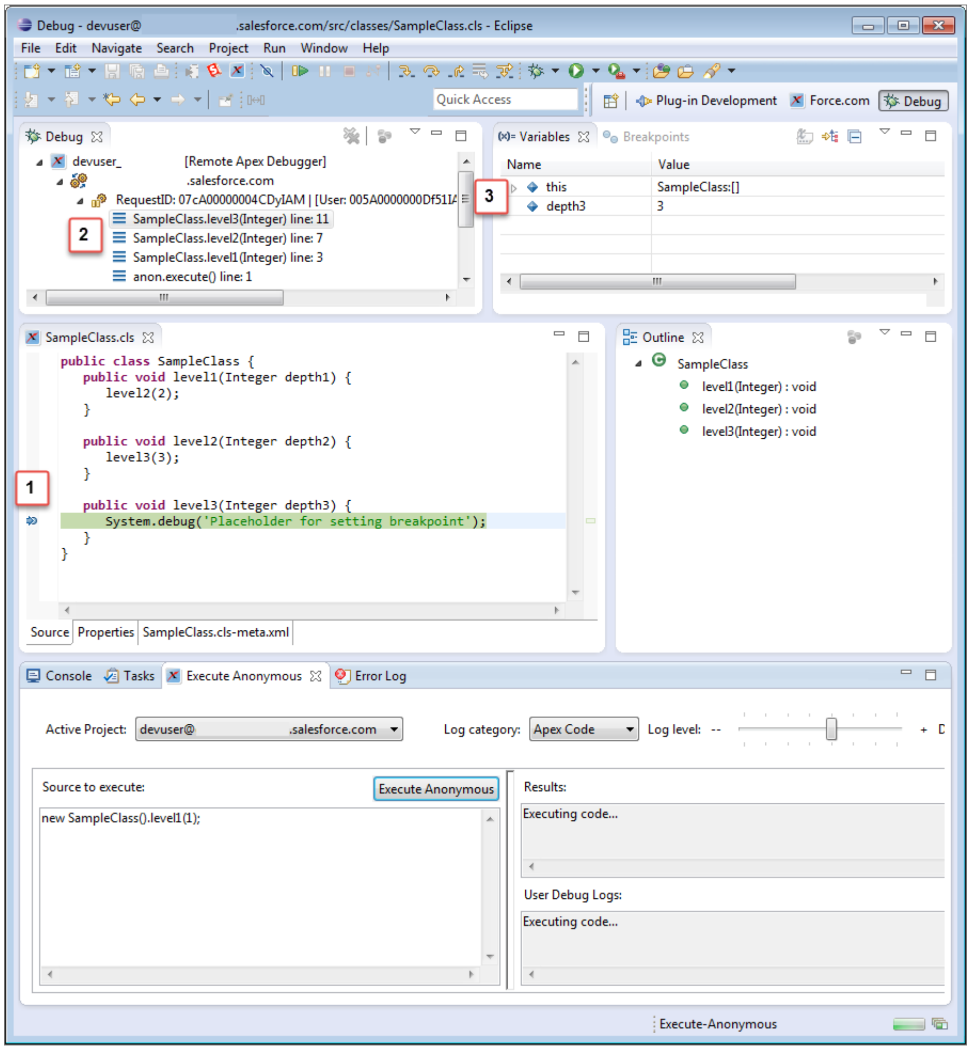
14. Select the **Execute Anonymous** tab at the bottom of the screen. If the Execute Anonymous tab isn't available, choose **Window** > **Show View** > **Other** > **Force.com** and add it.
15. Make sure that your project is the Active Project.
16. In the Source to execute field, enter:

```
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:

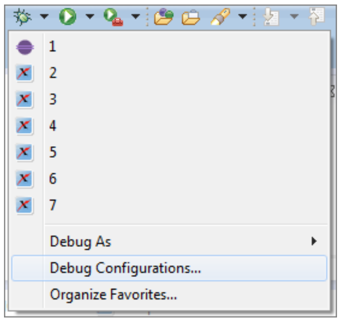
```
Executing code...
```

Execute-Anonymous

Whitelist Users and Request Types

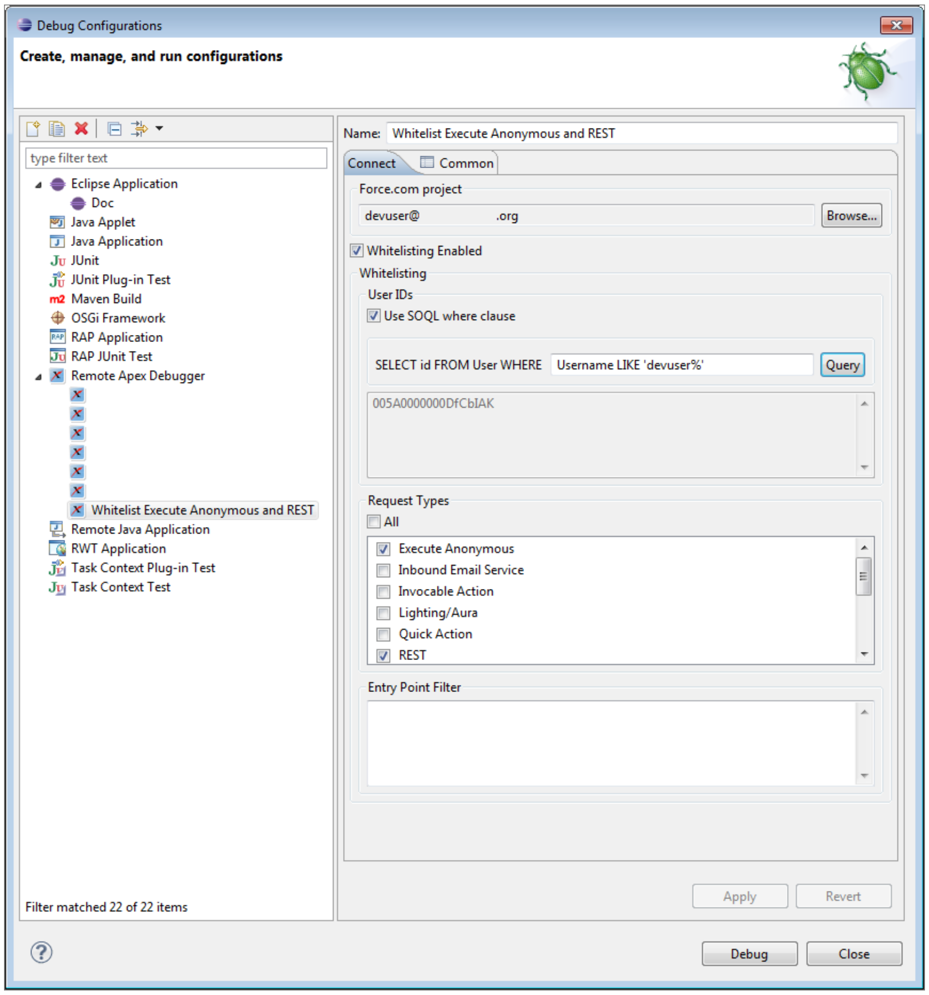
Filter which requests are debugged by setting up whitelisting. If you don't use whitelisting, all events in your org trigger debugging during a debugging session. Whitelist users or request types to focus only on the events that are relevant to the problem you're debugging.

1. To access your debug configurations, select the arrow next to the debug icon ( ), and then select **Debug Configurations**.



Typically, modifying an Eclipse debug configuration during a debugging session doesn't affect active debug targets. However, whitelist modifications affect both current and future sessions.

2. To set up whitelisting, select **Whitelisting Enabled** in your Remote Apex Debugger configuration.



Name: Whitelist Execute Anonymous and REST

Force.com project: devuser@ .org

☑ Whitelisting Enabled

Whitelisting

User IDs

☑ Use SOQL where clause

SELECT id FROM User WHERE Username LIKE 'devuser%'

005A0000000DfCbIAK

Request Types

☐ All

☑ Execute Anonymous
☐ Inbound Email Service
☐ Invocable Action
☐ Lighting/Aura
☐ Quick Action
☑ REST

Entry Point Filter

3. To filter by user ID, enter a SOQL condition expression in the `SELECT id FROM User WHERE` field. Or enter a comma-separated list of user IDs in the text field in the User IDs section.

4. To filter by request type, select one or more entry points, or enter a regular expression in the `Entry Point Filter` field. For example, to whitelist requests made by the Visualforce page `myPage`, enter *`.*/apex/myPage.apexp`*.

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.

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 how to fix the problem in `AccountViewerController.cls`. But you can't save changes to code while a debugging session is in process. Terminate the debugging session, and 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 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!

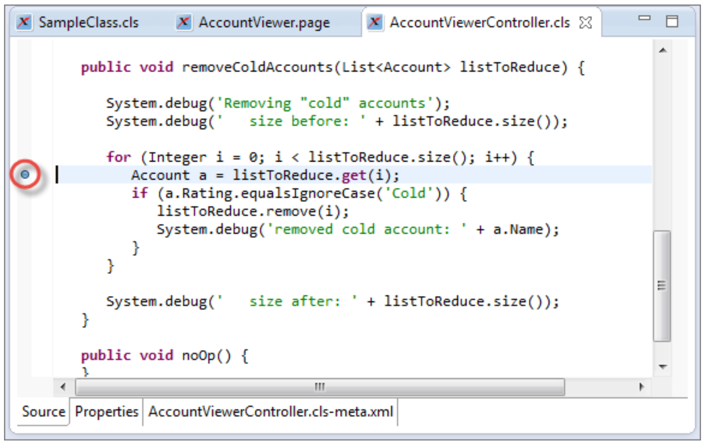
Debug the Problem

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.

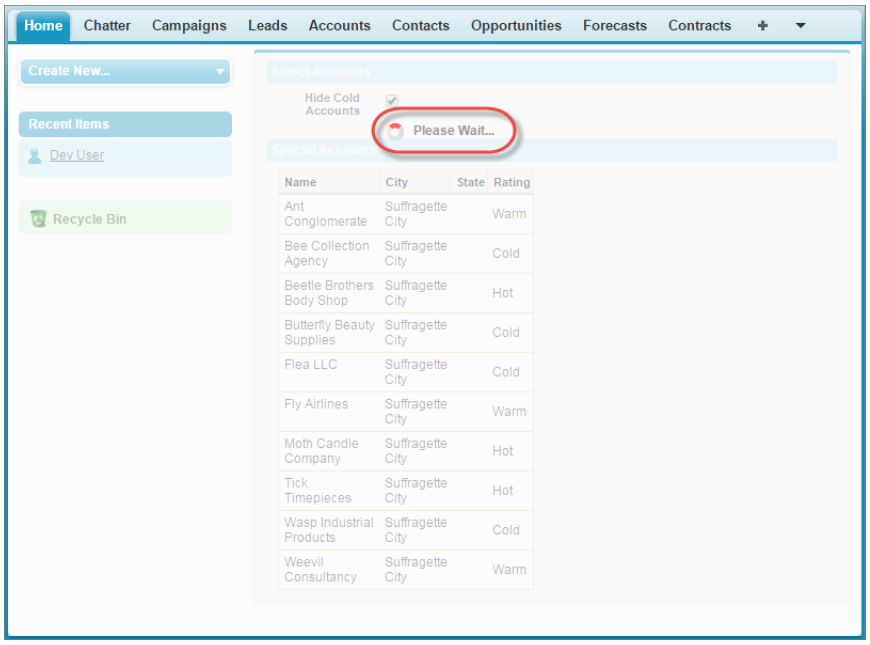
3. Make sure that the turning yellow gears ( ) are visible in the Debug pane. If you don't see the icon that depicts turning yellow gears, click the debug icon ( ) to launch your debug configuration.
4. Find the `removeColdAccounts(List<Account> listToReduce)` method.
5. Set a breakpoint on this line:

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


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

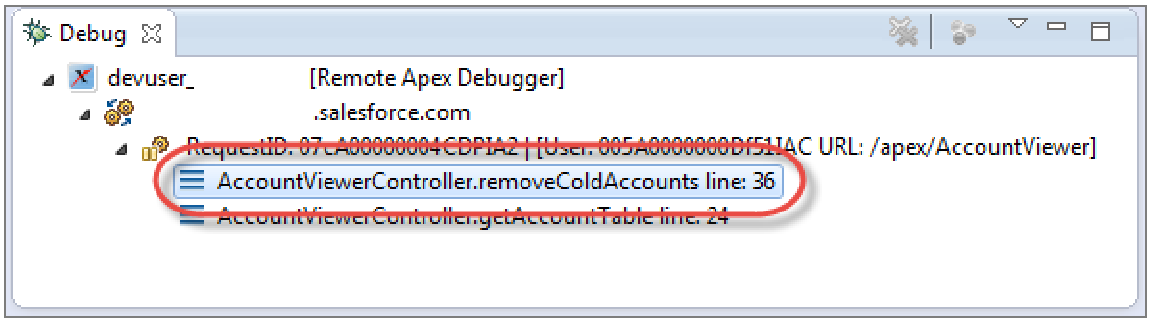
```
public void removeColdAccounts(List<Account> listToReduce) {

   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.



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.



Debug

devuser_ [Remote Apex Debugger]

.salesforce.com

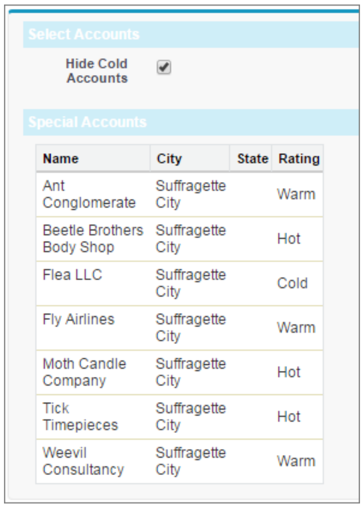
RequestID: 07cA00000004CDPIA2 | [User: 005A0000000Df51IAC URL: /apex/AccountViewer]

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 ✓

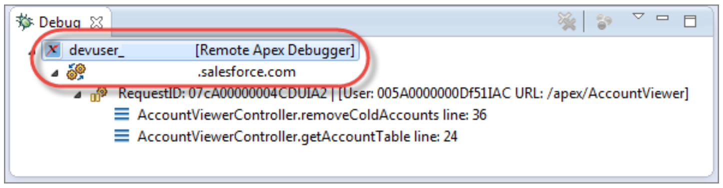
Special Accounts

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.



Debug

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 ( ).

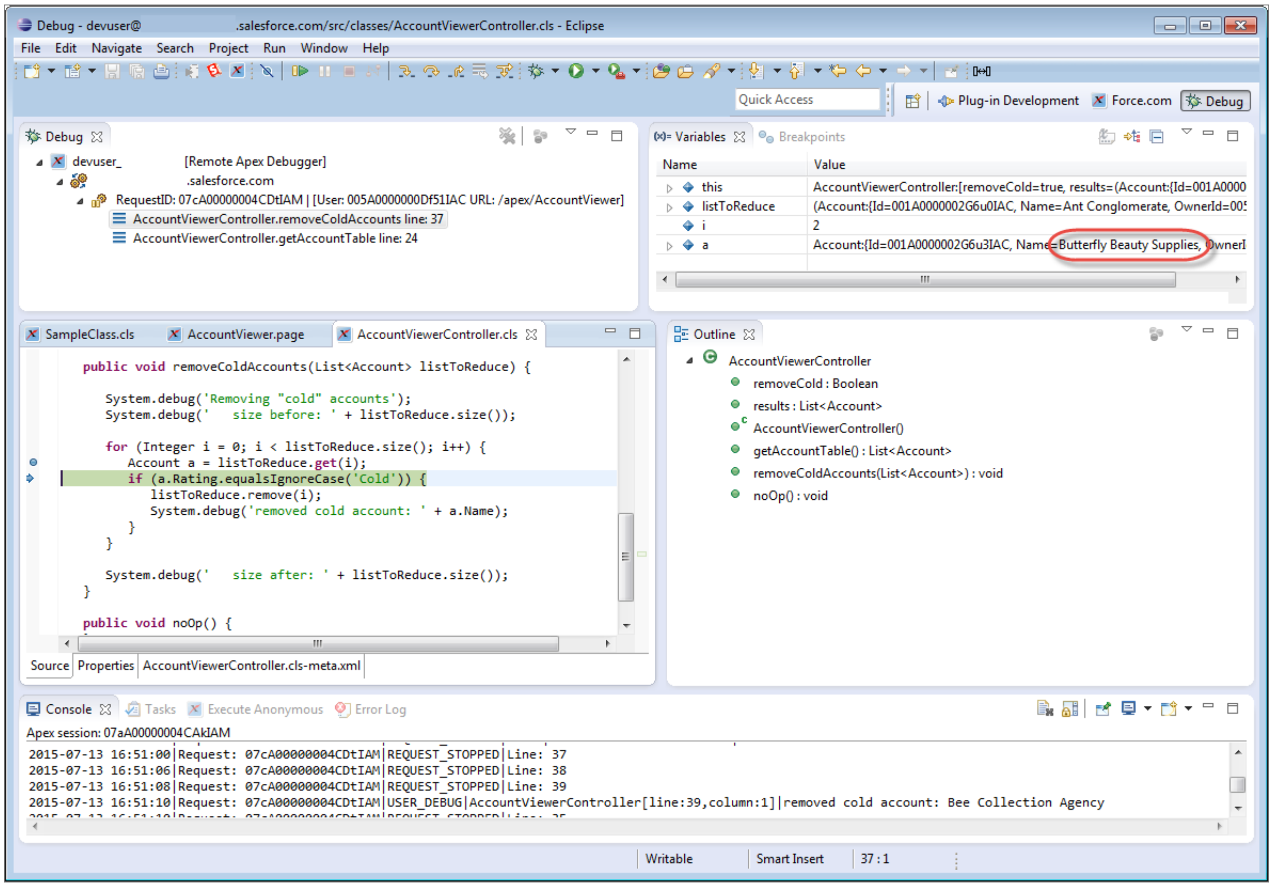
16. To start a new session, click the debug icon ( ).

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

Debug
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=005
i	2
a	Account:{Id=001A0000002G6u3IAC, Name=Butterfly Beauty Supplies, OwnerI

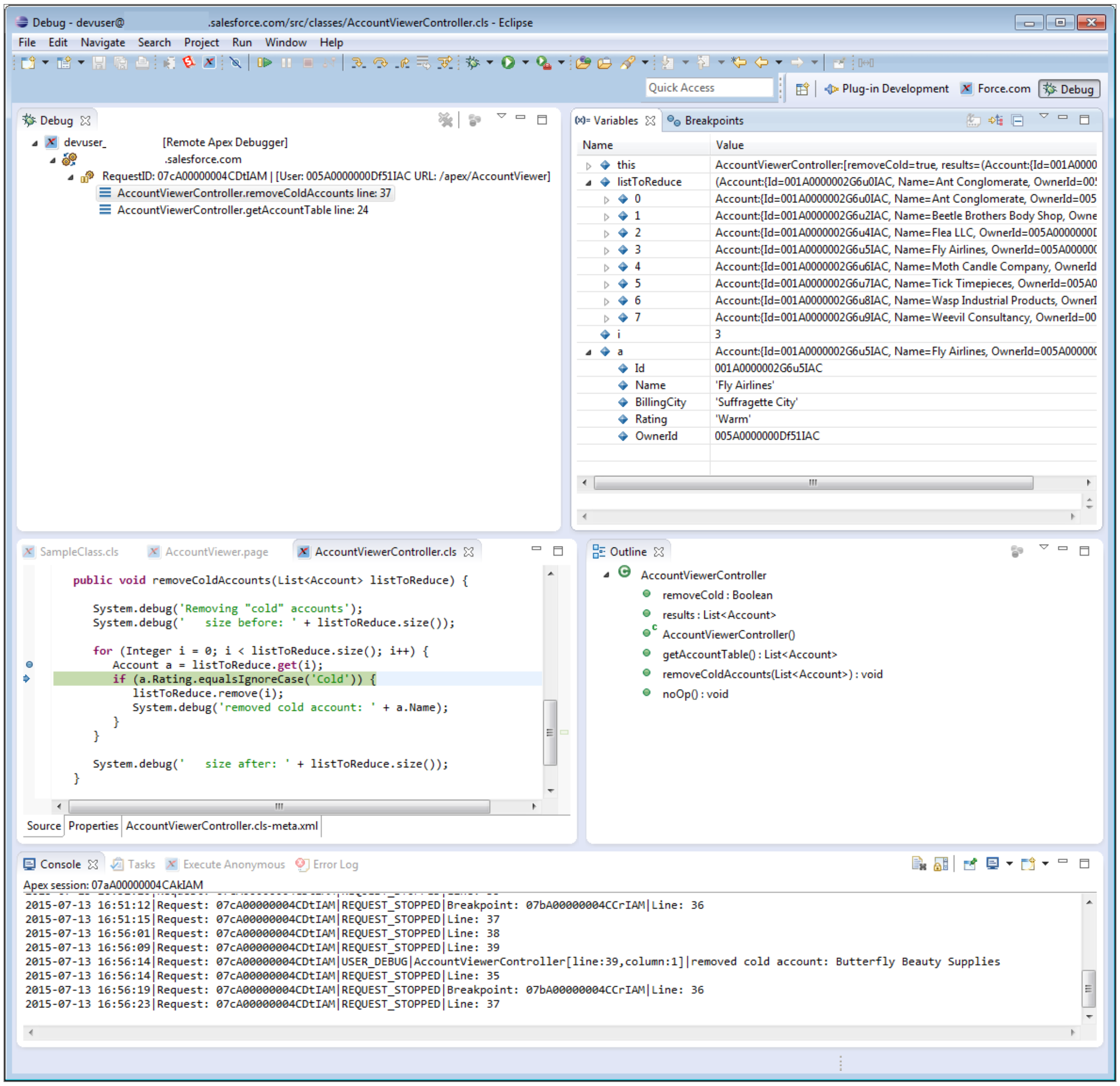
```
    public void removeColdAccounts(List<Account> listToReduce) {

        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<Account>) : void
noOp() : void

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
Variables
Breakpoints
Outline
SampleClass.cls
AccountViewer.page
AccountViewerController.cls
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
```

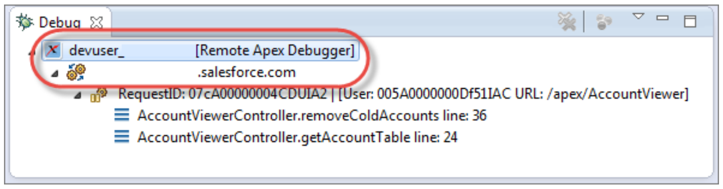
The problem lies in the value of the iterator, `i`. When we removed `'Butterfly Beauty Supplies'` from the list, we didn't account for the fact that removing this list item would change the positions of the subsequent items. Flea LLC is now at position `2`, formerly occupied by Butterfly Beauty Supplies,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

Fix the Problem

By now you've discovered how to fix the problem in `AccountViewerController.cls`. But you can't save changes to code while a debugging session is in process. Terminate the debugging session, and then fix the code.

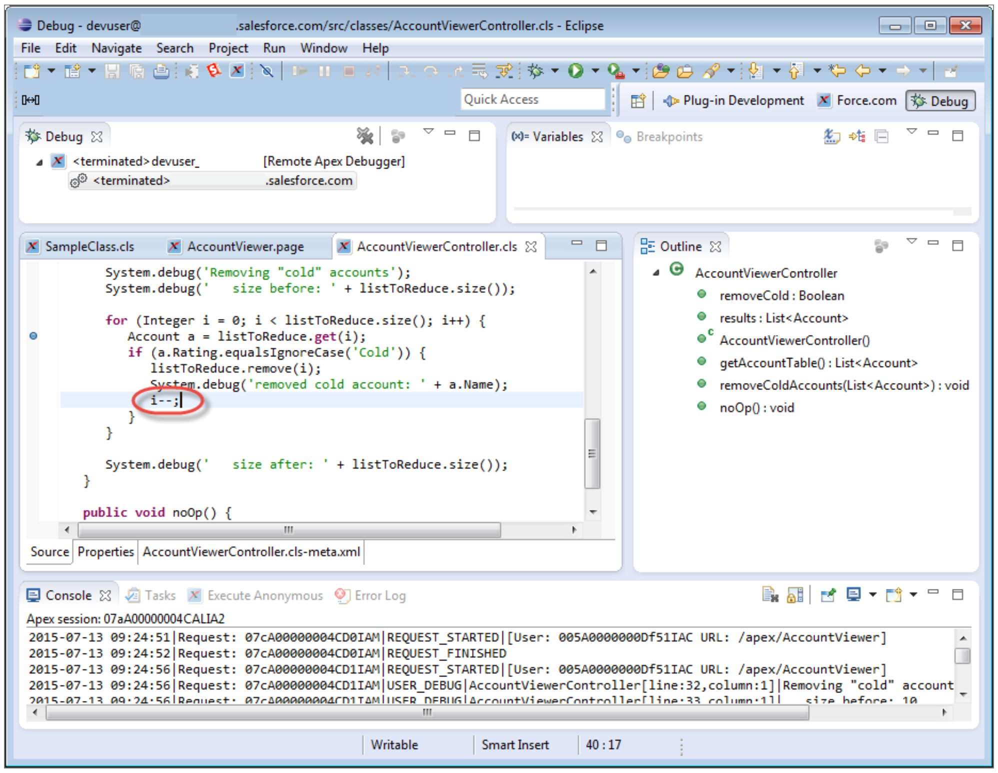
Complete these steps to modify your code to decrement the iterator `i` when removing an item from `listToReduce`.

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.



```
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`.


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
Variables
Breakpoints
<terminated>devuser_ [Remote Apex Debugger]
<terminated> .salesforce.com
SampleClass.cls
AccountViewer.page
AccountViewerController.cls
Outline
AccountViewerController
removeCold : Boolean
results : List<Account>
AccountViewerController()
getAccountTable() : List<Account>
removeColdAccounts(List<Account>) : void
noOp() : void
Source
Properties
AccountViewerController.cls-meta.xml
Console
Tasks
Execute Anonymous
Error Log
Writable
Smart Insert
40 : 17

```
      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. Disable the breakpoint so that you don't accidentally hit it during future debugging sessions. Right-click the breakpoint and select **Disable Breakpoint**, or hold down Shift and double-click the breakpoint. If you prefer to remove the breakpoint, right-click it and select **Toggle Breakpoint**, or double-click it without pressing Shift.
8. 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.

Manage Debugging Sessions

Any of your org's sandboxes can use the Apex Debugger sessions related to your parent org. Track and manage your sandbox orgs' sessions, and subscriber sessions for the ISV Customer Debugger, in Setup.

1. From Setup, enter `Apex Debugger` in the Quick Find box, then select **Apex Debugger**.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Enterprise**, **Performance** (one session included), and **Unlimited** (one session included) Editions

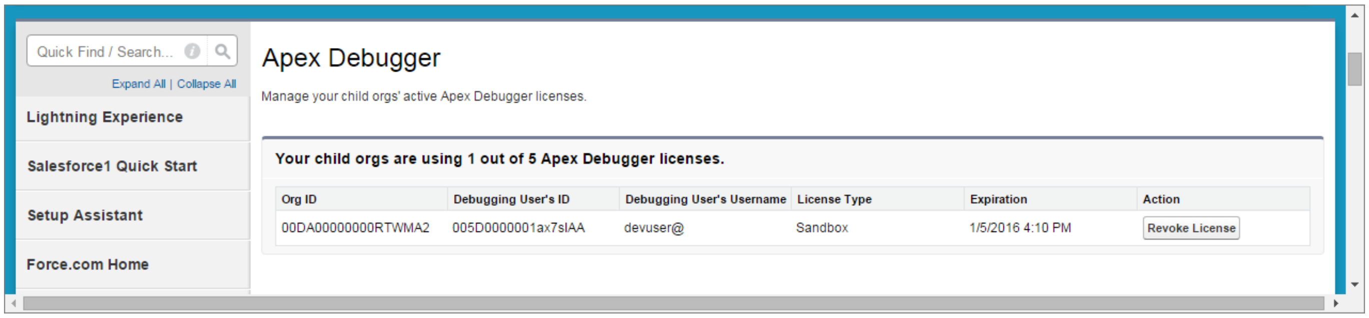
USER PERMISSIONS

To manage Apex Debugger sessions from Setup:

- In your production org: Debug Apex AND auto-enabled dependent permissions

  OR

  In your LMO: Modify All Data AND View and Debug Managed Apex



Apex Debugger

Manage your child orgs' active Apex Debugger licenses.

Your child orgs are using 1 out of 5 Apex Debugger licenses.

Org ID	Debugging User's ID	Debugging User's Username	License Type	Expiration	Action
00DA00000000RTWMA2	005D0000001ax7slAA	devuser@	Sandbox	1/5/2016 4:10 PM	Revoke License

2. If desired, contact the debugging user whose license you plan to revoke.
3. To end the debugging session that you want to terminate, select **Revoke License**.

The debugging session ends, and a license is freed up. The next user who initiates a debugging session consumes the available license. The user whose session you terminated isn't prevented from starting future sessions.

SEE ALSO:

Kill an Orphaned Session

Apex Debugger Limits and Considerations

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

Apex Debugger Limits

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

- You can debug only 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.

SEE ALSO:

Kill an Orphaned Session

Apex Debugger Troubleshooting

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

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.

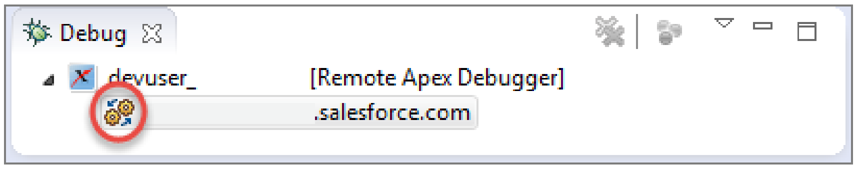
Report Drastic Issues to Customer Support

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.



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.

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**.

USER PERMISSIONS

To manage Apex Debugger sessions from Setup:

- Debug Apex AND auto-enabled dependent permissions

  OR

  Modify All Data AND View and Debug Managed Apex

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.

SEE ALSO:

Salesforce Help: Open the Developer Console

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.

SEE ALSO:

Kill an Orphaned Session

Get Started with the ISV Customer Debugger

The ISV debugging capability covers a gap in what you can do with the Apex Debugger. As an ISV, you can debug your own code. As a subscriber, you can debug your own code. However, because of the protections against seeing managed code, subscribers can't debug ISV code in their orgs.

With the ISV Customer Debugger, an ISV can work with a subscriber to debug issues specific to the subscriber's org. An ISV can reproduce issues in the specific environment, so problems can be diagnosed more quickly, facilitating faster fixes for the ISV's software offering. You can debug only sandbox orgs.

Set Up the Apex Debugger

Before you get started with the ISV Customer Debugger, set up the Force.com IDE and the Apex Debugger.

For information on setting up the Apex Debugger, see Install the Force.com IDE Plug-In.

Set Up a Debugging Session

Setting up an ISV Customer Debugger session is different from setting up sessions for the Apex Debugger. After you create an Eclipse workspace, log in to your customer's org, start a debugging session from Setup, and then debug your project.

USER PERMISSIONS

To view the Apex code in your packages, and debug code with the Apex Debugger, when logged in to subscriber orgs via the LMO:

- View and Debug Managed Apex

To manage ISV Customer Debugger sessions from the LMO:

- Modify All Data

  AND

  View and Debug Managed Apex

Create an Eclipse Workspace

We recommend that you use a different Eclipse workspace for debugging each subscriber org. To create a workspace, select **File** > **Switch Workspace** > **Other** > **Browse** > **New Folder**. Name your workspace, and then select **Create** > **Open** > **OK**.

Log In to a Customer Org

To set up an ISV Customer Debugger session, log in to your subscriber's org. If you're not familiar with this process, see the *ISVforce Guide*.

Start a Debugging Session

Use the simple drag-and-drop interface to initiate an ISV Customer Debugger session from Setup. Then, set up a debug configuration in Eclipse.

Create an Eclipse Workspace

We recommend that you use a different Eclipse workspace for debugging each subscriber org. To create a workspace, select **File** > **Switch Workspace** > **Other** > **Browse** > **New Folder**. Name your workspace, and then select **Create** > **Open** > **OK**.

Log In to a Customer Org

To set up an ISV Customer Debugger session, log in to your subscriber's org. If you're not familiar with this process, see the *ISVforce Guide*.

For information on how to obtain login access to your subscriber's org, see Request Login Access from a Customer. For information on how to log in via the Subscriber Support Console, see Logging In to Subscriber Orgs. You can debug only sandbox orgs.

Start a Debugging Session

Use the simple drag-and-drop interface to initiate an ISV Customer Debugger session from Setup. Then, set up a debug configuration in Eclipse.

Initiate a Debugging Session from Setup

Initiate an ISV Customer Debugger session from your browser while logged in to your subscriber's org.

Download Your Subscriber's Code

By default, we download only the packaged code from your subscriber's org. If you want to download and set breakpoints in your subscriber's code, download their code from the Force.com IDE.

Set Up a Debug Configuration

Before you start your first debugging session for a project, set up a debug configuration.

Initiate a Debugging Session from Setup

Initiate an ISV Customer Debugger session from your browser while logged in to your subscriber's org.

USER PERMISSIONS

To view the Apex code in your packages, and debug code with the Apex Debugger, when logged in to subscriber orgs via the LMO:

- View and Debug Managed Apex

1. Launch the Force.com IDE, if it's not already running.
2. Open the Eclipse workspace for the customer org that you want to debug.
3. If a welcome screen appears, close it.
4. In your browser, make sure that you're logged in to your subscriber's org.
5. From Setup, enter `Apex Debugger` in the `Quick Find` box, then click **Apex Debugger**.
6. Click **Start Partner Debugging Session**.
7. Drag the icon (>_) from your browser to your Eclipse window.

Dragging the icon to the Force.com IDE creates an Eclipse project and downloads all packaged code from your subscriber's org to your workstation. Non-global code from packages that you don't own is obfuscated. For information about code obfuscation, see Protecting Your Intellectual Property in the *ISVforce Guide*.

Stay logged in to the subscriber's org until you're done debugging. The ISV Customer Debugger session terminates when you return to your LMO.

Download Your Subscriber's Code

By default, we download only the packaged code from your subscriber's org. If you want to download and set breakpoints in your subscriber's code, download their code from the Force.com IDE.

1. Launch the Force.com IDE, if it's not already running.
2. In the Package Explorer, in the project that that was created when you initiated the debugging session, right-click **Referenced Packages** and select **Force.com** > **Refresh from Server**.
3. Right-click the project, and select **Force.com** > **Add/Remove Metadata Components**.
4. If you don't see the Force.com Project Contents pane in the properties window, select **Force.com** > **Project Contents**.
5. Click **Add/Remove**.
6. If you get a package manifest content warning describing the component types that the Force.com IDE can't import, click **OK**.
7. In the Choose Metadata Components window, select the components that you want to download. You can set breakpoints only in Apex code.
8. Select **OK** > **Apply**.
9. If you get a message asking whether you want to refresh the project from the server, click **Yes**.

   Eclipse refreshes your project contents from the server. For large codebases, this process can take a few minutes.
10. To close the properties window, click **OK**.

Warning: Keep in mind that your project is connected to your subscriber's org. Changes that you save to the server from Eclipse, or that you make in your browser, are saved to your customer's org.

Set Up a Debug Configuration

Before you start your first debugging session for a project, set up a debug configuration.

1. Switch to the Debug perspective by choosing **Window** > **Open Perspective** > **Other** > **Debug**.

Tip: 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.

2. Click the debug icon ( ) in the toolbar and select **Debug Configurations**.
3. Select **Remote Apex Debugger**.
4. To create a configuration for your project, click the New launch configuration icon ( ).
5. Name the configuration.
6. Click **Browse** and select your project.
7. During an ISV Customer Debugger session, all actions in your subscriber's org are debugged. If you want to debug actions only for specific users or request types, to set up whitelisting, select **Whitelisting Enabled** in your Remote Apex Debugger configuration. For more information, see Whitelist Users and Request Types.
8. Click **Apply**, and then click **Debug**.
9. Click the debug icon ( ) and launch the new debug configuration.

Debug Your Project

If you've used a debugger before, the Force.com IDE's debug perspective probably looks familiar. But we know that many Salesforce developers haven't used traditional debuggers, so we've provided some documentation to help you get started.

For information on using the Apex Debugger, see Get Started with the Apex Debugger.

Renew a Debugging Session

If your ISV Customer Debugger session expires, don't panic—just start a new one. To renew a session, drag the icon from Setup to Eclipse, just like you did to start your first session.

For details on how to start a session, see Initiate a Debugging Session from Setup.

Delete Your Copy of Your Customer's Code

The code from your subscriber's org is your subscriber's intellectual property. We advise against keeping it around after you're done debugging.

When you close the Force.com IDE, we prompt you to delete your subscriber's code. If you choose to not delete the code then, be sure to delete it later.

Close files before you delete them. Otherwise, you get an error message.

ISV Customer Debugger Limits and Considerations

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

- You can debug only one customer at a time. However, if you purchase Apex Debugger licenses, you can debug multiple customers at once. Apex Debugger licenses also let you debug in your sandboxes and scratch orgs.
- When you click **Return to subscriber overview**, your debugging session terminates. Stay logged in to your subscriber's org while you debug, and return to your LMO only when you're done debugging.

- You can debug only sandbox orgs.

CHAPTER 6 Solutions to Common Force.com IDE Problems

If you encounter problems when installing or using the Force.com IDE, try these troubleshooting techniques.

After Upgrade, Passwords Are No Longer Saved

Problem

After you upgrade the Force.com IDE, Eclipse no longer saves your passwords.

Solution

1. Select **Eclipse** > **Preferences** > **General** > **Security** > **Secure Storage** > **Contents** > **Force Projects** > **Delete** > **Yes** > **Yes**.
2. After Eclipse restarts, right-click your project in the Package Explorer and select **Force.com** > **Properties**.
3. Enter your password, and then click **Apply**.

   Note: When you create a project, you are sometimes prompted for a new master password. This master password is a separate password of your choosing. It's required by Eclipse secure storage but not associated with your Force.com IDE credentials. For information on Eclipse secure storage, see the *Eclipse Workbench User Guide*.

Tip: If these steps don't solve your problem, try manually deleting the file that stores your passwords. Look for this file in a location such as `/Users/`*`yourName`*`/.eclipse/org.eclipse.equinox.security/secure_storage` or `D:\Users\Administrator\.eclipse\org.eclipse.equinox.security\secure_storage`.

New Menu Options or Views Don't Appear After Upgrade

Problem

After you upgrade the Force.com IDE, you don't see the new menu options or views that you're expecting.

Solution

Create a new workspace.

1. Select **File** > **Switch Workspace** > **Other** > **Browse**.
2. Navigate up a directory, and then click **New Folder**.
3. Enter a name for your new workspace, and then select **Create** > **Open** > **OK**.
4. Switch to the Force.com perspective. Select **Window** > **Perspective** > **Open Perspective** > **Other** > **Force.com** > **OK**.

No Repository Found

Problem

Installation fails with the message, "No repository found at `http://media.developerforce.com/force-ide/eclipse42`. Error reading update site `http://media.developerforce.com/force-ide/eclipse42`."

Solution

Configure Eclipse to use your proxy server. You can find these settings in your Eclipse Preferences under **General** > **Network Connections**.

No Force.com Perspective (macOS)

Problem

On macOS, the Force.com perspective does not appear in Eclipse, despite having the current version of the Java Runtime Environment.

Solution

Download the full version of the Java SE Development Kit (JDK) from the Java download page.

No Force.com Perspective (Windows)

Problem

Installation succeeds on Windows, but the Force.com perspective does not appear under **Window** > **Perspective** > **Open Perspective** > **Other**.

Solution

Add the explicit path to the JRE in the `eclipse.ini` file located in the directory where you installed Eclipse. Add this line:

```
-vm <full path to your jre>\javaw.exe
```

Conflicting Dependency

Problem

Attempting to install the Force.com IDE generates the following error.

```
Cannot complete the install because of a conflicting dependency.
  Software being installed: Force.com IDE...
```

Solution

Move Eclipse to a folder other than the Program Files folder. See this thread for more information.

Other Installation Problems

Didn’t find the answer to your problem? Get help from a product expert on the discussion forums.

CHAPTER 7 Force.com IDE Release Notes

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

Winter '17 (Force.com IDE v38.0)

Winter '17 (Force.com IDE v38.0) contains the following updates.

Create and Edit Apex Test Suites

You can now create and edit suites of Apex tests in the Force.com IDE.

To create a test suite, select **File** > **New** > **Apex Test Suite**, or right-click your project in the Package Explorer and then select **New** > **Apex Test Suite**.

To edit a test suite, edit its XML file: `/src/testSuites/YourTestSuite.testSuite`. Apex test suites are represented in Metadata API as `ApexTestSuite` components.

Refresh Standard Objects Metadata from the Server

We've resolved the bug that you pointed out in GitHub Issue #203. You can once again refresh your standard objects from the Salesforce server.

Summer '16 (Force.com IDE v37.0)

Summer '16 (Force.com IDE v37.0) contains the following updates.

Increase Security with Java 8 and Eclipse 4.5 (Mars)

We've updated the Force.com IDE to require Java 8 and Eclipse 4.5 (Mars) or later. Most of you have switched to these technologies. Focusing on them enables us to optimize your experience with our tools. As we announced in November 2015, Salesforce is disabling the TLS 1.0 encryption protocol to increase security. TLS 1.0 is disabled by default in Java 8, which also includes other security fixes.

Install the Summer '16 (Force.com IDE v37.0) Plug-In from a New Update Site

To install the latest version of the plug-in:

1. Launch Eclipse and select **Help** > **Install New Software**.
2. Click **Add**.
3. In the Add Repository dialog, set the name to *`Force.com IDE`* and the location to *`https://developer.salesforce.com/media/force-ide/eclipse45`*. For Spring '16 (Force.com IDE v36.0) and earlier Force.com IDE versions, use *`https://developer.salesforce.com/media/force-ide/eclipse42`*.

For full installation instructions, see Install the Force.com IDE Plug-In.

Develop Lightning Components in the Force.com IDE

We've added support for Lightning components to the Force.com IDE. This release includes the following features. For more information about these features, see Notable Lightning Components Features in the Force.com IDE.

Multi-Page Editor

Easily access and edit all the resources in your Lightning component bundle. The multi-page editor opens by default when you open a component bundle.

Documentation Browser

Access documentation for built-in tags and functions without leaving the IDE. To open the Lightning Doc view, select **Window** > **Show View** > **Other** > **Other** > **Lightning Doc**.

Navigation

Outline View

Quickly locate and navigate between the elements in your Lightning component bundle. To open Outline View, press Ctrl+O (Cmd+O on macOS).

Open Dialog

Quickly locate and navigate between your Lightning component bundles. To display the Open Dialog, press Ctrl+Shift+A (Cmd+Shift+A on macOS).

Auto-Completion

Save time and avoid typos with the auto-completion feature for markup, JavaScript, and CSS files. Some auto-completion is provided by default as you type. To invoke more auto-completion options, press Ctrl+Space when you're editing a file.

Easily Open Apex Classes and Triggers with Open Type

We've enabled Eclipse's Open Type functionality for Apex types. You can now open a dialog to search your Apex classes and triggers by pressing Ctrl+Shift+T (Cmd+Shift+T on macOS) or selecting **Navigate** > **Open Apex Type**.

This functionality is available only in the Force.com perspective. For more information, see Open Type in the Eclipse documentation.

Set Breakpoints on Exceptions with the Apex Debugger

You can now set breakpoints on `System` and custom exceptions. When an exception on which you've set a breakpoint is thrown during a debugging session, execution halts. The Apex Debugger pauses on the line of code that caused the thrown exception.

To set an exception breakpoint, from the Breakpoints pane in the Debug perspective, click **Add Apex Breakpoint**. A list of all exceptions in your workspace displays. Select an exception, confirm that the exception's listed project is the project you're working in, and then click **Set Breakpoint**.

Automatically Switch to the Debug Perspective When You Hit a Breakpoint

When you hit a breakpoint in the Force.com perspective during a debugging session, the Force.com IDE now switches to the Debug perspective.

Update Test Method Names Without Updating Full-Class Test Run Configurations

You can now change the names of test methods that are included in full-class test run configurations and then execute test runs without errors. Previously, if you changed a test method's name after setting up a test run configuration that included the method's class, the Force.com IDE could execute the run configuration only after you updated the configuration. You still need to update run configurations that contain individual test methods when you change the methods' names. This fix is only for run configurations that contain full classes, including run configurations that execute all tests in your org.

Spring '16 (Force.com IDE v36.0)

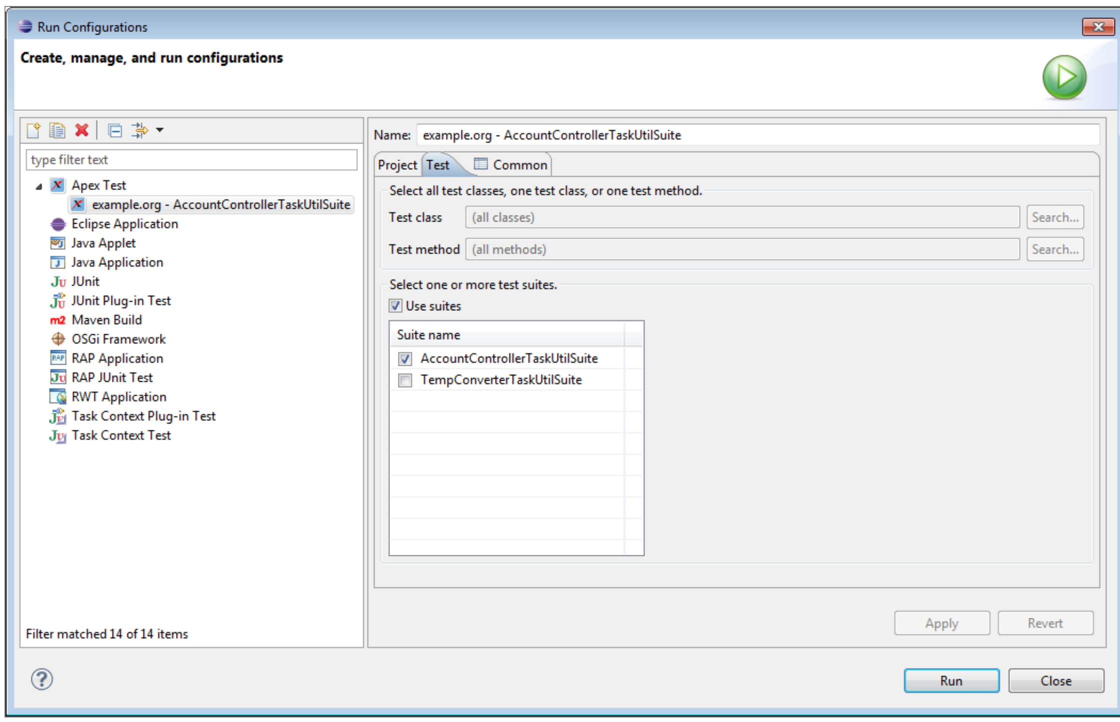
Spring '16 (Force.com IDE v36.0) contains the following updates.

January 22, 2016

Execute Apex Test Suites

You can now execute Apex test suites in the Force.com IDE. A test suite is a collection of Apex test classes that you run together. For example, create a suite of tests that you run every time you prepare for a deployment or Salesforce releases a new version.

To execute test suites, select **Run** > **Run Configurations** > **Apex Test** > **New launch configuration** ( ). Select the **Test** tab. To create a run of test suites, select **Use suites**, and then select one or more test suites. To execute the selected test run configuration, click **Run**.



Name: example.org - AccountControllerTaskUtilSuite

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

Test class (all classes)

Test method (all methods)

Select one or more test suites.

[✓] Use suites

Suite name
[✓] AccountControllerTaskUtilSuite
[ ] TempConverterTaskUtilSuite

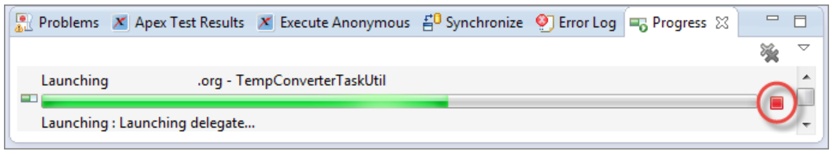
To create or customize a test suite, use the Developer Console or the API. See Salesforce Help: Manage Sets of Apex Test Classes with Test Suites or *Object Reference for Salesforce and Lightning Platform*: TestSuiteMembership.

Enjoy Optimized Test Execution

Synchronous Apex test runs can include only one class. Because synchronous test runs are faster than asynchronous runs, all Force.com IDE test runs that include methods from only one Apex class are now executed synchronously. Test runs that include multiple classes are executed asynchronously as they always have been. Your test run configurations no longer include an option to configure synchronous or asynchronous execution.

See Apex Test Results More Quickly

Test results now load in the Apex Test Results view as your test methods finish executing. Previously, test results displayed only after all tests in the run had executed. If your test run's results aren't looking promising, you can stop your test run using the Progress view's terminate icon.



```
Launching                .org - TempConverterTaskUtil
Launching : Launching delegate...
```

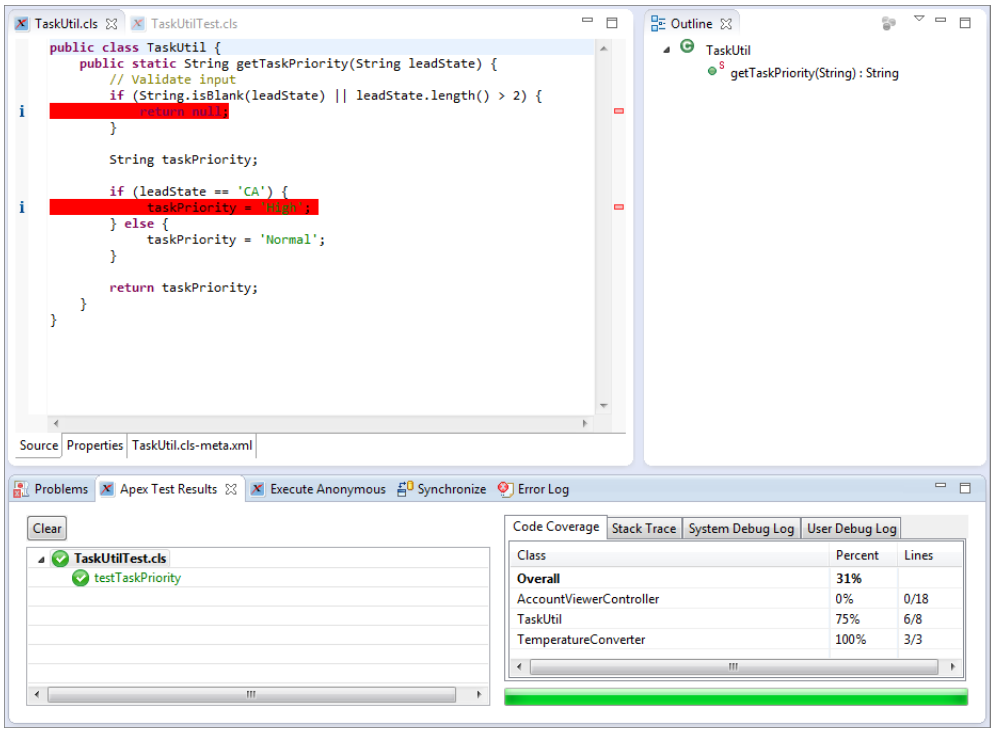
Note: If you expand test results in the Apex Test Results view before your test run finishes, the results collapse again when more methods finish executing.

January 8, 2016

Enabled code-coverage highlighting in Apex classes

To update your code coverage, execute Apex tests by selecting **Run** > **Run Configurations** > **Apex Test**. You can then view code coverage for your classes in the Apex Test Results pane.

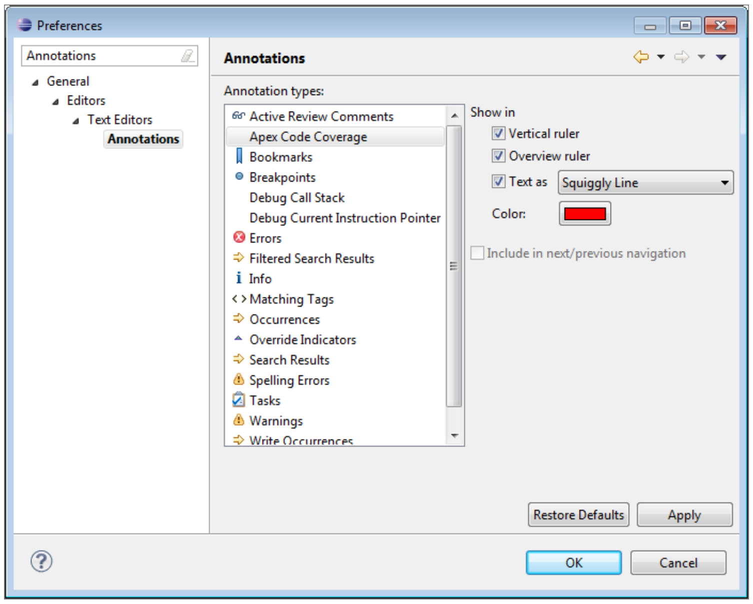
To view code coverage highlighting, double-click a class name in the Apex Test Results pane's Code Coverage list. Lines that aren't covered by your unit tests are highlighted. Unlike the Developer Console, which color-codes both covered and uncovered lines, the Force.com IDE highlights only the lines of code that still need coverage.



```
public class TaskUtil {
    public static String getTaskPriority(String leadState) {
        // Validate input
        if (String.isBlank(leadState) || leadState.length() > 2) {
            return null;
        }

        String taskPriority;

        if (leadState == 'CA') {
             taskPriority = 'High';
        } else {
             taskPriority = 'Normal';
        }

        return taskPriority;
    }
}
```

Class	Percent	Lines
Overall	**31%**	
AccountViewerController	0%	0/18
TaskUtil	75%	6/8
TemperatureConverter	100%	3/3

To change the color for your uncovered code or to replace the highlighting with squiggly underlining, select **Window** > **Preferences**. Enter `Annotations` in the filter text box, then select **Annotations** > **Apex Code Coverage**. Make your desired changes, and then click **Apply**.


Preferences

Annotations

- General
  - Editors
    - Text Editors
      - **Annotations**

Annotations

Annotation types:

- Active Review Comments
- Apex Code Coverage
- Bookmarks
- Breakpoints
- Debug Call Stack
- Debug Current Instruction Pointer
- Errors
- Filtered Search Results
- Info
- Matching Tags
- Occurrences
- Override Indicators
- Search Results
- Spelling Errors
- Tasks
- Warnings
- Write Occurrences

Show in

- [✓] Vertical ruler
- [✓] Overview ruler
- [✓] Text as: Squiggly Line

Color: 🟥

- [ ] Include in next/previous navigation

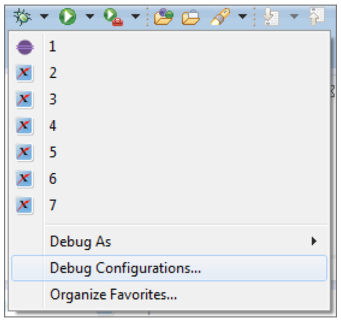
Restore Defaults | Apply

OK | Cancel

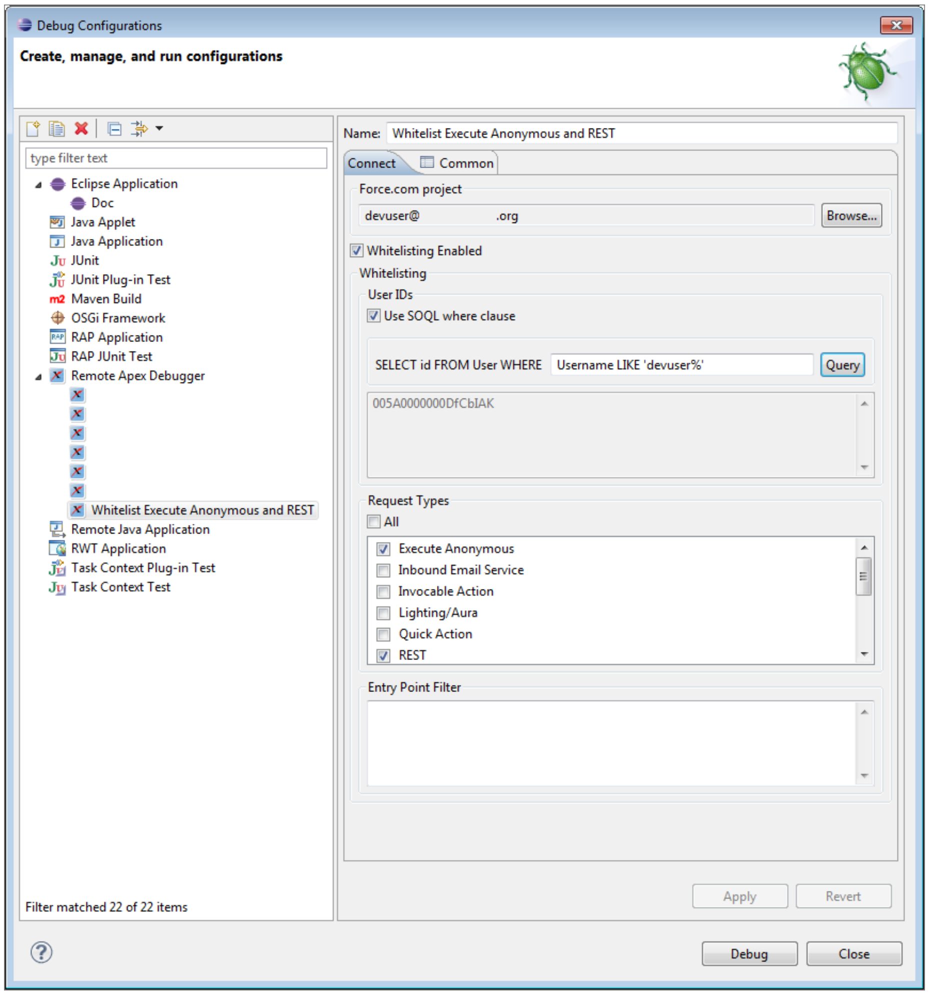
Provided whitelist filtering for user and request-type debugging in the Apex Debugger

You can now filter which requests are debugged by setting up whitelisting. Previously, all events in your org triggered debugging during a debugging session. Now you can focus only on the events that are relevant to the problem you're debugging.

To access your debug configurations, select the arrow next to the debug icon ( ), and then select **Debug Configurations**.



To set up whitelisting, select **Whitelisting Enabled** in your Remote Apex Debugger configuration, and then filter by user ID or request type. To filter by user ID, enter a SOQL condition expression in the `SELECT id FROM User WHERE` field. Or enter a comma-separated list of user IDs in the text field in the User IDs section. To filter by request type, select one or more entry points, or enter a regular expression in the `Entry Point Filter` field. For example, to whitelist requests made by the Visualforce page `myPage`, enter `.*/apex/myPage.apexp`.



Debug Configurations

Create, manage, and run configurations

type filter text

- Eclipse Application
  - Doc
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- Maven Build
- OSGi Framework
- RAP Application
- RAP JUnit Test
- Remote Apex Debugger
  - Whitelist Execute Anonymous and REST
- Remote Java Application
- RWT Application
- Task Context Plug-in Test
- Task Context Test

Filter matched 22 of 22 items

Name: Whitelist Execute Anonymous and REST

Connect | Common

Force.com project: devuser@ .org

[✓] Whitelisting Enabled

Whitelisting

User IDs

[✓] Use SOQL where clause

SELECT id FROM User WHERE Username LIKE 'devuser%'

005A0000000DfCbIAK

Request Types

[ ] All

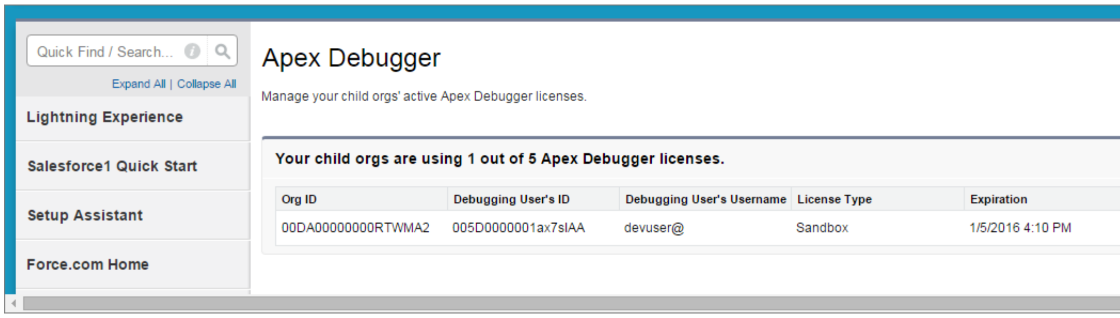
[✓] Execute Anonymous
[ ] Inbound Email Service
[ ] Invocable Action
[ ] Lighting/Aura
[ ] Quick Action
[✓] REST

Entry Point Filter

Typically, modifying an Eclipse debug configuration during a debugging session doesn't affect active debug targets. However, whitelist modifications affect both current and future sessions.

Share Apex Debugger licenses among your sandbox orgs

You can now purchase Apex Debugger licenses for your parent org and share them among the users in your org's sandboxes. In your parent org, view active sessions and revoke licenses on the Apex Debugger page in Setup. Revoking a license doesn't prevent your user from initiating future debugging sessions.



Apex Debugger

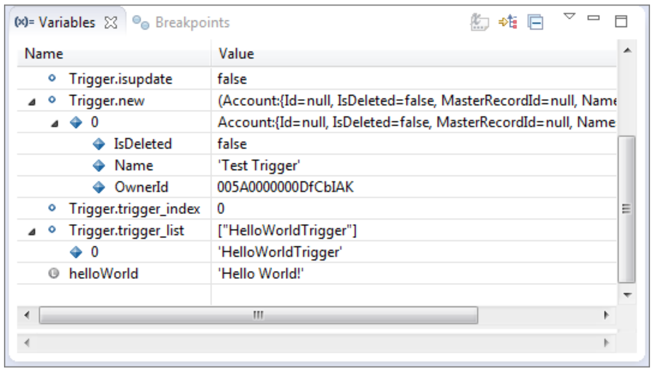
Manage your child orgs' active Apex Debugger licenses.

Your child orgs are using 1 out of 5 Apex Debugger licenses.

Org ID	Debugging User's ID	Debugging User's Username	License Type	Expiration
00DA00000000RTWMA2	005D0000001ax7sIAA	devuser@	Sandbox	1/5/2016 4:10 PM

Enabled Eclipse's Java development tools (JDT) icons for Apex variables in the Apex Debugger

JDT icons now display in the Apex Debugger's Variables pane to denote access and definition modifiers for Apex variables. You can determine at a glance whether a variable is static, local, global, and so on. Previously, JDT icons displayed in the Outline pane, but all variables in the Variables pane had the same icon.



Name	Value
Trigger.isupdate	false
Trigger.new	(Account:{Id=null, IsDeleted=false, MasterRecordId=null, Name
0	Account:{Id=null, IsDeleted=false, MasterRecordId=null, Name
IsDeleted	false
Name	'Test Trigger'
OwnerId	005A0000000DfCbIAK
Trigger.trigger_index	0
Trigger.trigger_list	["HelloWorldTrigger"]
0	'HelloWorldTrigger'
helloWorld	'Hello World!'

For a list of JDT icons, see JDT Icons in Eclipse's *Java development user guide*.

Winter '16 (Force.com IDE v35.0)

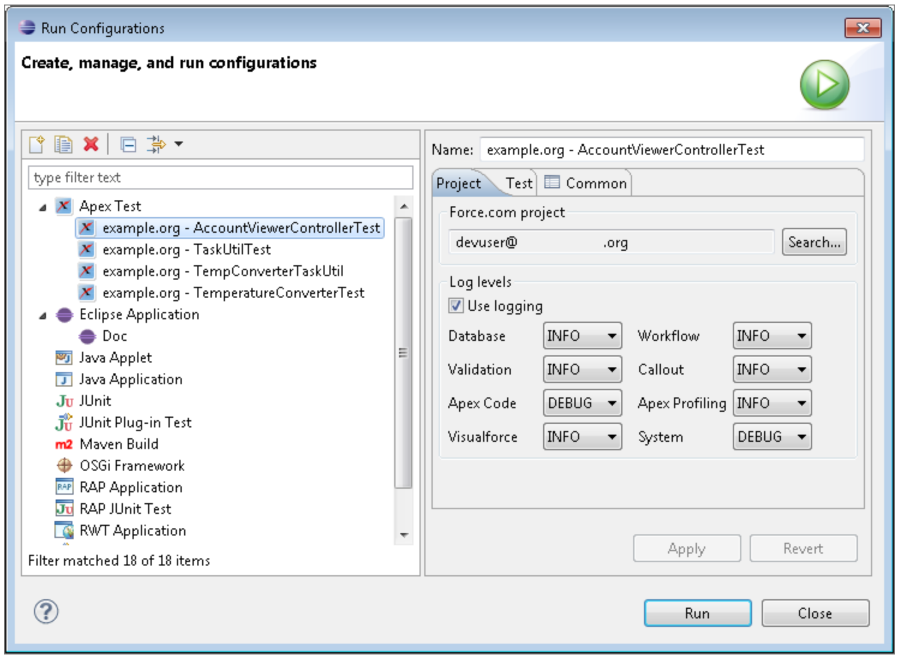
Winter '16 (Force.com IDE v35.0) contains the following updates.

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: example.org - AccountViewerControllerTest

Force.com project: devuser@ .org

Log levels

[✓] Use logging

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

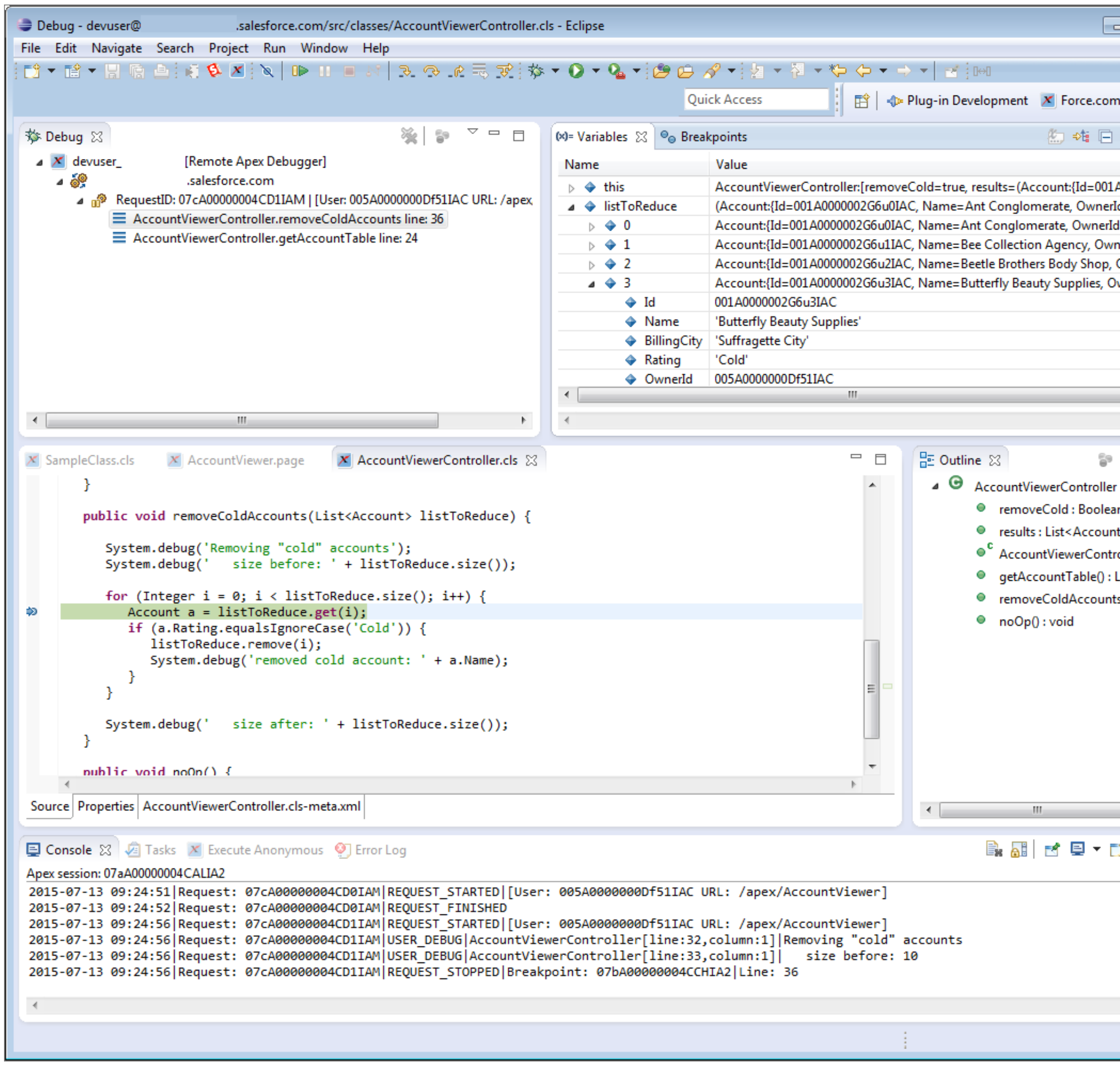
Made the interactive Apex Debugger generally available

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.



```
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=001A
listToReduce	(Account:{Id=001A0000002G6u0IAC, Name=Ant Conglomerate, OwnerI
0	Account:{Id=001A0000002G6u0IAC, Name=Ant Conglomerate, OwnerId
1	Account:{Id=001A0000002G6u1IAC, Name=Bee Collection Agency, Own
2	Account:{Id=001A0000002G6u2IAC, Name=Beetle Brothers Body Shop, C
3	Account:{Id=001A0000002G6u3IAC, Name=Butterfly Beauty Supplies, O
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() {
```

```
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.

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.

CHAPTER 8 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
- Troubleshoot the Force.com IDE: Debug Mode

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 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 Restrict Where and When Users Can Log In to Salesforce.
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.

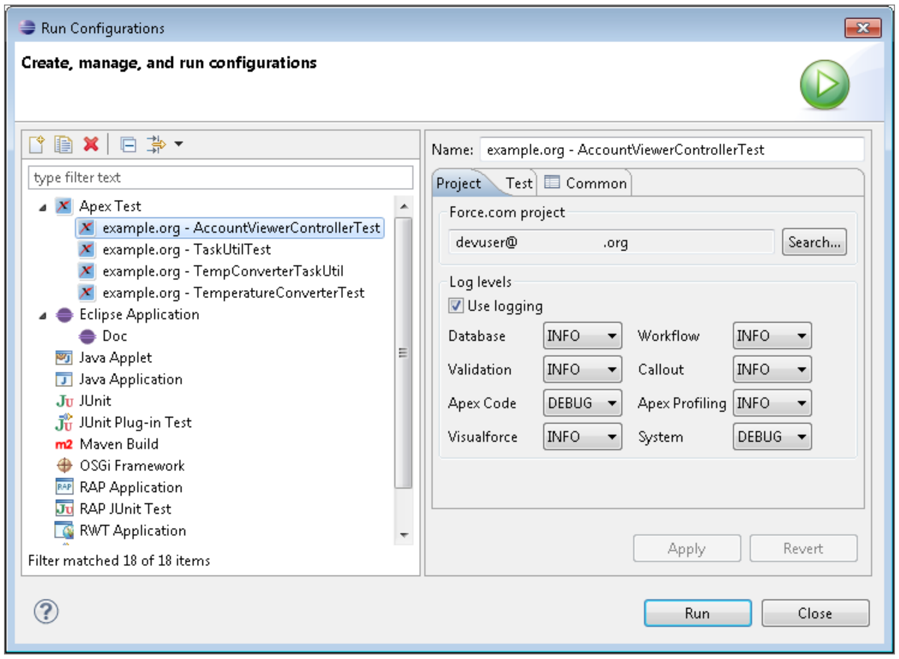
Field	Description
	• Other (Specify) - Choose this option if you want to connect to a specific instance.
Hostname	This field is only available if you have chosen Other (Specify), in the Environment field above. Enter a specific server instance.
Do not change endpoint	Choose this option if you want to 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 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:

1. Select **Run** > **Run Configurations** > **Apex Test**.
2. To create a test run configuration, select **New launch configuration** ( ).
3. To customize your log levels, select the Project tab, and then select **Use logging**.


Run Configurations

Create, manage, and run configurations

Name: example.org - AccountViewerControllerTest

Force.com project: devuser@ .org

Log levels

☑ Use logging

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

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 and the actions taken.
`Validation`	Includes information about validation rules, such as the name of the rule and whether the rule evaluated true or false.
`Callout`	Includes the request-response XML that the server is sending and receiving from an external web service. Useful when debugging issues related to using Lightning Platform web service API calls or troubleshooting user access to external objects via Salesforce Connect.
`Apex Code`	Includes information about Apex code. Can include information such as log messages generated by DML statements, inline SOQL or SOSL queries, the start and completion of any triggers, and the start and completion of any test method.
`Apex Profiling`	Includes cumulative profiling information, such as the limits for your namespace and the number of emails sent.
`Visualforce`	Includes information about Visualforce events, including serialization and deserialization of the view state or the evaluation of a formula field in a Visualforce page.
`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.

About Metadata Files

Metadata Types

SEE ALSO:

About Metadata Files

Metadata Types

Add or Remove Metadata Components

Project Properties

About Metadata Files

The Lightning 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 Salesforce 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.

Troubleshoot 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. Debug Mode causes 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, using the 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 Lightning Platform technical library is available online at `developer.salesforce.com/docs`.

Among the many documents you'll find useful are:

Force.com IDE Developer Guide (This is the same documentation that is packaged with the Force.com IDE plug-in.)

Developer Guides for the Lightning Platform Programming Languages and Controllers:

- *Apex Developer Guide*
- *Visualforce Developer Guide*
- *Lightning Components Developer Guide*

API Developer Guides:

- *SOAP API Developer Guide*
- *REST API Developer Guide*
- *Metadata API Developer Guide*
- *Streaming API Developer Guide*
- *Chatter REST API Developer Guide*

Reference Materials:

- *Object Reference for Salesforce*
- *SOQL and SOSL Reference*
- *Development Lifecycle Guide*
- *AppExchange Publishing Guide*
- *Migration Tool Guide*
- *Salesforce Sites Implementation Guide*
- *Security Implementation Guide*

INDEX