force.com
Dream. Build. Succeed.

Force.com Workbook

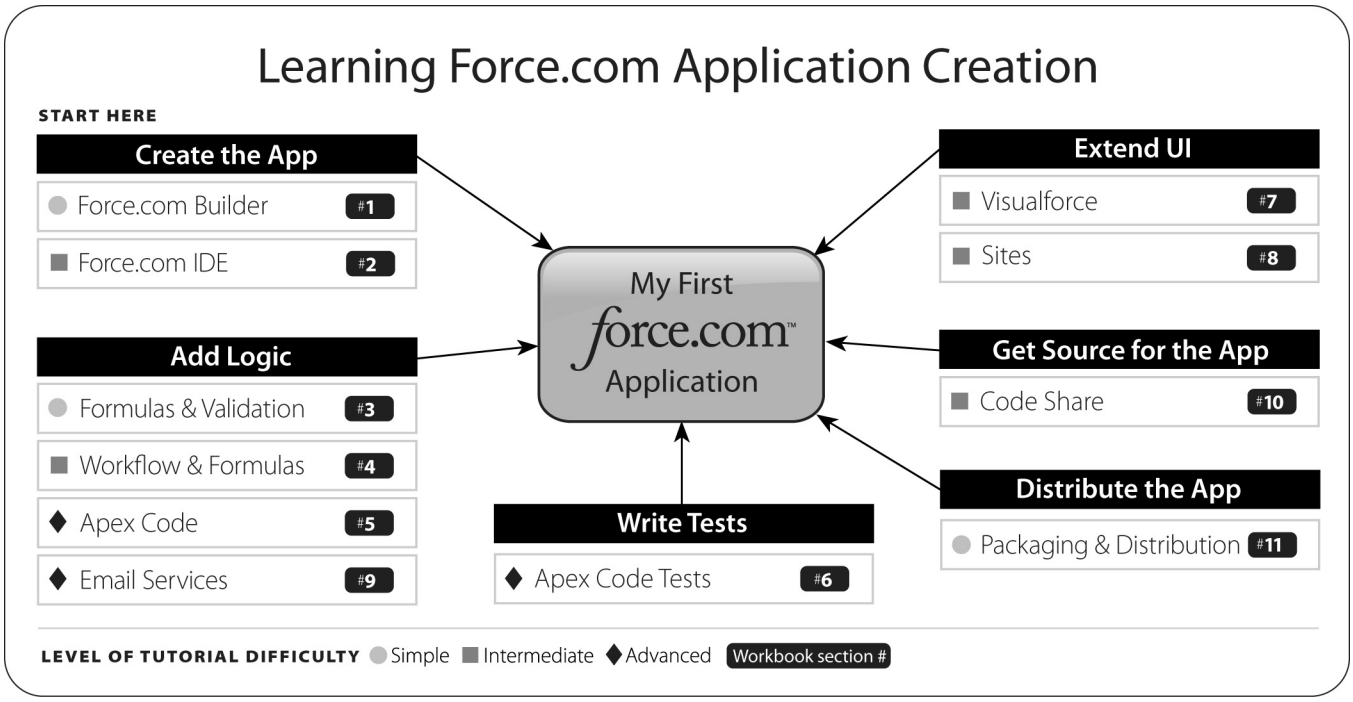
Get Started Building Your First App in the Cloud

Quick 30 min Tutorials

Table of Contents

About the Force.com Tutorials

The Force.com Tutorials show you how to use the Force.com platform to build applications in the cloud. Each of the eleven tutorials included in this workbook uses a simple application named Mileage Tracker to showcase a particular feature of the platform.



To use the tutorials in this workbook, you must first create or download your own version of the Mileage Tracker app:

- If you are an existing Salesforce administrator or are new to building applications on the Force.com platform, create the Mileage Tracker application by following the instructions in Tutorial #1: Creating an App with Point-and-Click Tools on page 3.
- If you have a programming background or are already familiar with the Force.com platform, create the Mileage Tracker applications by following the instructions in Tutorial #2: Creating an App with the Force.com IDE on page 9.
- If you want to download the completed Mileage Tracker application, which includes the completed steps in Tutorial #5: Adding Business Logic with Apex on page 29, follow the instructions in Tutorial #10: Downloading and Deploying an App Using Code Share on page 57.

The following table outlines the tutorials that are included in this workbook.

Tutorial	Difficulty	Estimated Time to Complete
Create the App		
Tutorial #1: Creating an App with Point-and-Click Tools	Beginner	20-30 minutes
Tutorial #2: Creating an App with the Force.com IDE	Intermediate	30 minutes
Add Logic		
Tutorial #3: Using Formulas and Validation	Beginner	20-30 minutes
Tutorial #4: Using Workflow and Approvals	Intermediate	30 minutes
Tutorial #5: Adding Business Logic with Apex	Advanced	30 minutes
Tutorial #9: Adding Email Services	Advanced	20-30 minutes
Extend the User Interface		
Tutorial #7: Building a Custom User Interface Using Visualforce	Intermediate	30 minutes
Tutorial #8: Create a Public Web Page Using Force.com Sites	Intermediate	30 minutes
Get Source for the App		
Tutorial #10: Downloading and Deploying an App Using Code Share	Intermediate	20-30 minutes
Write Tests		
Tutorial #6: Adding Tests to Your Application	Advanced	45-60 minutes
Distribute the App		
Tutorial #11: Packaging and Distributing an Application	Beginner	15-20 Minutes

To Learn More

To learn more about Force.com and to access a rich set of resources, go to http://developer.force.com.

The latest version of this Workbook can be found at http://developer.force.com/workbook.

Tutorial #1: Creating an App with Point-and-Click Tools

Level: Beginner; **Duration:** 20-30 minutes

In this tutorial you will build an application to track the miles a user travels to visit customers. The application will have a data object that stores information, and it will include screens to list, view and edit this information. You will create the entire application very quickly and easily using the point-and-click tools.

Step 1: Create a Custom Object

The first step in creating an application is to create an object that will hold the data—in this case, the miles a user drives on a customer visit. On the Force.com platform, an object is similar to a database table. Your Salesforce applications already contain many such prebuilt objects—known as standard objects—for all the data those applications need. However, you can also build objects to hold data that is relevant to your custom application. Such objects are called custom objects.

1. Log into your Developer Edition organization.

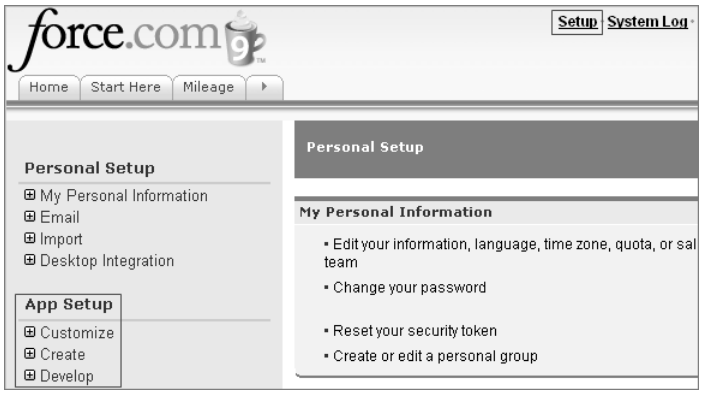
   

   Note: If you don't already have a Developer Edition organization, go to `http://developer.force.com/join` and follow the instructions to sign up for a Developer Edition account. A Developer Edition account gives you access to a free Developer Edition organization.

   a. Launch your browser and go to `https://login.salesforce.com`.
   b. Enter your Developer Edition username (in the form of an email address) and password.

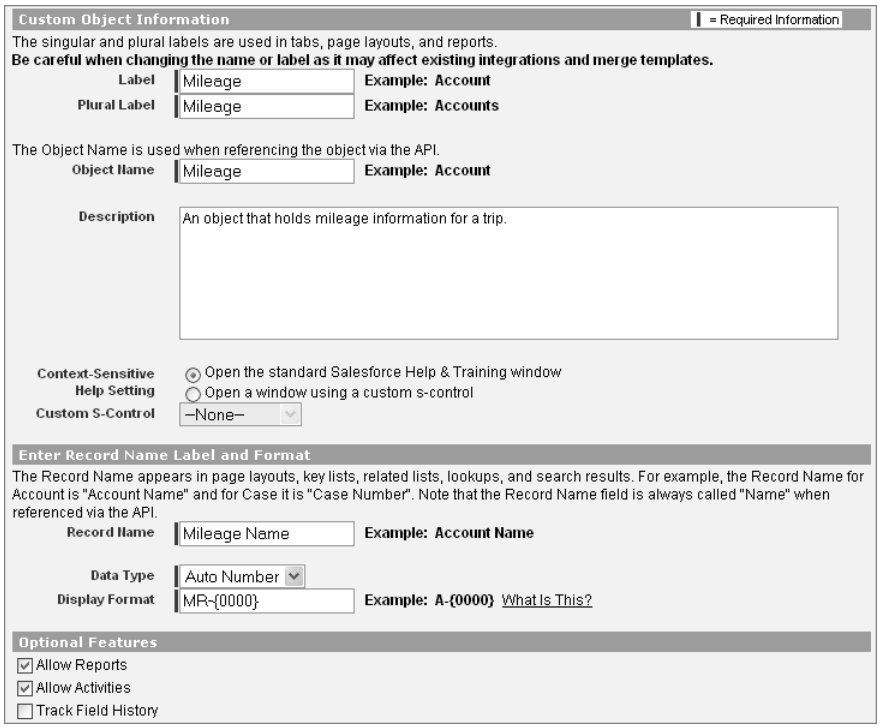
2. Create the Mileage record custom object.

   a. Click **Setup** in the upper right corner to go to the setup area.



   b. Click **Create ➤ Objects** in the sidebar menu to display the Custom Objects page.
   c. Click **New Custom Object** to display the New Custom Object wizard.
   d. Fill in the custom object definition:

      - For the `Label`, enter `Mileage`.
      - For the `Plural Label`, enter `Mileage`.
      - For the `Object Name`, `Mileage`.
      - For the `Description`, enter `An object that holds mileage information for a trip`.
      - Select the `Allow Activities` checkbox.
      - Select the `Allow Reports` checkbox.
      - Leave all other values as they are.



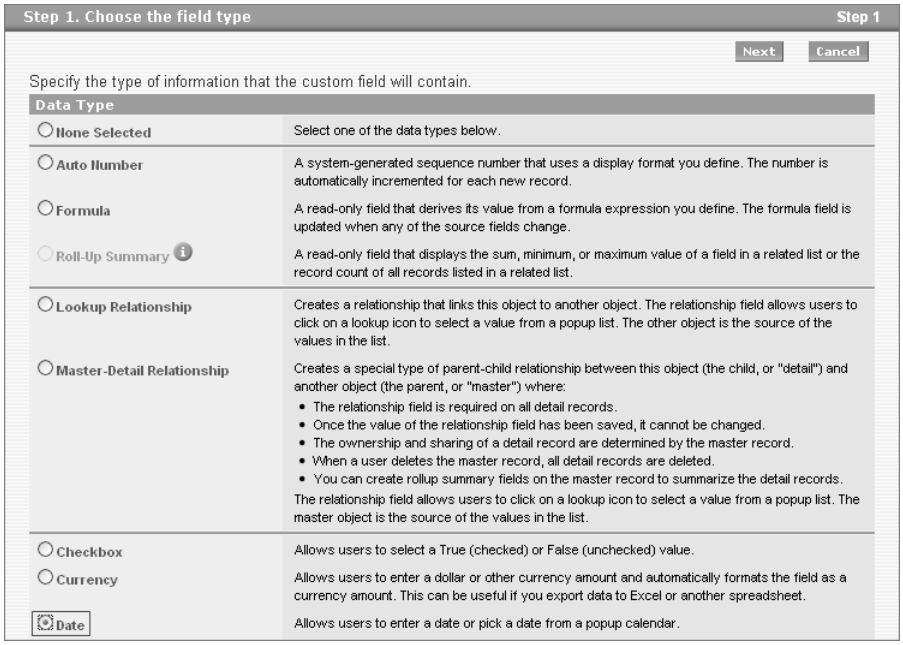
e. Click **Save** to finish creating your new object.

Step 2: Create Custom Fields

Now that you have created the Mileage object that will hold each mileage record, you will want to add fields to capture the date of the trip, the number of miles driven, and the person visited.

1. Create the `Date` field.

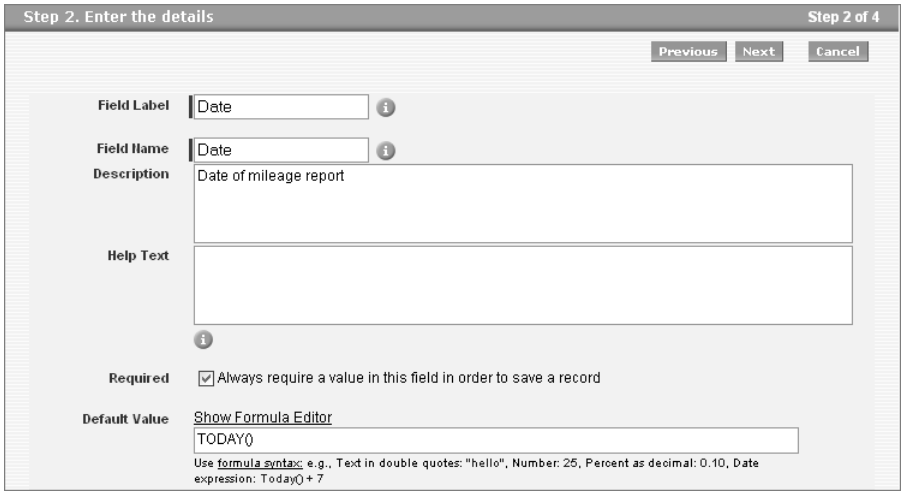
a. Scroll down to the Custom Fields & Relationships related list.
b. Click **New** to launch the New Custom Field wizard.
c. For Data Type, select `Date` and click **Next**.



d. Fill in the custom field details:

- For the `Field Label`, enter `Date`.
- For the `Field Name`, enter `Date`.
- For the `Description`, enter `Date of mileage report`.
- Check the `Required` checkbox.
- For the `Default Value`, enter `TODAY()`

  `TODAY()` is a built-in formula that automatically populates today's date.



e. Click **Next**, accept the defaults, and click **Next** again.

f. Click **Save & New** to create the `Date` field and to return to the first step of the wizard.

2. Create the `Miles` field.

a. For Data Type, select `Number`, and click **Next**.

b. Fill in the custom field details:

- For the `Field Label`, enter `Miles`.
- For the `Length`, enter `4`.
- For the `Decimal Places`, enter `0`.
- For the `Field Name`, enter `Miles`.
- For the `Description`, enter `Miles driven`.
- Check the `Required` checkbox.

c. Leave the defaults for the remaining fields, and click **Next**.
d. In the next step, accept the defaults, and click **Next**.
e. In the next step, click **Save & New** to create the `Miles` field and to return to the first step of the wizard.

3. Create the `Contact` field.

a. For Data Type, select `Lookup Relationship` and click **Next**. The Lookup Relationship data type lets you link two data objects—in this case, the Mileage object and the Contact object.
b. In the `Related To` drop-down list, select Contact, and click **Next**.
c. Fill in the custom field details:

- For the `Field Label`, enter `Contact`.
- For the `Field Name`, enter `Contact`.
- For the `Description`, enter `Person that was seen on this trip`.

d. Leave the defaults for the remaining fields, and click **Next**.
e. In the next step, accept the defaults, and click **Next**.
f. Click **Next** two more times.
g. Click **Save** to create the `Contact` field and to return to the Mileage Custom Object page.

Step 3: Create a Tab

To create a tab in the online user interface for the Mileage custom object, you need to add a custom tab to the existing standard tabs. When users click this tab, they will be able to create, view, and edit Mileage records.

1. Within the setup area, click **Create ➤ Tabs**.
2. Click **New** in the Custom Objects tab related list to launch the New Custom Tab wizard.
3. From the `Object` drop-down list, select Mileage.
4. For the `Tab Style`, click the lookup icon () and select the Car icon.

5. Accept the remaining defaults, and click **Next**.
6. Click **Next** and then **Save** to complete the creation of the tab.

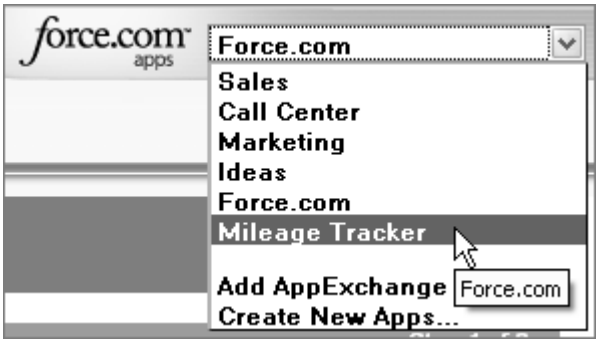
As soon as you create the tab, you can see the new tab included in the set of tabs.

Step 4: Create an App

You now have created three fields and a tab. You can combine these objects, plus others such as dashboards and reports, to create an application.

1. Navigate to **Setup ➤ Create ➤ Apps**.
2. Click **New** to launch the New Custom App wizard.
3. Fill in the custom app details:
   - For the `App Label`, enter `Mileage Tracker`.
   - For the `App Name`, enter `Mileage_Tracker`.
   - For the `Description`, enter `This application tracks mileage records`.
4. Click **Next**.
5. Accept the defaults on the next page, using the default logo for the app, and click **Next**.
6. In the Available Tabs box, locate the Mileage tab and click **Add** to add it to the box of selected tabs.
7. Repeat the above step, this time selecting the Contacts Tab.
8. Leave the `Default Landing Tab` set to the Home tab, and click **Next**.
9. Select the `Visible` checkbox to make the application available to all user profiles.
10. Click **Save** to create the Mileage Tracker application.

    As soon as you create the application, it appears in the Force.com app menu in the upper right of the page.



Step 5: Enter Sample Data

Now check to see how your newly-created app works.

1. Select the Mileage Tracker application from the Force.com app menu.
2. Click the Mileage tab and then click **New** to create a new mileage record.
3. Enter some test data, but do not enter more than 500 miles for any single record because you may want to complete Tutorial #5: Adding Business Logic with Apex on page 29, which will add business logic to limit daily mileage to 500.
4. For the `Contact` field, click the lookup icon, click **New** to create a new contact named `Tim Barr`, and then click **Save** within the lookup dialog.
5. Click **Save** to save your record and to return to the detail page for the newly-created record.

Step 6: Set Up Auto-numbering

When you create new records in this app, you enter a name for each record to identify it. However, to make life easier and to make sure each record has a unique identifier, the Force.com platform can automatically assign record numbers. In this step, you will set up this functionality.

1. Click **Setup ➤ Create ➤ Objects**.
2. Click the Mileage custom object.
3. Scroll down to the Standard Fields related list, locate the `Mileage Name` field, and click **Edit**.
4. Change the `Data Type` to Auto Number.
5. For `Display Format`, enter `MR-{0000}`.
6. For `Starting Number`, enter `1`. If you have created Mileage records already, those existing records will not use the new auto-numbering; they will retain their existing name.
7. Click **Save** to save your changes.
8. Repeat Step 5: Enter Sample Data on page 7. Notice how you no longer have to enter a mileage name.

Summary

Congratulations, you have just built an application—without any coding. Using the online user interface, you simply followed the steps in each wizard to create an object, fields, and a tab, and then pulled it all together into an application that lets users create mileage records to track their business trips.

Tutorial #2: Creating an App with the Force.com IDE

Level: Intermediate; **Duration:** 30 minutes

The Force.com IDE is a toolkit extension to the Eclipse integrated development environment (IDE) that helps developers and development teams build, deploy, and track versions of Force.com components, including custom objects and fields, Apex classes, and Visualforce pages.

In this tutorial, you will build a Mileage Tracker app using both the Force.com IDE and the online user interface. You will learn how to change the metadata that describes the structure of an app by manipulating XML descriptors in the Force.com IDE. You can change the metadata much more quickly in the IDE than by using the online user interface, especially when doing bulk operations such as migrating schemas or cloning a Force.com organization.

Prerequisites

Basic Knowledge

For this tutorial, it helps to have basic knowledge of XML, but it is not required.

Software Requirements

- Eclipse 3.3: http://wiki.apexdevnet.com/index.php/Force.com_IDE_Installation_for_Eclipse_3.3.x
- Force.com IDE

Getting Started

If you have already created the Mileage Tracker application or downloaded it in Tutorial #10: Downloading and Deploying an App Using Code Share on page 57, you must first delete all the components you created—including the custom object, the tab, and the application—so that the names don't conflict with those in this tutorial.

Step 1: Create a Project in the IDE

Use the following procedure to create a project and connect it to the Developer Edition (home) organization:

1. Start Eclipse. Select **File ➤ New ➤ Force.com Project**

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

2. Specify your Developer Edition organization credentials:

   a. Enter a project name.
   b. Enter your Developer Edition username.
   c. Enter your Developer Edition password.
   d. Accept the remaining defaults and click Next.

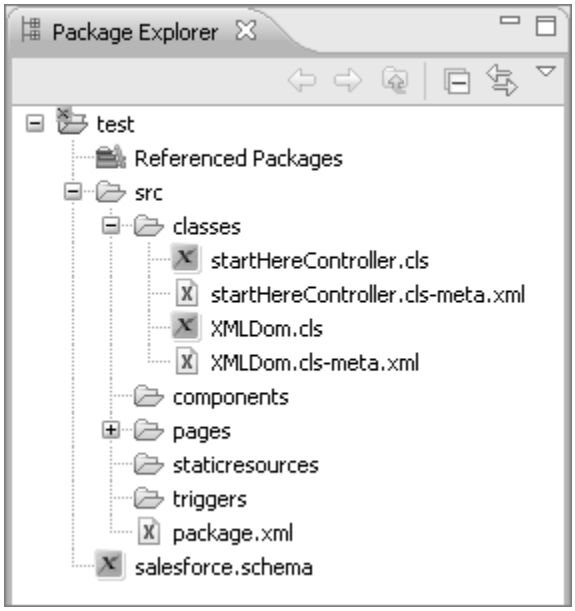
   Note: If you get a LOGIN_MUST_USE_SECURITY_TOKEN error, it means your current IP is not recognized from a trusted network range. You must reset your security token. From the Salesforce user interface, select **Setup ➤ My Personal Information ➤ Reset My Security Token** and follow the instructions there.

3. On the Project Contents page, click **Finish**.

Note: If you plan to complete Tutorial #5: Adding Business Logic with Apex or Tutorial #7: Building a Custom User Interface Using Visualforce, your project is now configured to retrieve those components.

4. In the Package Explorer, expand the **src** folder and the folders within to see a list of items you've retrieved from your organization.

If this is a new Developer Edition org, you'll see the following:

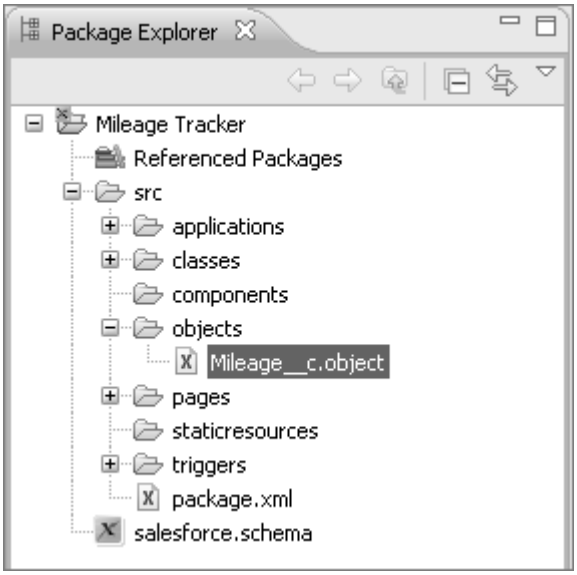


Notice that the platform creates Apex classes and a Visualforce page to get you started. Notice also that there are no other metadata components retrieved at this time. When you create a project, you have the option to retrieve metadata components, but since you haven't created any yet, there would be little use for that now. You'll add a custom object in the next step.

Step 2: Create the Mileage Record Custom Object

In the Force.com platform, an object is similar to a database table: a list of information, organized in records and fields, about the person, thing, or concept you want to track. Each object automatically generates a user interface through a tab and uses built-in features such as security and sharing, workflow processes, and much more.

1. Create a custom object:
    a. Right-click (Ctrl-click on a Mac) on your project's folder and click **New ➤ Custom Object**.
    b. Enter `Mileage` in the **Label** field and then tab to the next field. Notice that the **Name** field is completed for you.
    c. Tab out of the field and click **Finish**.
    d. In the Package Explorer, expand the nodes for **src ➤ objects**. You should now see the custom object you just created, called **Mileage__c.object** in the **objects** folder.



An important thing happened to your project when you created your custom object just now; you changed the project manifest. The project manifest determines what components you retrieve in your project, and is controlled by the `package.xml` file. When you use the IDE to create a metadata component, that component is automatically added to the project manifest.

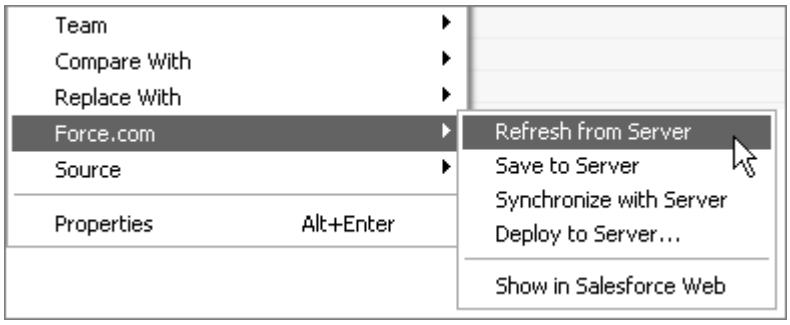
2. Edit the custom object:
   a. The new custom object definition should already be open in the XML editor. At the bottom of the editor, notice there are tabs for **Design** and **Source** view. Make sure you are in the **Source** view.
   b. In the XML editor, change the label of the `nameField` from `Custom Object Label Name` to `Mileage Report Number`. This field will be used to identify the record in the Mileage object.
   c. Click the **Save** icon in the menu bar. When you save changes, they are saved both in your project and on the server.
3. Verify that the object exists on the server:
   a. In the Package Explorer, right-click the **Mileage__c.object** and choose **Force.com ➤ Show in Salesforce Web**.
   b. You should now see the page describing the Mileage__c custom object.

Step 3: Auto-number Mileage Records

The next step is to auto-number the records so the user does not have to specify a name every time a Mileage record is created. You could implement this in the IDE, but to show how the project and server are synchronized, you'll do this in the browser:

1. In the Package Explorer, right-click the **Mileage__c.object** and choose **Force.com ➤ Show in Salesforce Web**. You should now see the page in your browser describing the Mileage__c custom object.
2. In the Standard Fields section of the page, find the field labeled `Mileage Report Number` that you renamed earlier.
3. Click the **Edit** link next to the `Mileage Report Number` field.
   - Change the data type to `Auto-Number`.
   - Specify the display format to be `MR-{0000}`.
   - Specify the starting number to be `1`.
4. Click **Save**.
5. Refresh the IDE project so that the changes you made in the online user interface exist in your project:

a. Go back to the Force.com IDE Package Explorer, right-click the **Mileage__c.object**, and choose **Force.com ➤ Refresh from Server**.



b. Click **Yes** to confirm the refresh. You should now see a new element under the `nameField`, called `displayFormat`. The value of `Type` changed from `Text` to `AutoNumber`.

Note: Refresh from Server replaces your project files with the files from the server. Only the files you specify in your project manifest are retrieved.

Step 4: Create Custom Fields in the Browser

Now that you have created the Mileage object, you will want to add a field to capture the date of the record.

1. In the Package Explorer, right-click the **Mileage__c.object** and choose **Force.com ➤ Show in Salesforce Web**.
2. In the Mileage custom object detail page, in the Custom Fields & Relationships area, click **New** to launch the field creation wizard.

3. In Step 1 of the wizard, select `Date` as the Data Type and click **Next**. This field will be used to store the date of the mileage report.
4. In Step 2, fill in the custom field details:
   - For the `Field Label`, enter `Date`.

- For the `Field Name`, enter `Date`.
- For the `Description`, enter `Date of mileage report`.
- Select the `Required` checkbox.
- For the `Default value`, enter `TODAY()`

5. Click **Next**.
6. In Step 3, accept the field-level security defaults and click **Next**.
7. In Step 4, accept the default (to add the field to the page layout) and click **Save**. You have just created a new custom field for your custom object, made it accessible to all user profiles, and automatically added it to the page layout.
8. Return to the IDE.
9. In the Package Explorer, right-click the **Mileage__c.object** and select **Force.com ➤ Refresh from Server** to refresh your project files with changes you made on the server.
10. Click **Yes** to confirm the refresh. You will see the entry for the `Date` field that you added.

Step 5: Create Custom Fields in the IDE

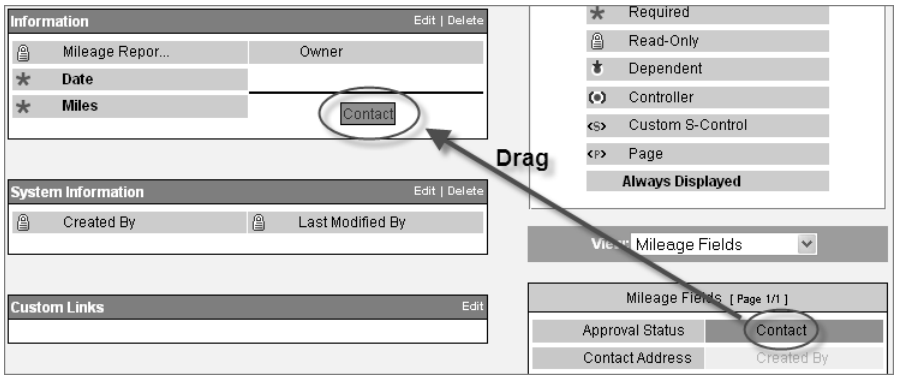
Next you will add two additional fields: `Contact` and `Miles`, but this time you'll add them in the IDE.

1. In the XML editor, the Mileage__c.object should already be open. Insert the text shown in bold to add Contact__c and Miles__c. You'll notice that as soon as you start typing, the Force.com IDE supports code-completion.

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
<deploymentStatus>Deployed</deploymentStatus>
 <description> This is a generic template for CustomObject. With this template, you may
 adjust the default elements and values and add new elements and values.
 </description>
 <fields>
  <fullName>Date__c</fullName>
  <description>Date of Mileage Report</description>
  <label>Date</label>
  <required>true</required>
  <type>Date</type>
 </fields>
 <fields>
  <fullName>Contact__c</fullName>
  <label>Contact</label>
  <referenceTo>Contact</referenceTo>
  <relationshipName>Mileage</relationshipName>
  <type>Lookup</type>
 </fields>
 <fields>
  <fullName>Miles__c</fullName>
  <label>Miles</label>
  <precision>4</precision>
  <required>true</required>
  <scale>0</scale>
  <type>Number</type>
 </fields>
 <label>Mileage</label>
 <nameField>
  <displayFormat>MR-{0000}</displayFormat>
  <label>Mileage Report Number</label>
  <type>AutoNumber</type>
 </nameField>
 <pluralLabel>Mileage</pluralLabel>
 <sharingModel>ReadWrite</sharingModel>
</CustomObject>
```

2. Click **Save** to automatically save the changes to your organization.
3. Verify the fields in the browser and add them to the page layout:

   a. In the Package Explorer, right-click **Mileage__c.object** and choose **Force.com ➤ Show in Salesforce Web**. You should see the `Contact` and `Miles` fields in the Custom Fields & Relationships area, along with the `Date` field you created earlier.
   b. In the Page Layouts section, click the **Edit** button next to Mileage Layout.
   c. Drag and drop the `Contact` field (on the bottom right of the page) into the Information section. Note that the `Miles` field was automatically added to the page layout because you specified it as required in the XML.



   d. Click **Save**.

Step 6: Create a Tab

In this step, you will create a tab that exposes the Mileage custom object to the user through a tab at the top of the online user interface. The tab will display the most recently accessed items for the object as well as the buttons to bring up pages for creating new Mileage records or editing existing records. When you created a custom object in the IDE, the project manifest was updated automatically. This time you'll create a new tab in the Web, and that will require changing your project manifest so that you can retrieve the tab.

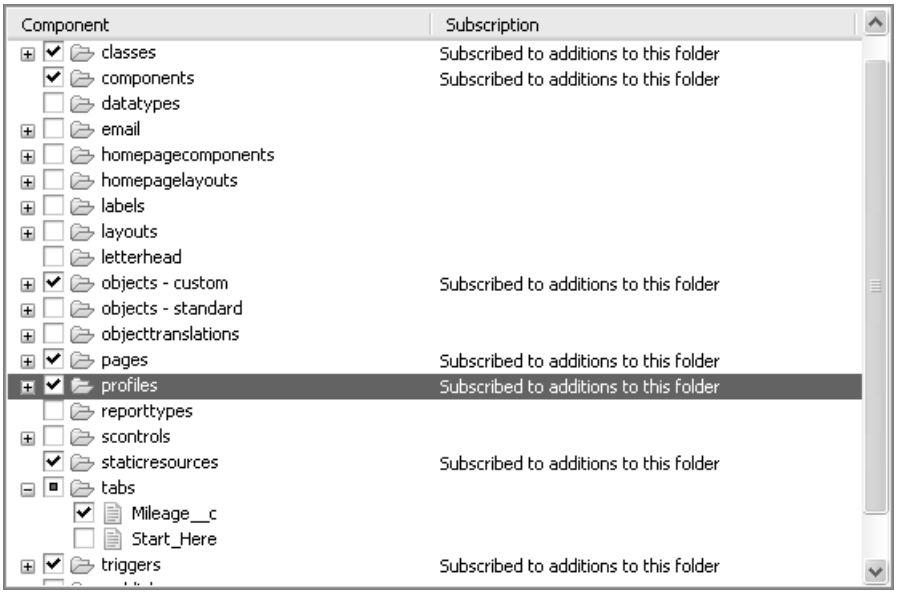
1. Click **Setup ➤ Create** and click **Tabs**.
2. Click **New** at the top of the Tabs list to launch the New Custom Tab wizard.
3. Select `Mileage` for the `Object`.
4. Select `Car` for the `Tab Style`.
5. Click **Next**.
6. In Step 2, accept the defaults and click **Next**.
7. In Step 3, accept the defaults and click **Save**. You should now see a Mileage tab as part of your tab set.
8. In the IDE Package Explorer, right-click the **src** folder and choose **Force.com ➤ Refresh from Server**. Notice that there is no **tabs** folder in the Package Explorer. To retrieve tabs, you'll edit your project properties in the next step.

Step 7: Edit Project Properties

When you created a custom object in the IDE, the project manifest was updated automatically, because the IDE was aware that you created the object. In the previous step you created a new tab in the Web, and so the IDE didn't know of its existence. So you need to change your project manifest to retrieve the tab. In the next step, you'll need to edit a profile, so right now you'll edit your project manifest to retrieve the new tab you created and all profiles.

1. In the Package Explorer, right-click your project and choose **Properties**.
2. In the navigation tree, expand the **Force.com** node and click **Project Content**.

3. Click **Add/Remove** to open the Choose Metadata Components dialog.
4. Expand the checkbox for **tabs** and select the Mileage__c custom tab. Notice that the Choose Metadata Components dialog allows you to drill down into the component folders and retrieve only the items you want.
5. Select the folder-level checkbox for profiles. When you select the entire folder, the project will retrieve all of the components of that type, even new ones that you create.



6. Click **OK**.
7. A dialog box will prompt you about refreshing from the server, click **OK**.
8. Expand the nodes in the Package Explorer, and you'll see your project now retrieves the custom tab you created, and all profiles.

Step 8: Create an App

Apps can combine several tabs and include other Force.com components, such as dashboards and reports. In this step, you will create an application containing the Mileage custom object and the Contact standard object, and then make the application visible to the default Administrator profile.

1. Create the app in the IDE.
   a. In the Package Explorer, right-click your project folder and select **New ➤ Custom Application**.
   b. In the **Label** field, enter `Mileage Tracker`.
   c. Tab to the **Name** field and then click **Finish**. The Mileage Tracker opens in the XML editor.
   d. Edit the Mileage Tracker app so that it resembles the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomApplication xmlns="http://soap.sforce.com/2006/04/metadata">
    <defaultLandingTab>standard-home</defaultLandingTab>
    <description>Mileage Tracker Application</description>
    <label>Mileage Tracker</label>
    <tab>Mileage__c</tab>
    <tab>standard-Contact</tab>
</CustomApplication>
```

   e. Click **Save**.

Note: When you save the app, the server automatically updates your organization's profiles. A profile is a collection of settings and permissions that determine what a user can see or do. To allow the default Administrator profile to see the new app, you must make it visible to the Administrator profile.

2. Configure user profiles:

   a. To bring the updated profiles back to the IDE, right-click the **src** folder and choose **Force.com ➤ Refresh from Server**.
   b. Expand the **src ➤ profiles** folder, double-click the `Admin.profile` file to open it in the XML editor.
   c. Change the value of the `<visible>` element for Mileage Tracker to `true`, as illustrated in the bold text below:

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile xmlns="http://soap.sforce.com/2006/04/metadata">
 <applicationVisibilities>
  <application>Mileage Tracker</application>
  <default>false</default>
  <visible>true</visible>
 </applicationVisibilities>
 ...
</Profile>
```

   d. Click **Save**.
   e. Go back to the browser window and refresh. You should now see an app called Mileage Tracker in the **Force.com app menu** at the top right of the page.

Step 9: Test the App

You can now test the app's functionality by creating a new record.

1. In the browser, select the Mileage Tracker application from the Force.com app menu.
2. Click the Mileage tab and click **New** to create a new record.
3. Enter some test data and save the record.

Note: If you are going to do Tutorial #5: Adding Business Logic with Apex on page 29, enter `Pat Stumuller` as the contact's name and keep the miles under 500.

Summary

In this tutorial, you created the basic Mileage Tracker app with the Force.com IDE. Some key points are the following:

- When you create a project in the IDE, you can select the metadata components you want to retrieve, and you can later modify the project manifest.
- You can create and edit Force.com components in the Web or in the IDE.
- You can edit the metadata files of a Force.com organization to change setup information.
- When you save your project, the files are saved directly to the server. You can refresh from the server to overwrite the files in your project with files from the server.

Tutorial #3: Using Formulas and Validation

Level: Beginner; **Duration:** 20-30 minutes

The Force.com platform includes the capability to create formulas and field validation rules to help you maintain and enhance the quality of the data that is entered in your app. Both formula fields and field validation rules utilize built-in functions that allow you to automatically manipulate your data, validate your data, and calculate other values based on your data. The functions you use in formula fields and field validation rules are much like the functions you would use in a spreadsheet program, where you can reference values in other cells of the spreadsheet, perform calculations, and return a result. However, with formula fields and validation rules, you reference fields in your app's records rather than cells in a spreadsheet.

In this tutorial, you will enhance the Mileage Tracker app by creating one field validation rule to ensure that users do not enter dates in the future when logging their mileage. You will also create two custom formula fields: one to calculate kilometers from the miles entered, and another to display and map the contact's address using Google Maps.

Prerequisites

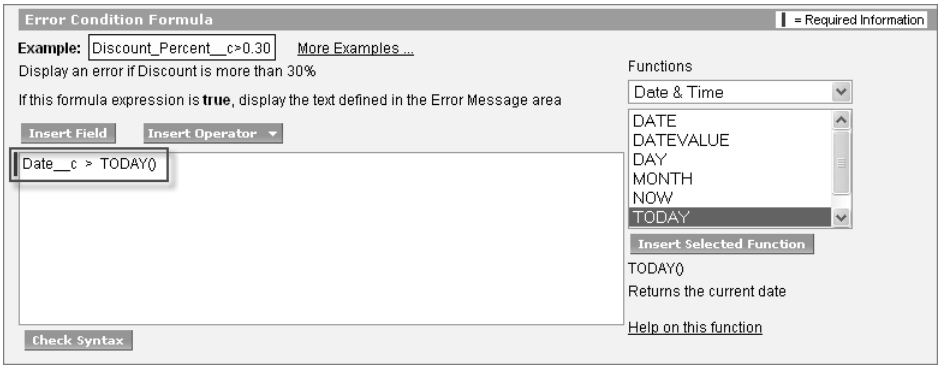
Mileage Tracker App
You first need to create the basic Mileage Tracker application as described in Tutorial #1: Creating an App with Point-and-Click Tools on page 3 or Tutorial #2: Creating an App with the Force.com IDE on page 9.

Step 1: Validate Date Entries using Field Validation

In the first step of this tutorial, you will create a simple formula to validate the data entered into the `Date` field of the Mileage custom object in the Mileage Tracker app. This type of field validation is known as a *validation rule* on the platform. This validation rule will ensure that users do not use any dates in the future when they enter the miles they have traveled. Simple validation rules like this help ensure the quality of your app's data.

1. Click **Setup ➤ Create ➤ Objects** to navigate to the setup area.
2. Click the **Mileage** custom object name.
3. Scroll down to the Validation Rules related list, and click **New**.
4. Specify the rule name: `Date_Validation`.
5. Leave the `Active` checkbox selected.
6. Specify a description: `Validates the Date field so that it cannot be in the future`.
7. In the Error Condition Formula section, click **Insert Field** to open the Insert Field popup window.
   a. Select `Mileage >` in the first column.
   b. Select `Date` in the second column.
   c. Click **Insert**.
8. With your cursor in the formula box after `Date__c`, click **Insert Operator** and select **> Greater Than** from the drop-down button.
9. Next, select `TODAY` in the Functions box, and click **Insert Selected Function**.

   Your formula now looks like this: `Date__c > TODAY()`



10. In the `Error Message` field, specify the following error message: `Date cannot be in the future`.
11. For the `Error Location`, select `Field`, then choose `Date` from the drop-down list.
12. Click **Save**.

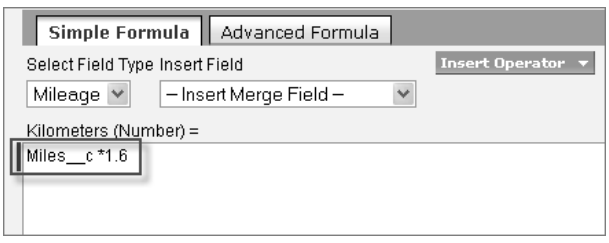
Now try it out:

1. Go to the Mileage tab.
2. Click **New**.
3. Fill in all fields making sure to specify a date in the future.
4. Click **Save**. You should have triggered an error specifying that the date cannot be in the future.

Step 2: Display Mileage in Kilometers Using a Formula Field

Now you will use a formula field to automatically calculate the number of kilometers traveled in each Mileage record. With this formula field, users only need to enter the number of miles traveled in the `Miles` field. The formula will then automatically convert that number to kilometers and display it on the Mileage record detail page.

1. Navigate back to the Mileage custom object page by clicking **Setup ➤ Create ➤ Objects ➤ Mileage**.
2. Scroll down to the Custom Fields & Relationships related list, and click **New**.
3. Choose `Formula` as the field type, and click **Next**.
4. Fill in the custom field details:
   - For the `Field Label`, enter `Kilometers`.
   - For the `Field Name`, enter `Kilometers`.
   - For the `Formula Return Type`, select `Number`.
   - For the `Decimal Places`, select `2`.
5. Click **Next**.
6. In the Insert Merge Field drop-down list, select `Miles`. You should now see `Miles__c` in the text box.
7. In the formula text box, multiply Miles__c by 1.6 to get kilometers. The formula should now look like this:



8. Specify a description: `Mileage in kilometers`. Then click **Next**.

9. In the Establish field-level security step of the New Custom Field wizard, accept the defaults, and click **Next**.
10. In the Add to page layouts step, accept the defaults, and click **Save**.

Now try it out:

1. Click the Mileage tab.
2. Click **New**.
3. Fill in all fields. Notice that the new formula field does not display on the edit page of the record. Formula fields only display on the detail page after you save the record.
4. Click **Save**. You should now see miles automatically converted to kilometers in the newly-added field.

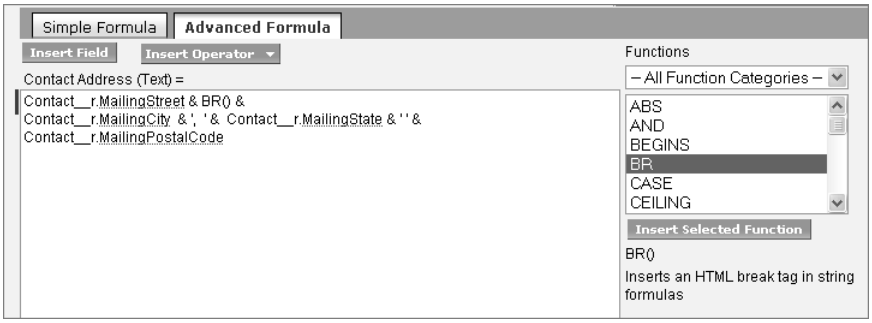
Step 3: Display the Contact's Address Using a Formula Field

In this step, you will create another formula field on the Mileage custom object. This formula is slightly more advanced, but just as easy to create. Remember that in our app, every Mileage record is associated with a Contact record through a lookup relationship field. To make the Mileage records more useful, we want to display the contact's address information directly on the Mileage record. To do that, we'll create a new Mileage object formula field to reference fields on the Contact object. This capability to reference and calculate formulas based on fields in related objects is known as a *cross-object formula*.

1. Navigate back to the Mileage custom object page by clicking **Setup ➤ Create ➤ Objects ➤ Mileage**.
2. Scroll down to the Custom Fields & Relationships related list, and click **New**.
3. Choose `Formula` as the field type, and click **Next**.
4. Fill in the custom field details:
   - For the `Field Label`, enter `Contact Address`.
   - For the `Field Name`, enter `Contact_Address`.
   - For the `Formula Return Type`, select `Text`.
5. Click **Next**.
6. Click the Advanced Formula tab.
7. Click **Insert Field** to open the Insert Field popup window.
   a. Select `Mileage >` in the first column.
   b. Select `Contact >` in the second column.
   c. Select `Mailing Street` in the third column.
   d. Click **Insert**.
8. Insert more contact address fields with the **Insert Field** button, and use the `&` concatenate operator and the `BR` new line function to join the fields into the following formula:

```
Contact__r.MailingStreet & BR() &
Contact__r.MailingCity & ', ' & Contact__r.MailingState & ' '& Contact__r.MailingPostalCode
```

   Notice that the `&` operator concatenates the fields and any punctuation (like the comma between the city and state) together. Plain text, such as the comma and the space between the state and zip code, are enclosed in single quotes. The `BR()` function introduces a new line between the street and the rest of the address.



9. Click **Check Syntax** to ensure there are no errors. If there are errors, fix them before proceeding.
10. Specify a description: `Display the contact's address`. Then click **Next**.
11. In the Establish field-level security step of the New Custom Field wizard, accept the defaults, and click **Next**.
12. In the Add to page layouts step, accept the defaults, and click **Save**.

Now try it out:

1. Go to the Mileage tab.
2. Click **New**.
3. Fill in all fields. Make sure you choose Pat Stumuller for the `Contact` field; that record has an address that can be mapped.
4. Click **Save**. You should now see the contact's address automatically displayed on the Mileage record.

Step 4: Link the Contact's Address to a Mapping Service

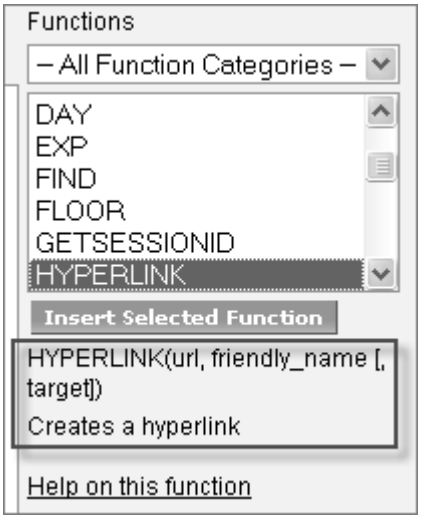
Now you will build on the `Contact Address` formula field you just created to build a mash-up between your Force.com data and Google maps. You will use the `HYPERLINK` function to pass the contact's address to Google Maps.

1. Navigate back to the Mileage custom object page by clicking **Setup ➤ Create ➤ Objects ➤ Mileage**.
2. Scroll down to the Custom Fields & Relationships related list.
3. Click **Edit** next to the `Contact Address` field.
4. Place your cursor after the existing formula in the box, and click **Insert Operator** and select `& Concatenate`, then add `' ' &` to add a space before the map link. Your formula should now look like this:

```
Contact__r.MailingStreet & BR() &
Contact__r.MailingCity & ', ' & Contact__r.MailingState & ' '& Contact__r.MailingPostalCode
& ' ' &
```

5. In the Functions box, select `HYPERLINK`.

   When you select a function, the syntax for that function displays. The `HYPERLINK` function requires a *`url`*, *`friendly_name`*, and optionally, a *`target`*. For this formula, you will only use the *`url`* and *`friendly_name`*.



6. Click **Insert Selected Function**, and modify the `HYPERLINK` function to look like this.

```
HYPERLINK('http://maps.google.com/?q=' & Contact__r.MailingStreet & ', ' &
Contact__r.MailingPostalCode, ' Click to Map')
```

7. Click **Check Syntax** to ensure there are no errors. If there are errors, fix them before proceeding.

   The entire formula should look like this:

```
Contact__r.MailingStreet & BR() &
Contact__r.MailingCity & ', ' & Contact__r.MailingState & ' '& Contact__r.MailingPostalCode
&
HYPERLINK('http://maps.google.com/?q=' & Contact__r.MailingStreet & ', ' &
Contact__r.MailingPostalCode, 'Click to Map')
```

8. Click **Save**.

Now try it out:

1. Go to the Mileage tab.
2. Click the Mileage record you created in the previous step.
3. Click the **Click to Map** link in the `Contact Address` field. The link should automatically go to a Google map showing the contact's address.

Summary

In this tutorial, you utilized the built-in functions of the platform to create formula fields and validation rules to enhance and validate your app's data. The validation rule you created ensures that users do not enter any dates in the future. This same principle can be applied to any standard or custom field in your app where you want to make sure that the data entered meets certain criteria. The first formula field you created was a simple calculation of miles to kilometers, but it demonstrates the power of the platform to leverage the data entered by your users into even more useful information. The final formula field you created was done in two steps—first to pull Contact information into the Mileage record, and second to use that information to integrate with Google Maps. Because your app is delivered on the Web as a service, this type of integration is easy to accomplish with a simple formula and URL. The possibilities for other types of integrations are limited only by your imagination.

Tutorial #4: Using Workflow and Approvals

Level: Intermediate; **Duration:** 30 minutes

Your organization operates more efficiently with standardized internal procedures and automated business processes. You can use workflow rules and approval processes to automate your procedures and processes. Not only do you save time, but you also enforce consistency across your business practices. Build workflow rules to trigger actions, such as email alerts, tasks, field updates, and outbound messages based on time triggers, criteria, or formulas. Use approval processes to automate all of your organization's approvals, from simple to complex.

In this tutorial, you create and test a workflow rule that runs when a mileage record is created or updated with a value greater than 100 miles. An email notification is sent when this happens. In the second part of the tutorial, you create and test an approval process that requires an approval for any submitted mileage record greater than 200. An email notification is also sent when this condition is encountered.

Prerequisites

Basic Knowledge
For this tutorial, it helps to have basic knowledge of Salesforce administration.

Mileage Tracker App
You first need to create the basic Mileage Tracker application as described in Tutorial #1: Creating an App with Point-and-Click Tools on page 3 or Tutorial #2: Creating an App with the Force.com IDE on page 9.

Step 1: Create a User to Receive Approval Requests

To begin, create a new user to act as the approver for an approval request that is covered in a later step of this tutorial.

1. Click **Setup ➤ Manage Users ➤ Users**.
2. On the All Users page, click **New User**.
3. Set `First Name` to `Bob`.
4. Set `Last Name` to `Smith`.
5. Set `Alias` to `bsmith`.
6. Set `Email` to your email address so that you receive the approval requests routed to Bob Smith.
7. Set `Username` to a unique username for Bob Smith.
8. Set `Manager` to your initial user—the one you created when you signed up for your Developer Edition organization.
9. Set `Profile` to `Standard User`.
10. Click **Save**.

You should now receive an email confirming the creation of the new user. Log in with the temporary password provided, and change the password when prompted.

Step 2: Create Email Templates

Next, create the email templates to be used by the workflow rule and approval process that you create in later steps. An email template is a form email that communicates a standard message. The first email template is used for sending an email alert when a mileage record exceeds 100 miles—covered in the workflow part of this tutorial. The second email template is used in the approval process part of this tutorial.

Create a folder to store the email templates:

1. Click **Setup ➤ Communication Templates ➤ Email Templates**.

2. On the Email Templates page, click **Create New Folder** next to the Folder drop-down list.

3. Set `Email Template Folder Label` to `MileageTracker Email Templates`.
4. Set `Folder Unique Name` to `MileageTracker_Email_Templates`.
5. Accept the remaining defaults and click **Save**.

Create the first template:

1. In the MileageTracker Email Templates folder, click **New Template**.
2. In Step 1 of the Email Template wizard, select `Text` and click **Next** to continue to Step 2 of the wizard.
3. Select `Available for Use`.
4. Set `Email Template Name` to `Large Mileage Notification`.
5. Set `Template Unique Name` to `Large_Mileage_Notification`.
6. Set `Subject` to `Large Mileage Notification`.
7. Set `Email Body` to `{!Mileage__c.OwnerFullName} submitted a Mileage record with {!Mileage__c.Miles__c} miles on {!Mileage__c.LastModifiedDate}.`
8. Click **Save**.

Create the second template.

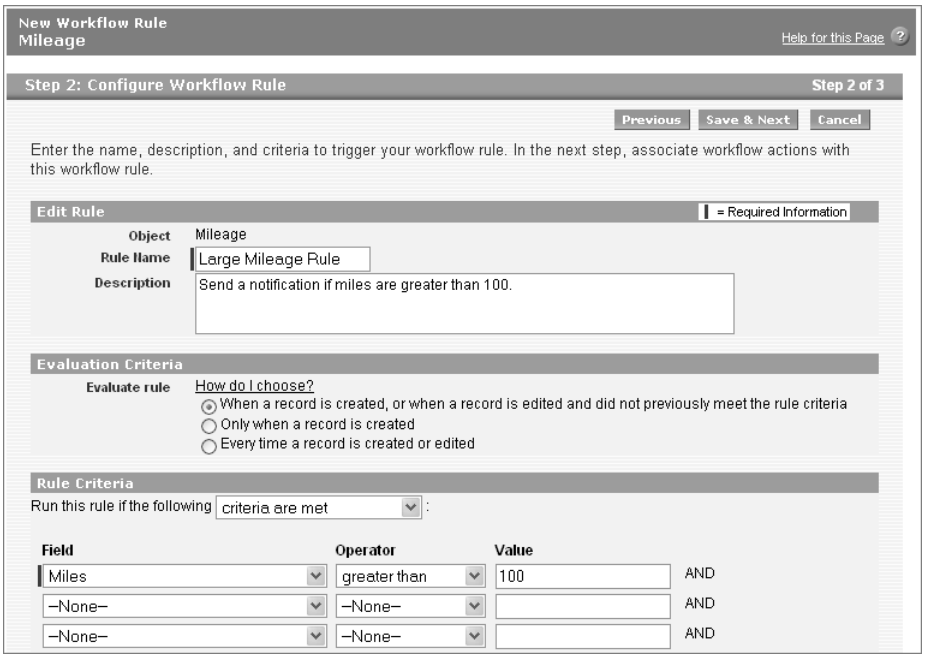
1. Click **Setup ➤ Communication Templates ➤ Email Templates**.
2. Select the `MileageTracker` Email Templates folder from the `Folder` drop-down list.
3. Click **New Template**.
4. In Step 1 of the Email Template wizard, select `Text` and click **Next** to continue to Step 2 of the wizard.
5. Select `Available for Use`.
6. Set `Email Template Name` to `Large Mileage Approval Notification`.
7. Set `Template Unique Name` to `Large_Mileage_Approval_Notification`.
8. Set `Subject` to `Large Mileage Approval Notification`.
9. Set `Email Body` to `{!Mileage__c.OwnerFullName} submitted for approval a Mileage record with {!Mileage__c.Miles__c} miles on {!Mileage__c.LastModifiedDate}.`

Step 3: Create Workflow

In this step, we create a workflow that runs when a Mileage record exceeds 100 miles. An email is then sent to notify the designated recipient and the person who submitted it that a large mileage has been entered.

1. Click **Setup ➤ Create ➤ Workflow & Approvals ➤ Workflow Rules**.
2. If you see the Understanding Workflow screen, click **Continue**; otherwise proceed to the next step.
3. On the All Workflow Rules page, click **New Rule**.
4. In Step 1 of the Workflow Rule wizard, select Mileage as the object and click **Next** to continue to Step 2 of the wizard.
5. Set `Rule Name` to `Large Mileage Rule`.
6. Set `Description` to `Send a notification if miles are greater than 100.`

7. In `Evaluation Criteria` leave the default set to `When a record is created or when a record is edited and did not previously meet the rule criteria`.
8. Set `Rule Criteria` to `Miles greater than 100`.



9. Click **Save & Next**.
10. In Step 3 of the Workflow Rule wizard, in the Immediate Workflow Actions section, click **Add Workflow Action** and choose New Email Alert.

Note: If you do not see the email template you created earlier, make sure you selected the `Available for Use` when you created the template.

11. Set `Description` to `Sends an email when mileage exceeds 100 miles.`
12. Set `Email Template` to `Large Mileage Notification`.
13. Set the `Recipient Type` to `User` and in `Selected Recipient` select the `Bob Smith` user that you created earlier.
14. Click **Save**.
15. On the Specify Workflow Actions page, click **Done**.
16. On the Workflow Rule page, click **Activate**. A common mistake is forgetting to activate the workflow rule; if your rule is not active, it is not evaluated when records are created or saved.

Step 4: Test the Workflow

In this step, you test the workflow that you just created.

1. Click the Mileage tab.
2. Click **New** to create a new Mileage record.
3. Fill out the form completely, making sure you set `Miles` to be greater than 100.
4. Click **Save**. This should cause the workflow rule to trigger an email alert notifying the designated recipient that a Mileage record has been submitted with greater than 100 miles.

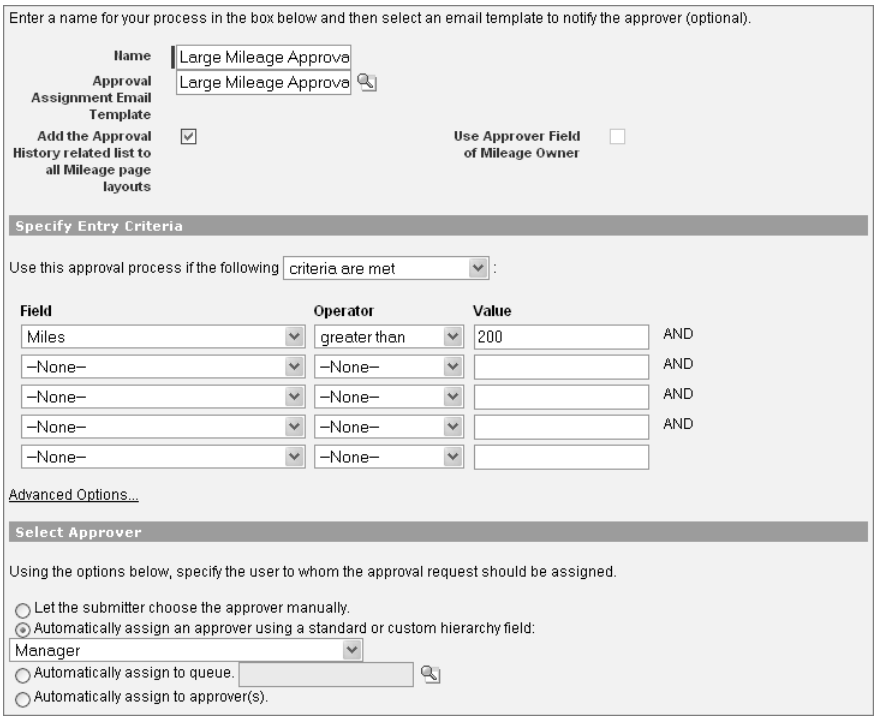
Step 5: Create an Approval Process

You have seen how powerful a workflow rule can be and, at the same time, simple to create. The workflow rule created in step 3 notified the manager of a high mileage entry. In this step you create an approval process that requires explicit approval from the manager to continue the mileage reimbursement. Creating and using an approval process is just as simple as creating a workflow rule. First, you need to create a field to indicate the approval state for a Mileage record.

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Mileage**.
3. On the Custom Object page, click **New** in the Custom Fields & Relationships section.
4. In Step 1 of the New Custom Field wizard, select `Picklist` for the data type and click **Next** to continue to Step 2.
5. Set `Field Label` to `Approval Status`.
6. For the picklist values, enter `Pending`, `Approved`, and `Rejected`.
7. Leave the remaining defaults and click **Next**.
8. In Step 3 of the New Custom Field wizard, select the `Read-Only` checkbox so that the field is read-only for all profiles, then click **Next**.
9. In Step 4 of the New Custom Field wizard, accept the defaults and click **Save**.

Now, create an approval process where a mileage value greater than 200 requires approval by the manager.

1. Click **Setup ➤ Create ➤ Workflow & Approvals ➤ Approval Processes**.
2. In the `Manage Approval Processes For` drop-down list, choose Mileage.
3. Click **Create New Approval Process** and choose **Use Jump Start Wizard** from the drop-down button.
4. Set `Name` to `Large Mileage Approval Process`.
5. Set `Approval Assignment Email Template` to the `Large Mileage Approval Notification` email template.
6. Set `Criteria` to `Miles greater than 200`.
7. Select `Automatically assign an approver using a standard or custom hierarchy field` and choose `Manager` for the hierarchy field.



8. Click **Save**.

The approval process has been created, but before it is usable, you must first define what happens to records upon initial submission, upon approval, and upon rejection.

1. Navigate to the approval process you just created. If you see the Jump Start Wizard confirmation page, click **View Approval Process Detail Page**.
2. Create new field update actions by clicking **Add New** and selecting **Field Update** for each of the sections specified in the following table. Configure each field update action as shown.

Section	Name	Field to Update	Picklist Options
Initial Submission Actions	`Set Initial Approval Status`	`Approval Status`	Choose `A specific value` and select `Pending`.
Final Approval Actions	`Set Final Approval Status`	`Approval Status`	Choose `A specific value` and select `Approved`.
Final Rejection Actions	`Set Final Rejection Status`	`Approval Status`	Choose `A specific value` and select `Rejected`.

3. After configuring each field update action, click **Save**.
4. Click **Approval processes** to go back to the approval list.
5. Click **Activate** to activate the approval process.

Step 6: Test the Approval Process

Before testing the approval process, make sure that your home page can display items that require approval.

1. Click **Setup ➤ Customize ➤ Home ➤ Home Page Layouts**.
2. Click **Edit** next to your home page layout.
3. Select the `Items to Approve` option.
4. Click **Save**.

While developing the application, you have been logged in as an administrator. To create a Mileage record and test the approval process, log out of your administrator account and log back in as a standard user.

1. Log in as the Bob Smith user you created earlier.
2. Click the Mileage tab.
3. Click **Create** and create a mileage entry with miles greater than 200.
4. Click **Save**.
5. Log out of the application.

An email is sent to Bob Smith's designated manager—your Developer Edition administrator account—notifying the manager that an approval request has been sent. To verify this, check your email. When you log in to the application using your administrator user, you see the pending approval request on the Home tab. The record is locked until the manager approves the mileage. An approval field allows the manager to approve or deny the mileage and continue the workflow.

Note: Notice that the `Approval Status` field is read only. In a real-world application, you hide these fields using Visualforce until they are required, since the page edit always shows them.

Summary

Once the initial setup of creating a user and email template is complete, setting up a workflow rule and an approval process is easy. The workflow rule runs whenever a Mileage record with greater than 100 miles is entered, which sends an email alert to the designated recipient. The approval process runs whenever a Mileage record with greater than 200 miles is entered, which routes the approval request to the appropriate manager.

Workflow rules and approval processes help you automate your business processes and enforce your standards. No more inflated mileage reports!

Tutorial #5: Adding Business Logic with Apex

Level: Advanced; **Duration:** 30 minutes

Apex is a is a strongly-typed, object-oriented, Java-like programming language that runs on salesforce.com servers. With Apex you have the flexibility to implement complex and sophisticated logic that is driven by changes in the data, so that you can create robust applications, such as inventory checking and order fulfillment, that run as a service.

In this tutorial, you will add business logic to the Mileage Tracker application built in Tutorial #2: Creating an App with the Force.com IDE on page 9. This code checks any values the user enters into the `Mileage` field to enforce a mileage limit. To enforce this validation, you will create an Apex class to hold the logic for the validation, as well as a trigger that calls on that logic whenever the user tries to save the record.

Prerequisites

Basic Knowledge

For this tutorial, it helps to have basic knowledge of Java and the Force.com API, but it is not required.

Software Requirements

- Eclipse 3.3: http://wiki.apexdevnet.com/index.php/Force.com_IDE_Installation_for_Eclipse_3.3.x
- Force.com IDE

Mileage Tracker App

You first need to create the basic Mileage Tracker application as described in Tutorial #1: Creating an App with Point-and-Click Tools on page 3 or Tutorial #2: Creating an App with the Force.com IDE on page 9.

Step 1: Create an Apex Class

Apex supports the concept of classes to organize and expose Apex methods and variables. In this step, you will create a class to contain the logic that determines whether a new mileage report can be created. This logic limits users to filing reports with a maximum of 500 miles for any given day, even across multiple mileage reports.

1. Create a new Apex class in the Force.com IDE:
   a. In the IDE, right-click your project folder and select **New ➤ Apex Class**.
   b. On the Create Apex Class page, enter `MileageUtil` in the **Name** field.
   c. Tab out of this field and accept the default values for **Version** and **Template**, and then click **Finish** to create the class.
2. Code the class logic:
   a. Declare a static constant called `MAX_MILES_PER_DAY` that holds the limit for daily mileage.

```
public class MileageUtil {
  static final Integer MAX_MILES_PER_DAY = 500;
```

   b. Next, declare a method called `areMilesAllowed` that takes an array of mileage records as an argument.

```
public class MileageUtil {
  static final Integer MAX_MILES_PER_DAY = 500;

  public static void areMilesAllowed(Mileage__c[] miles) {
  }
```

c. Then retrieve the `UserId` of the current user and save it in a string variable.

```
public static void areMilesAllowed(Mileage__c[] miles) {
  String createdbyId = UserInfo.getUserId();
}
```

Note: `UserInfo` is a built-in type that supplies global context information about the current user.

d. Run a query to return the mileage reports already submitted by the current user for today's travel (the default) and sum up the miles into the `totalMiles` variable. This code uses a `for` loop based on the results of the SOQL statement. For each record retrieved by the query, the loop adds the associated miles to the `totalMiles` variable.

```
public static void areMilesAllowed(Mileage__c[] miles) {
  String createdbyId = UserInfo.getUserId();

  Double totalMiles = 0;

  /* adds miles that were created in previous requests for today */
  for(Mileage__c mq:[SELECT miles__c FROM Mileage__c
          WHERE Date__c = TODAY
          AND Mileage__c.createdById = :createdbyId
          AND miles__c != null]) {
    totalMiles += mq.miles__c;
  }
```

3. The final piece of code adds the miles from the current record to those of any others for that day. It also determines whether the total is greater than the `MAX_MILES_PER_DAY` static variable you defined previously, at the class level. If that total is greater, the code prevents the operation by creating an error with an appropriate error message.

```
public static void areMilesAllowed(Mileage__c[] miles) {
    Double totalMiles = 0;
    String createdbyId = UserInfo.getUserId();

    /* Adds the miles created in previous requests for today */
    for(Mileage__c mq:[select miles__c from Mileage__c
            WHERE Date__c = TODAY
            AND Mileage__c.createdById = :createdbyId
            AND miles__c != null]) {
               totalMiles += mq.miles__c;
            }

    /* Totals the miles in the request */
    for (Mileage__c m:miles) {
      totalMiles += m.miles__c;
      if(totalMiles > MAX_MILES_PER_DAY)
        m.addError('Mileage request exceeds daily limit: ' + MAX_MILES_PER_DAY);
    }
}
```

4. Verify that the class looks like the following code:

```
public class MileageUtil {
  static final Integer MAX_MILES_PER_DAY = 500;

  public static void areMilesAllowed(Mileage__c[] miles) {
    Double totalMiles = 0;
    String createdbyId = UserInfo.getUserId();
```

```
    /* Adds the miles created in previous requests for today */
    for(Mileage__c mq:[SELECT miles__c FROM Mileage__c
            WHERE Date__c = TODAY
            AND Mileage__C.createdById = :createdbyId
            AND miles__c != null]) {
      totalMiles += mq.miles__c;
    }

    /* Totals the miles in the request */
    for (Mileage__c m:miles) {
      totalMiles += m.miles__c;
      if(totalMiles > MAX_MILES_PER_DAY)
        m.addError('Mileage request exceeds daily limit: ' + MAX_MILES_PER_DAY);
    }
  }
}
```

5. Click **File ➤ Save**.

Step 2: Create an Apex Trigger

Apex triggers are fired in response to data actions, such as inserts, updates and deletes, either before or after one of these events. Your trigger will implement the logic written in the Apex class, which calls the method in the class before an insert or update event.

1. Create a new trigger for the Mileage object.
   a. In the IDE, right-click your project folder and select **New ➤ Apex Trigger**.
   b. On the Create New Apex Trigger page, specify the following:
      - `Name`: `MileageTrigger`
      - `Version`: `14.0`
      - `Trigger Object`: `Mileage__c`
      - `Apex Trigger Operations`: **before insert**, **before update**
   c. Click **Finish**. You have now created a trigger that runs each time a Mileage record is saved, either as a new record (insert) or as an update of an existing record. The trigger does not yet do anything.

```
trigger MileageTrigger on Mileage__c (before insert, before update) {
}
```

2. Add code to the trigger.
   a. In the IDE, type the second line of code below. This line calls the method you created in the previous step, passing the current record values to the method.

```
trigger MileageTrigger on Mileage__c (before insert, before update) {
     MileageUtil.areMilesAllowed(Trigger.new);
}
```

   b. Save the file to compile the trigger and report any errors.
3. Make sure the trigger is activated.
   a. With the `MileageTrigger.trigger` file open, switch from the **Source** tab to the **Metadata** tab.
   b. Verify that the value of the `<active>` element is set to **true**.
   c. Save the file.

Note: If the trigger does not compile for any reason, a red X appears next to the line of code that contains the error. To determine what the problem is, hover your pointer over the marker and a popup identifies the error. You can also open the Problems view with **Window ➤ Show View ➤ Problems**.

Step 3: Test the Trigger

Now test the trigger in the app:

1. In your browser, go to the Mileage tab.
2. Create a new Mileage record. For the Miles value, enter `200`, accept the default for the Date value, and enter `Tim Barr` as the Contact.
3. Click **Save and New** to save the record and display a blank screen for another record.
4. Create another record with the same values and actions listed above, to bring the total miles for the day to 400.
5. Repeat Step 4 to bring the total miles to over 500 and click **Save**. You should get an object-level error message, indicating a problem with the save action, and a field-level error message, with the text you specified in the previous step.
6. Fix the error and click **Save**.

Summary

In this tutorial, you built on the Mileage Tracker application you created previously by adding business logic that detects and prevents the entry of excess miles. To do this, you defined a `MileageUtil` Apex class to perform the actual validation, and a `MileageTrigger` trigger that executes whenever a Mileage record is saved. In the process, you saw how the Force.com IDE can be useful in creating, compiling and debugging Apex.

Tutorial #6: Adding Tests to Your Application

Level: Advanced; **Duration:** 45-60 minutes

One of the important steps in developing any application is testing. Testing enables developers to programmatically validate the code behavior and expected results.

In the last few tutorials, you've focused on building the mileage tracking application. In Tutorial #5: Adding Business Logic with Apex on page 29, you added code to the Mileage Tracker application to enforce a mileage limit. To verify this code, you executed a few manual tests through the online user interface and confirmed the behavior of the application. But what if you want to programmatically test this behavior and provide automated code coverage for longer term development? The Apex test framework provides key tools to help you emulate and validate application behavior as well as automate these test processes.

Prerequisites

"Developer Mode," "Modify All Data," and "Author Apex" permissions
Since this tutorial involves working with Apex, ensure that you have the proper permissions to create Apex classes and configure Apex email services.

Note: Your Developer Edition Administration User has these permissions by default.

Mileage Tracker App
You first need to create the basic Mileage Tracker application as described in Tutorial #1: Creating an App with Point-and-Click Tools on page 3 or Tutorial #2: Creating an App with the Force.com IDE on page 9.

Software Requirements

- Eclipse 3.3: http://wiki.apexdevnet.com/index.php/Force.com_IDE_Installation_for_Eclipse_3.3.x
- Force.com IDE

Step 1: Create an Apex Class

All unit tests are contained in Apex classes. In this step you are creating the class to contain the unit tests.

1. Create a new Apex class in the Force.com IDE:
   a. In the IDE, right-click your project folder and select **New ➤ Apex Class**.
   b. On the Create Apex Class page, specify `MileageTrackerTestSuite` for the name.
   c. In the **Template** field, choose **Test Class**. This creates a class with the `@isTest` annotation, which tells Force.com that this class is for test purposes only. The code in this class will not count against the total 1 MB Apex code limit for your organization.
   d. Click **Finish** to create the class.
2. Start coding the class logic:
   a. The test class template creates a method defined as `testMethod`, which declares the method as a *unit test*. 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 defined using the `testMethod` keyword. Test methods must always be declared as `static`. The method produced by the template is called `myUnitTest`. Rename this method as

`runPositiveTestCases`. Because there is no access modifier on this method, by default, it is `private`. Your method should look like the following:

```
static testMethod void runPostiveTestCases() {
    // TO DO: implement unit test
```

b. Delete the generated comment inside the method and declare the following method variables: `totalMiles`, `maxTotalMiles`, `singletotalMiles`, `createdbyId` and `deleteMiles`.

```
Double totalMiles = 0;
final Double maxtotalMiles= 500;
final Double singletotalMiles= 300;
final String createdbyId = UserInfo.getUserId();
List<Mileage__c> deleteMiles = new List<Mileage__c>();
```

Step 2: Clean up Existing Data

If you run this test more than once in a single day, you want to make sure that no data from a previous test remains. Add the following code to the end of the existing class:

1. Add text to the debug log, indicating the next step of the script:

```
System.debug('Setting up testing - deleting any mileage records for today');
```

2. Delete any data created by this user if found. This assumes that you are always signing onto Salesforce with the same user:

```
deleteMiles = [SELECT miles__c from Mileage__c
   WHERE createdDate = TODAY and createdById = :createdbyId];
   if(!deleteMiles.isEmpty()) {
      delete deleteMiles;
   }
```

Step 3: Add Tests for the Positive Case

Now let's add tests to the existing class for the positive case, that is to verify that the code behaves as expected. First test for inserting a single record, then test for a bulk record insert. Add the following code to the end of the existing `MileageTrackerTestSuite` class:

1. First, add text to the debug log, indicating the next step of the script:

```
System.debug('Inserting 300 more miles...single record validation');
```

2. Then create a Mileage__c object and insert it into the database.

```
Mileage__c testMiles1 = new Mileage__c(Miles__c = 300, Date__c = System.today() );
insert testMiles1;
```

3. Validate the code by returning the inserted records:

```
for(Mileage__c m:[SELECT miles__c FROM Mileage__c
   WHERE createdDate = TODAY
   and createdById = :createdbyId
   and miles__c != null]) {
```

```
        totalMiles += m.miles__c;
    }
```

4. Use the `system.assertEquals` method to verify that the expected result is returned:

```
System.assertEquals(singletotalMiles, totalMiles);
```

5. Before moving to the next test, set the number of total miles back to 0:

```
totalMiles = 0;
```

6. Validate the code by creating a bulk insert of 200 records.

   First, add text to the debug log, indicating the next step of the script:

```
System.debug('Inserting 200 Mileage records...bulk validation');
```

7. Then insert 200 Mileage__c records:

```
List<Mileage__c> testMiles2 = new List<Mileage__c>();
for(integer i=0; i<200; i++){
testMiles2.add( new Mileage__c(Miles__c = 1, Date__c = System.today()) );
   }
insert testMiles2;
```

8. Use `System.assertEquals` to verify that the expected result is returned:

```
for(Mileage__c m:[SELECT miles__c FROM Mileage__c
   WHERE createdDate = TODAY
   and createdById = :createdbyId
   and miles__c != null]) {
       totalMiles += m.miles__c;
    }
       System.assertEquals(maxtotalMiles, totalMiles);
```

Step 4: Add Negative Test Cases

Add a test for negative cases by confirming the validation behavior.

1. Create a static test method called `runNegativeTestCases`:

```
static testMethod void runNegativeTestCases(){
```

2. Add text to the debug log, indicating the next step of the script:

```
System.debug('Inserting 501 miles... negative test case');
```

3. Create a Mileage__c record with 501 miles.

```
Mileage__c testMiles3 = new Mileage__c(Miles__c = 501, Date__c = System.today());
```

4. Place the `insert` statement within a `try/catch` block. This allows you to catch the validation exception and assert the generated error message.

```
try {
     insert testMiles3;
     } catch (DmlException e) {
```

5. Now use the `System.assert` and `System.assertEquals` to do the testing. Add the following code to the `catch` block you previously created:

```
//Assert Error Message
   System.assert(e.getMessage().contains('Insert failed. First exception on row 0; '+
      'first error: FIELD_CUSTOM_VALIDATION_EXCEPTION, '+
      'Mileage request exceeds daily limit(500): [Miles__c]'),
         e.getMessage());

//Assert Field
   System.assertEquals(Mileage__c.Miles__c, e.getDmlFields(0)[0]);

//Assert Status Code
   System.assertEquals('FIELD_CUSTOM_VALIDATION_EXCEPTION'  , e.getDmlStatusCode(0));
          }
   }
}
```

6. Verify that the class looks like the following code:

```
@isTest
private class MileageTrackerTestSuite {

   static TestMethod void runPostiveTestCases(){

   Double totalMiles = 0;
   final Double maxtotalMiles= 500;
   final Double singletotalMiles= 300;
   final String createdbyId = UserInfo.getUserId();
   List<Mileage__c> deleteMiles = new List<Mileage__c>();

// Data clean-up
   System.debug('Setting up testing - deleting any mileage records for today');
   deleteMiles = [select miles__c from Mileage__c where createdDate = TODAY
     and createdById = :createdbyId];
   if(!deleteMiles.isEmpty()){
      delete deleteMiles;
   }

// Positive tests
    System.debug('Inserting 300 miles...');
    Mileage__c testMiles1 = new Mileage__c(Miles__c = 300, Date__c = System.today() );
    insert testMiles1;

// Validate single insert
     for(Mileage__c m:[SELECT miles__c FROM  Mileage__c
         WHERE createdDate = TODAY
         and createdById = :createdbyId
         and miles__c != null]) {
             totalMiles += m.miles__c;
         }
       System.assertEquals(singletotalMiles, totalMiles);
       totalMiles = 0;

// Validate bulk insert
    System.debug('Inserting 200 more miles...bulk validation');
```

```
    List<Mileage__c> testMiles2 = new List<Mileage__c>();
    for(integer i=0; i<200; i++){
       testMiles2.add( new Mileage__c(Miles__c = 1, Date__c = System.today()) );
    }
    insert testMiles2;

    // Assert Mileage
    for(Mileage__c m:[SELECT miles__c FROM Mileage__c
      WHERE createdDate = TODAY
      and createdById = :createdbyId
      and miles__c != null]) {
         totalMiles += m.miles__c;
      }
       System.assertEquals(maxtotalMiles, totalMiles);

   }

// Negative tests
   static testMethod void runNegativeTestCases(){
     System.debug('Inserting 501 miles... negative test case');
     Mileage__c testMiles3 = new Mileage__c(Miles__c = 501, Date__c = System.today());
   // Assert Error Message
    try {
         insert testMiles3;
         } catch (DmlException e) {
                // Assert Error Message
                System.assert(e.getMessage().contains('Insert failed. '+
                'First exception on row 0; first error: ' +
                'FIELD_CUSTOM_VALIDATION_EXCEPTION, '+
                'Mileage request exceeds daily limit(500): [Miles__c]'),
                e.getMessage());
                // Assert Field
                System.assertEquals(Mileage__c.Miles__c, e.getDmlFields(0)[0]);
                // Assert Status Code
                System.assertEquals('FIELD_CUSTOM_VALIDATION_EXCEPTION',
                    e.getDmlStatusCode(0));
              }
   }
}
```

7. Click **File ➤ Save**. This saves the class locally, and if the Force.com IDE is in online mode, to the server as well.

Note: You cannot run any tests until the `MileageTrackerTestSuite` and `MileageUtil` classes have been saved to the server. In addition, the Mileage__c object must also be saved to the server.

Step 5: Run the Apex Test Class in the IDE

Now that you've created your test class, you can run it. The Force.com IDE provides you with an easy way to run your tests, providing you with detailed information.

If possible, you should have 100% of your Apex code should be covered with test cases, and every trigger should have some code coverage. At least 75% of your code must be covered in order to deploy your code from a development organization to a production organization.

To run your test class:

1. Click on the Apex Test Runner tab.
2. Select Apex Code as the `Log category` and Finest as the `Log level` (slide the level all the way to the right).

Note: If the controls on the Apex Test Runner tab are not enabled, in the IDE, right click and select **Force.com ➤ Run Tests**. After you run the tests once, the controls are enabled.

3. In the IDE, select the `MileageTrackerTestSuite` class, then right click and select **Force.com ➤ Run Tests**.

Note: If you have other methods or classes defined as tests in your organization, they also are run.

4. In the Code Coverage Results, you should see a green check mark next to the `MileageTrackerTestSuite` class, indicating that 0 lines of the `MileageTrackerTestSuite` class are not tested, and that 100% of your code in that class is covered by tests.
5. To the right, in the Apex Test Runner tab, is the debug log.
   - The first part of the log details the events that occurred while running the `runNegativeTestCases` method in the `MileageTrackerTestSuite` class.

     ```
     Debug Log:

     *** Beginning Test 1: MileageTrackerTestSuite.private static testMethod void
     runNegativeTestCases()
     ```

   - This line in the debug log

     ```
     20080924214612.541:Class.MileageTrackerTestSuite.runNegativeTestCases: line 47, column
      5: Inserting 501 miles... negative test case
     ```

     occurs as a result of this line in the `MileageTrackerTestSuite` class:

     ```
     System.debug('Inserting 501 miles... negative test case');
     ```

   - The next few lines in the debug log give details on how long it took to execute specific lines of code. This can be very useful when doing performance tuning:

     ```
     20080924214612.541:Class.MileageTrackerTestSuite.runNegativeTestCases: line 51, column
      10: Insert: SOBJECT:Mileage__c
     20080924214612.541:Class.MileageTrackerTestSuite.runNegativeTestCases: line 51, column
      10:     DML Operation executed in 331 ms
     20080924214612.541:Class.MileageTrackerTestSuite: line 46, column 25:     returning
     from end of method static testMethod void runNegativeTestCases() in 332 ms
     ```

   - The Apex runtime engine keeps track of the number of resources 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 will receive an error message. The debug log lists how many resources a program uses, as well as the total amount available for that resource.

     ```
     Cumulative resource usage:

     Resource usage for namespace: (default)
     Number of SOQL queries: 0 out of 100
     Number of query rows: 0 out of 500
     Number of SOSL queries: 0 out of 20
     Number of DML statements: 1 out of 100
     Number of DML rows: 1 out of 500
     Number of script statements: 3 out of 200000
     Maximum heap size: 0 out of 500000
     Number of callouts: 0 out of 10
     ```

```
Number of Email Invocations: 0 out of 10
Number of fields describes: 0 out of 10
Number of record type describes: 0 out of 10
Number of child relationships describes: 0 out of 10
Number of picklist describes: 0 out of 10
Number of future calls: 0 out of 10
Number of find similar calls: 0 out of 10
Number of System.runAs() invocations: 0 out of 20

Total email recipients queued to be sent : 0
*** Ending Test MileageTrackerTestSuite.static testMethod void runNegativeTestCases()
```

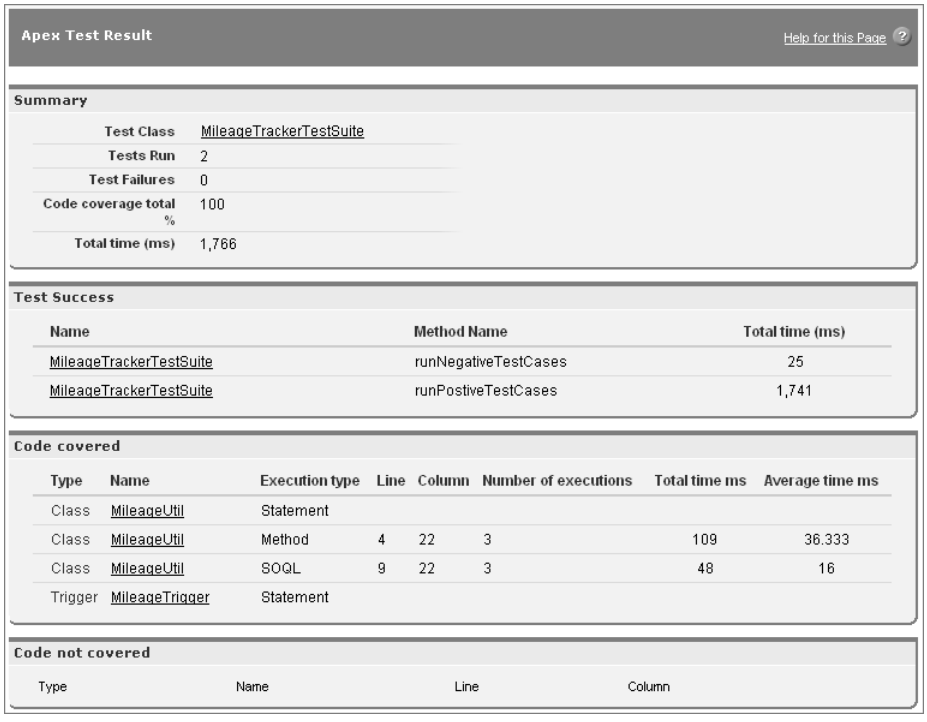
- The debug log then lists the same statistics for the other method in the class, `runPostiveTestCases`.

Step 6: Run the Test Class in the Online User Interface

In addition to running test classes in the Force.com IDE, you can also run them in the online user interface. The debug log produced by the online user interface includes the same information, just presented in a different way. Also, the online user interface gives you the ability to run tests in a single class. If you use the IDE, every test in your organization is run.

Note: You can also run all the tests in your organization using the online user interface.

1. In the online user interface, click **Setup ➤ Develop ➤ Apex Classes**, then click the name of the class, `MileageTrackerTestSuite`.
2. Click **Run Test**.
3. The Apex Test Result page displays:



Apex Test Result — Help for this Page

Summary

Test Class	MileageTrackerTestSuite
Tests Run	2
Test Failures	0
Code coverage total %	100
Total time (ms)	1,766

Test Success

Name	Method Name	Total time (ms)
MileageTrackerTestSuite	runNegativeTestCases	25
MileageTrackerTestSuite	runPostiveTestCases	1,741

Code covered

Type	Name	Execution type	Line	Column	Number of executions	Total time ms	Average time ms
Class	MileageUtil	Statement					
Class	MileageUtil	Method	4	22	3	109	36.333
Class	MileageUtil	SOQL	9	22	3	48	16
Trigger	MileageTrigger	Statement					

Code not covered

Type	Name	Line	Column

Note the `Code covered` and `Code not covered` sections. These are used when a class runs tests on a trigger or another class. For example, if the test class `MileageTrackerTestSuite` also contained a method that explicitly tested the `MileageUtil` class, these sections would contain data.

The debug log at the bottom of the page contains the same information as the debug log produced by the IDE.

Summary

In this tutorial, you built on the Mileage Tracker application you previously created. You created `MileageTrackerTestSuite`, an Apex class defined as a test class to test the application, then added code to clean up after any previous tests. Then you added positive test cases to verify the expected behavior. You made sure the test class works for both bulk insert as well as for a single record. You also tested to verify the negative test case, to ensure that errors are handled correctly. You ran the tests in the Force.com IDE as well as the online user interface, and saw how useful the debug log could be for performance tuning as well as avoiding resource limits.

Tutorial #7: Building a Custom User Interface Using Visualforce

Level: Intermediate; **Duration:** 30 minutes

Using Visualforce you can build and deliver any user experience as a service, with all the power of modern user interface development and modeling technologies. Visualforce combines full HTML and JavaScript support with a reusable component library and an Apex-based controller model, so that the user interfaces you create are not only highly flexible, but also well architected and easily maintainable.

In this tutorial, you will use Visualforce to extend the standard interface of the Mileage Tracker application built in Tutorial #2: Creating an App with the Force.com IDE on page 9. You will also use the Force.com IDE to develop a new extension to a standard controller.

Prerequisites

Basic Knowledge

For this tutorial, it helps to have basic knowledge of Java and the Force.com API, but it is not required.

Software Requirements

- Eclipse 3.3: http://wiki.apexdevnet.com/index.php/Force.com_IDE_Installation_for_Eclipse_3.3.x
- Force.com IDE

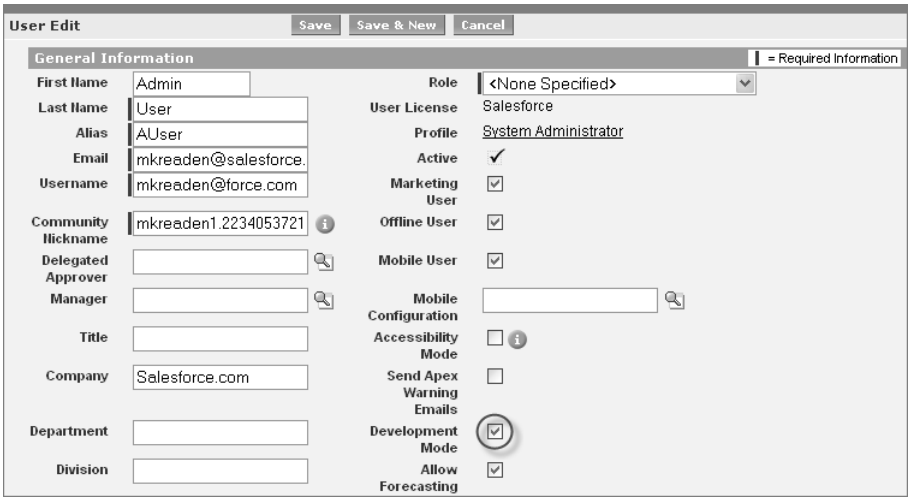
Getting Started

If you have already created the Mileage Tracker application or downloaded it in Tutorial #10: Downloading and Deploying an App Using Code Share on page 57, you must first delete all the components you created—including the custom object, the tab, and the application—so that the names don't conflict with those in this tutorial.

Step 1: Enable Visualforce Developer Mode

Development Mode helps you create and edit Visualforce pages.

1. Log in to your Developer Edition account.
2. Click **Setup ➤ My Personal Information ➤ Personal Information**.
3. Click **Edit**.
4. Select the checkbox for `Development Mode` and click **Save**.



Step 2: Create a Hello World Page

In this step, you will create a basic Hello World Visualforce page.

1. Create a new Visualforce page.
   a. In your browser, add the text `/apex/MyMileagePage` to the URL for your Salesforce instance. For example, if your Salesforce instance is `https://na1.salesforce.com`, the new URL would be `https://na1.salesforce.com/apex/MyMileagePage`. You will get an error message: Page MyMileagePage Does Not Exist.

   b. Click the **Create page MyMileagePage** link to create the new page.
   c. Click the Page Editor link in the bottom left corner of the page. This displays the code for the new page, which should look like this:

```
<apex:page>
<!-- Begin Default Content REMOVE THIS -->
<h1>Congratulations</h1>
This is your new Page: MyMileagePage
<!-- End Default Content REMOVE THIS -->
</apex:page>
```

2. Use the Page Editor.
   a. Note that the existing code looks a lot like standard HTML. That's because a Visualforce page combines HTML tags, such as `<h1>`, with Visualforce-specific tags, which all start with `<apex:>`
   b. Change the code to match the following:

```
<apex:page>
<h1>My Mileage Page</h1>
</apex:page>
```

   c. Click **Save** at the top of the Page editor. The page refreshes to reflect your changes.

3. You can also use variables to access information on the Force.com environment, including data in your organization. Modify the code to display your name in the heading, using the global variable `$User` and the data field `FirstName`, so that the Visualforce page code looks like the following code:

```
<apex:page>
<h1>{!$User.FirstName}'s Mileage Perspective </h1>
</apex:page>
```

4. Save the page and see the changes you just made.

Step 3: Create a Custom View for a Mileage Record

The Visualforce page you created is the visual component of Visualforce. To access data in your organization, the Visualforce page typically uses a *controller* to handle access to the data. All Force.com objects have default standard controllers that can be used to interact with the data associated with the object. In this step, we will reference a controller to access data in the Mileage custom object created in Tutorial #2: Creating an App with the Force.com IDE on page 9.

1. Modify your code to enable the Mileage__c standard controller. This controller will be used to access the data in the custom Mileage object.

```
<apex:page standardController="Mileage__c">
    <h1>{!$User.FirstName}'s Mileage Page</h1>
</apex:page>
```

2. Use the `apex:detail` component to display the record's detail view. This standard view is delivered by the standard controller, which requires the record ID for an existing record as part of the URL. Note that when you use the Page Editor, there is content assistance for tags.

```
<apex:page standardController="Mileage__c">
    <h1>{!$User.FirstName}'s Mileage Page</h1>
    <apex:detail></apex:detail>
</apex:page>
```

   a. Click **Save**.
   b. Locate an existing Mileage record (or, if you have not created any records, create a new record now).
   c. Copy the record ID from the Mileage record. The record ID is the 15-character string following the Salesforce instance portion of the URL. For example, if the URL for your Salesforce instance is `https://na1.salesforce.com`, the complete URL for the display of a Mileage record could be `https://na1.salesforce.com/a0070000009c3QQ`
   d. Add the record's ID into the URL for your Visualforce page, prefixed with `?id=`. For example, if the URL for your Visualforce page is `https://na1.salesforce.com/apex/MyMileagePage` and the record ID is `a0070000009c3QQ`, the completed URL would be `https://na1.salesforce.com/apex/MyMileagePage?id=a0070000009c3QQ`.
   e. Press Enter. As you can see, this simple tag brought a lot of information and functionality with it.

3. Display the details of a related record:

   a. Modify your code to display the details of the associated Contact record.

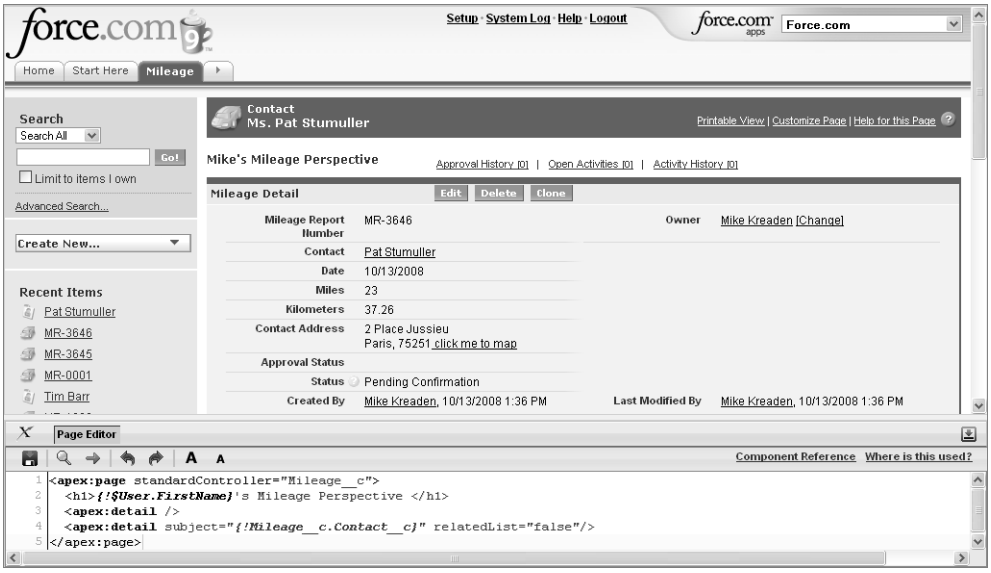
```
<apex:page standardController="Mileage__c">
    <h1>{!$User.FirstName}'s Mileage Perspective </h1>
    <apex:detail/>
    <apex:detail subject="{!Mileage__c.Contact__c}"/>
</apex:page>
```

   With this simple tag, you can display the Contact details, including the object's related records, below the Mileage record details. You can suppress these details with a simple attribute. If you don't see any detailed record, you have not associated a Contact with your Mileage record. The dot notation (`Mileage__c.Contact__c`) allows you to navigate the data model from one record to a related record. In this case, allowing you to display Contact information related to a Mileage record.

   b. Modify your code to turn off the Contact's related records.

```
<apex:page standardController="Mileage__c">
    <h1>{!$User.FirstName}'s Mileage Perspective </h1>
    <apex:detail/>
    <apex:detail subject="{!Mileage__c.Contact__c}" relatedList="false"/>
</apex:page>
```

This code displays a page that looks something like this:



Step 4: Extend the Mileage Controller

The standard controller gives your Visualforce page basic access to records for a Force.com object. You can extend the standard controller so that it acts only on specific records. In this step, you will extend the controller to return only Mileage records created today.

1. Make sure you are in the Page Editor for MyMileagePage.
2. Change the following code:

```
<apex:page standardController="Mileage__c">
    <h1>{!$User.FirstName}'s Mileage Page</h1>
    <apex:detail> </apex:detail>
    <apex:detail subject="{!Mileage__c.Contact__c}" relatedList="false"></apex:detail>
</apex:page>
```

to (add the part in bold):

```
<apex:page standardController="Mileage__c" extensions="MileageExtension">
    <h1>{!$User.FirstName}'s Mileage Page</h1>
    <apex:detail> </apex:detail>
    <apex:detail subject="{!Mileage__c.Contact__c}" relatedList="false"></apex:detail>
</apex:page>
```

3. Click the save icon. Above the page editor an error message displays, stating that the MileageExtension class doesn't exist.
4. Click the **Create Apex Class 'MileageExtension'** link that is part of the error message. You should get another error that the constructor MileageExtension is unknown.
5. Click the **Create Apex method 'MileageExtension.MileageExtension(ApexPages.StandardController controller)'**.
6. Switch to the Force.com IDE.

Note: Controllers can be edited in the same page as the Visualforce page. Extensions must be edited either in the Force.com IDE or the Salesforce user interface.

7. Expand the **mileageproj ➤ src ➤ classes** folder.
8. Add the new classes to your project. Right click the **classes** folder, and select **Force.com ➤ Refresh from Server**.
9. Double-click the **MileageExtension.cls** file to edit it.
10. Add an instance variable and a constructor to the existing empty class:

```
public class MileageExtension {
    private final Mileage__c mileageObj;

    public MileageExtension(ApexPages.StandardController controller) {
        this.mileageObj = (Mileage__c)controller.getSubject();
    }
}
```

11. Modify the extension to add a new method, `getTodaysMileageRecords()`. This method returns a list of Mileage__c records to a Visualforce page.

```
public class MileageExtension {
    private final Mileage__c mileageObj;
    public MileageExtension(ApexPages.StandardController controller) {
        this.mileageObj = (Mileage__c)controller.getRecord();
    }

    public Mileage__c[] getTodaysMileageRecords() {
        String createdbyId = UserInfo.getUserId();
        Mileage__c[] mileageList =
           [SELECT name, miles__c
            FROM Mileage__c
            WHERE Date__c = TODAY
            AND createdbyid = :createdbyId];
    return mileageList;
    }
}
```

The method gets the ID of the current user from the class `UserInfo` and then uses this information and `TODAY` in a SOQL query to return an array of the mileage records created today by the current user.

12. Save the file to compile the extension and view any errors.

Step 5: Display a Table of Data

The extension you created lets you retrieve a list of mileage records. You can use this list with the standard `dataTable` Visualforce component to display a table of data.

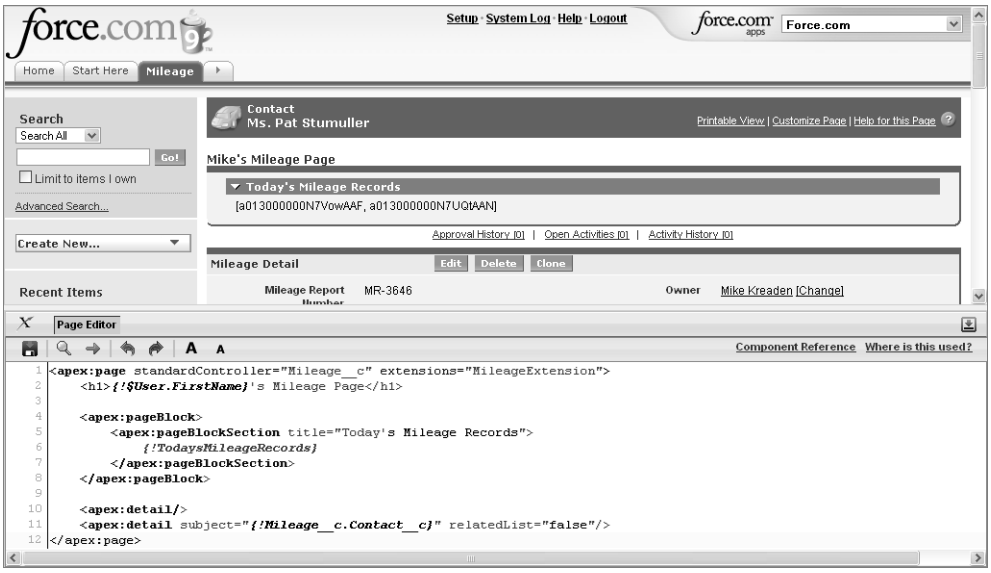
1. Use the `getTodaysMileageRecords` method. Return to the Page Editor for MyMileagePage and modify the page as shown below.

```
<apex:page standardController="Mileage__c" extensions="MileageExtension">
    <h1>{!$User.FirstName}'s Mileage Page</h1>

    <apex:pageBlock>
        <apex:pageBlockSection title="Today's Mileage Records">
            {!TodaysMileageRecords}
        </apex:pageBlockSection>
    </apex:pageBlock>

    <apex:detail/>
    <apex:detail subject="{!Mileage__c.Contact__c}" relatedList="false"/>
</apex:page>
```

In this code, you can see a few new things:

- The `<apex:pageBlock>` tag divide the screen display into a separate logical unit.
- The `<apex:pageBlockSection>` tag further divides the page and give you a way to add a title to the section.
- The `{!TodaysMileageRecords}` code binds to the `getTodaysMileageRecords` method in the controller extension to retrieve the data.

2. Click **Save**. Your output should resemble the following:



Note: The output displays what looks like one (or more) record IDs. These are the record IDs of the Mileage records that were returned by the `getTodaysMileageRecords` method defined in your controller extension.

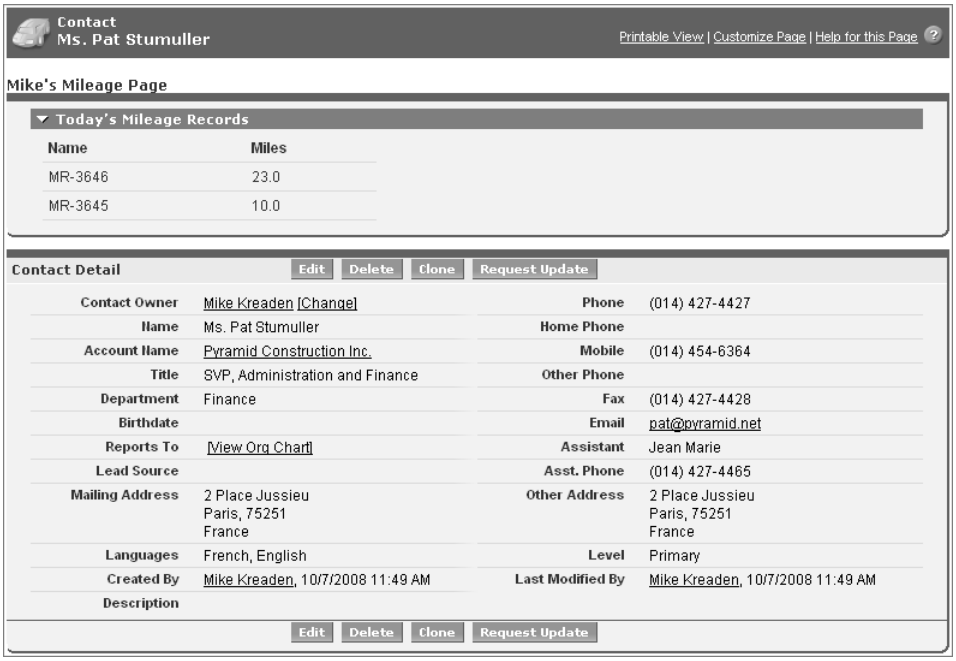
3. Display the records in a table format:

   a. Modify the page by replacing the `{!TodaysMileageRecords}` code with the code in bold below:

```
<apex:page standardController="Mileage__c" extensions="MileageExtension">
    <h1>{!$User.FirstName}'s Mileage Page</h1>
    <apex:pageBlock>
        <apex:pageBlockSection title="Today's Mileage Records">
            <apex:dataTable value="{!TodaysMileageRecords}"
                            var="mileage" styleClass="list">
                <apex:column>
                    <apex:facet name="header">Name</apex:facet>
                    <apex:outputText value="{!mileage.Name}"/>
                </apex:column>
                <apex:column>
                    <apex:facet name="header">Miles</apex:facet>
                    <apex:outputText value="{!mileage.Miles__c}"/>
                </apex:column>
            </apex:dataTable>
        </apex:pageBlockSection>
    </apex:pageBlock>
    <apex:detail subject="{!Mileage__c.Contact__c}" relatedList="false"/>
</apex:page>
```

   The `dataTable` component iterates through a collection of data that was returned by the `getTodaysMileageRecords` method in the controller extension you created. The `var` attribute of the `dataTable` contains the name (mileage) of

the variable that is used to reference fields in that collection in each column of the `dataTable`. Your final output should look something like the following:



Summary

Congratulations! You have created a new interface for your Mileage Tracker application by creating both a Visualforce page and an extension to a standard controller. Your page also uses powerful tags to rapidly implement the functionality of a standard Force.com application.

Tutorial #8: Create a Public Web Page Using Force.com Sites

Level: Intermediate; **Duration:** 30 minutes

Force.com Sites enables you to create public websites and applications that are directly integrated with your Force.com organization—without requiring users to log in with a username and password.

In this tutorial, you will register and enable Sites for your Developer Edition organization, create a Visualforce page, register your Force.com domain name, and expose the Visualforce page you created as a public Web page using Sites.

Prerequisites

Basic Knowledge
For this tutorial, it helps to have basic knowledge of Visualforce and Salesforce administration, but it is not required.

Step 1: Enable your Developer Edition Organization for Sites Developer Preview

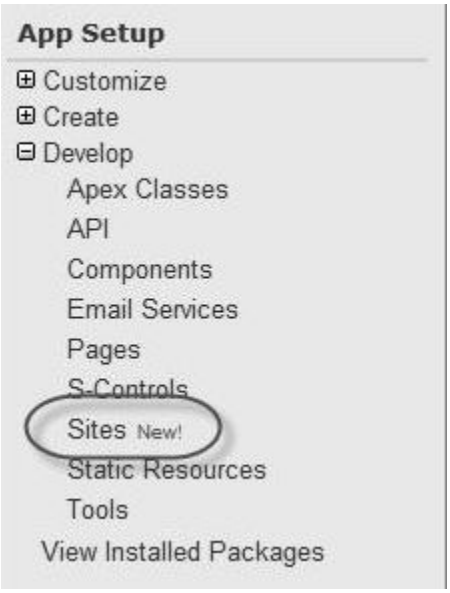
Force.com Sites is currently in developer preview. As such, functionality is subject to change.

To enable your Developer Edition organization for Sites Developer Preview:

1. Open a browser and navigate to http://developer.force.com/sitespreview.
2. Fill out the new member form or use the existing member shortcut.

In a few moments, Force.com Sites will be enabled for your Developer Edition organization.

You can verify that Force.com Sites has been enabled for your Developer Edition organization by clicking **Setup ➤ Develop**. You should see the link for **Sites**.



Step 2: Create a Visualforce Page

In this step, you create a Visualforce page to use as the home page for your site.

Note: Make sure that you have Developer Mode enabled. Follow the instructions in Step 1: Enable Visualforce Developer Mode on page 41 if you do not have Developer Mode enabled.

1. In your browser, add the text `/apex/MileagePolicy` to the URL for your Salesforce instance. For example, if your Salesforce instance is `https://na1.salesforce.com`, the new URL would be `https://na1.salesforce.com/apex/MileagePolicy`.

   You will get an error message: Page MileagePolicy Does Not Exist.

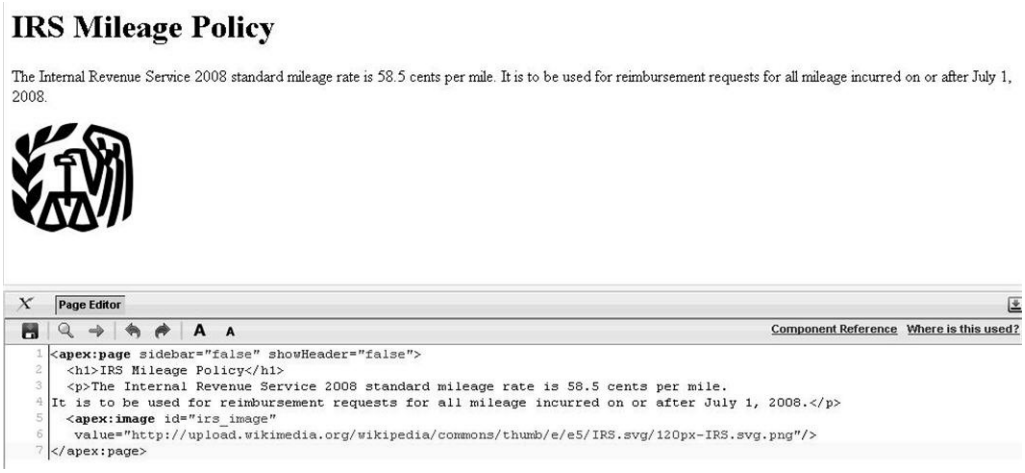
2. Click the **Create page MileagePolicy** link to create the new page.
3. Click the **Page Editor** link in the bottom left corner of the page. This displays the code for the new page, which should look like this:

```
<apex:page>
<!-- Begin Default Content REMOVE THIS -->
<h1>Congratulations</h1>
This is your new Page: MyMileagePage
<!-- End Default Content REMOVE THIS -->
</apex:page>
```

4. Replace the existing page markup with the following:

```
<apex:page sidebar="false" showHeader="false">
  <h1>IRS Mileage Policy</h1>
  <p>The Internal Revenue Service 2008 standard mileage rate is 58.5 cents per mile.
It is to be used for reimbursement requests for all mileage incurred on or after July 1,
 2008.</p>
  <apex:image id="irs_image"

value="http://upload.wikimedia.org/wikipedia/commons/thumb/e/e5/IRS.svg/120px-IRS.svg.png"/>
</apex:page>
```

5. Click the save icon at the top of the Page Editor. Your page should now look like this:



Step 3: Register a Force.com Domain Name

Your unique Force.com domain, which hosts your site, is constructed from the unique *domain prefix* that you register, plus `force.com`. For example, if you choose "mycompany" as your domain prefix, your domain name would be `http://www.mycompany.force.com`.

Note: The construction of the secure URLs for your Force.com sites depends on the type of organization they are built on. In the following examples, the domain prefix is "mycompany," the sandbox name is "mysandbox," the instance name is "na1," and the sandbox instance name is "cs1 ":

Organization Type	Secure URL
Developer Edition	`https://mycompany-developer-edition.na1.force.com`
Developer Sandbox	`https://mysandbox.mycompany.cs1.force.com`
Production	`https://mycompany.secure.force.com`

The domain prefix for Developer Edition must contain 22 characters or fewer. The secure URL is displayed on the Login Settings page.

To get started, register your company's Force.com domain by doing the following:

1. Click **Setup ➤ Develop ➤ Sites**.
2. Enter a unique name for your Force.com domain. The name is case-sensitive. Only alphanumeric characters are allowed. Salesforce.com recommends using your company's name or a variation, such as "mycompany."

   Caution: You cannot modify your Force.com domain name after you have registered it.

3. Click **Check Availability** to confirm that the domain name you entered is unique. If it is not unique, you are prompted to change it.
4. Read and accept the Force.com Sites Terms and Use by selecting the checkbox.
5. Click **Register My Force.com Domain**. After you accept the Terms and Use and register your Force.com domain, the change is tracked in your organization's setup audit trail.

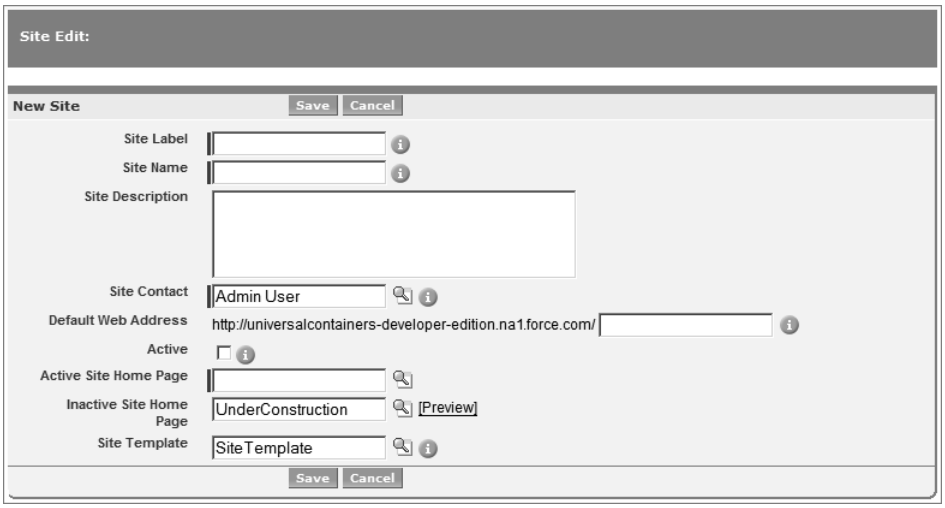
Congratulations! You are now ready to create your first Force.com site.

Note: Custom Web addresses registered through a domain name registrar, such as `www.mycompany.com`, are not supported in Developer Edition.

Step 4: Create a Force.com Site

Now that you have registered your domain, you can make the Visualforce page you created in Step 2: Create a Visualforce Page the home page for your new site.

1. Go to the Sites page by clicking **Setup ➤ Develop ➤ Sites**.
2. Click **New**. The Site Edit page displays:



3. On the Site Edit page, fill in the site details:
   a. For the `Site Label`, enter `Mileage Policy`.
   b. For the `Site Name`, enter `Mileage_Policy`.
   c. For the `Active Site Home Page`, enter `MileagePolicy`.
4. Check the `Active` checkbox.
5. Click **Save**.

Step 5: Test the Site

Now that you've created your site, you want to make sure that you can access it when you are not logged into your Developer Edition organization.

1. Go to the listing of Sites by clicking **Setup ➤ Develop ➤ Sites**.
2. Click the `Site URL` link for the Mileage Policy site. It should look something like this:
   `http://nick-developer-edition.na6.force.com/`

   This opens either a new tab or a new window (depending on your browser).
3. In the original tab or window, log out of Salesforce.
4. Go back to the Mileage Policy page and click the refresh icon.

   Notice that even though you are no longer logged in to Salesforce, you still have access to this page.

Summary

Congratulations! You have now created a Visualforce page that can be accessed publicly without requiring the user to log in. To create the site, you enabled your Developer Edition org to use the Sites preview functionality, then created a Visualforce page as your home page. Then you registered a unique Force.com domain name, linked it to your home page, and tested the site to see that it works.

Tutorial #9: Adding Email Services

Level: Advanced; **Duration:** 20-30 minutes

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

In this tutorial, you will create an email service that updates the Mileage Tracker application. The Apex logic that you develop will create a new Mileage record and populate it with data from an inbound email. You will first define the Apex class that creates a new object, then activate the email service which will execute your code.

Prerequisites

"Developer Mode," "Modify All Data," and "Author Apex" permissions
Since this tutorial involves working with Apex, ensure that you have the proper permissions to create Apex classes and configure Apex email services.

Note: Your Developer Edition Administration User has these permissions by default.

Mileage Tracker App
You first need to create the basic Mileage Tracker application as described in Tutorial #1: Creating an App with Point-and-Click Tools on page 3 or Tutorial #2: Creating an App with the Force.com IDE on page 9.

Software Requirements

- Eclipse 3.3: http://wiki.apexdevnet.com/index.php/Force.com_IDE_Installation_for_Eclipse_3.3.x
- Force.com IDE

Step 1: Create the Email Class

The first thing you must do is create the Apex email class which will be associated with the email service.

1. Open the Force.com IDE.
2. Ensure that you are in the `MileageTracker` project.
3. Right-click the project folder and select **New ➤ Apex Class**.
4. On the Create Apex Class page set the name to `EmailToApex`, and select the **Inbound Email Services** template.
5. Click **Finish** to create the class.

Step 2: Add a Utility Method

We need to create a utility method that will be used to parse the email body. This code takes in a line of text from the email as `plainTextBody`, offsets the beginning by the length of `pLabel`, then returns the substring of the remaining line. Using this utility method, we will extract relevant values for use in our Mileage object.

Add the following method to the class:

```
public static String getFieldValue(String plainTextBody, String pLabel) {
 Integer startPos = plainTextBody.indexOf(pLabel);
 Integer endPos = plainTextBody.indexOf('\n');
```

```
 return plainTextBody.substring(startPos+pLabel.length(), endPos);
}
```

Step 3: Add Logic to Deconstruct Inbound Email

After you enable the email to Salesforce functionality, every email received by the specified Salesforce email address creates an `InboundEmail` object that contains the contents and attachments of the email. In order to work with these objects, the `EmailtoApex` class needs to implement the `Messaging.InboundEmailHandler` interface. Because you are working with the email services, the class must also be declared as `global`.

Because we are implementing the `InboundEmailHandler` interface, it is mandatory to define the `handleInboundEmail` method. This method takes two input parameters: the inbound email and the envelope, and returns the `Messaging.InboundEmailResult` object.

1. Within the `handleInboundEmail` method, add logic to deconstruct an inbound email. Since the method returns a `Messaging.InBoundEmailResult` object, we instantiate it on the first line. For our example, the subject will contain the contact's name. This value is extracted using `email.subject` on the second line. On the third line, we employ the utility method defined previously to extract the value for the Miles field in the Mileage object.

   ```
   Messaging.InboundEmailResult result = new Messaging.InboundEmailResult();
   String contactName = email.subject;
   Double mileageInt = Double.valueOf(getFieldValue(email.plainTextBody, 'Miles:'));
   ```

2. Now, add logic to look up the contact and create the mileage record. In this part of the code, we first instantiate a list to contain the list of contacts that could be returned in a `SELECT` query based on the contact's name that was extracted from the email's subject line. Prepend and append a percentage sign (`'%'`) to `contactName` so that it can be used as a wild card search in the `SELECT` query.

   Next, instantiate a list of `Mileage__c` records that will be inserted. We begin the `try...catch` block by querying for the contact that was specified in the subject line of the email. We use a `for` loop to create `newMileage` records from the list of results. `Mileage__c` contains the name of the contact we queried, as well as the number of miles that was extracted from the email body.

   Then insert the record by calling `insert newMileage`.

   ```
   List<Contact> contactResult = new List<Contact>();
   contactName = '%'+contactName+'%';
   Mileage__c[] newMileage = new Mileage__c[0];
   try {
    for (Contact c :
     [Select Id, Name, Email From Contact Where Name like :contactName Limit 1]) {

      newMileage.add(new Mileage__c(miles__c = mileageInt, Contact__c = c.Id));
    }
    insert newMileage;
   } catch (System.Exception e) {
     System.debug('Error: ' + e);
   }
   ```

3. The last step is to return the result. Add the following to the end of the `handleInboundEmail` method:

   ```
   result.success = true; // If false, an email can be sent back with a message
   return result;
   ```

4. Verify that the complete class looks like this:

   ```
   global class EmailToApex implements Messaging.InboundEmailHandler {
   ```

```
public static String getFieldValue(String plainTextBody, String pLabel) {
 Integer startPos = plainTextBody.indexOf(pLabel);
 Integer endPos = plainTextBody.indexOf('\n');
 return plainTextBody.substring(startPos+pLabel.length(), endPos);
}

global Messaging.InboundEmailResult handleInboundEmail(Messaging.inboundEmail email,
 Messaging.InboundEnvelope envelope) {
 Messaging.InboundEmailResult result = new Messaging.InboundEmailResult();

 String contactName =email.subject;
 Double mileageInt = Double.valueOf(getFieldValue(email.plainTextBody, 'Mileage:'));

 List<Contact> contactResult = new List<Contact>();
 contactName = '%'+contactName+'%';
 Mileage__c[] newMileage = new Mileage__c[0];
 try {
  for (Contact c :
   [Select Id, Name, Email From Contact Where Name like :contactName Limit 1]) {
    newMileage.add(new Mileage__c(miles__c = mileageInt, Contact__c = c.Id));
  }
  insert newMileage;
 } catch (System.Exception e) {
   System.debug('Error: ' + e);
 }
 result.success = true; // If false, an email can be sent back with a message.
 return result;
}
}
```

5. Click **File ➤ Save**.

Note: If the project is in offline mode, you cannot use the `EmailToApex` class until it has been saved to the server. To save the class to the server:

1. In the IDE, select the **Classes** folder, then right-click and select **Force.com ➤ Save to Server**.
2. Click **Yes** to confirm the save.
3. Check the **Problems** tab for any errors or warnings that may have occurred as a result of saving the class.

Step 4: Set Up Email Services

Now you need to set up an email service that uses the `EmailToApex` class you completed in Step 3: Add Logic to Deconstruct Inbound Email on page 54.

1. In the online user interface, click **Setup ➤ Develop ➤ Email Services**.
2. Click **New Email Service**.
3. Set the `Email Service Name` to `MyEmailService`.
4. Set `Apex Class` to `EmailtoApex`.
5. Set `Accept Attachments` to `None`.
6. Select `Advanced Email Security Settings`.
7. Set `Accept Email From` to `yourdomain.com`.

Note: If your email address is `abc@xyz.com`, for example, your email domain is `xyz.com`

8. Select `Active`.
9. Click **Save**.

Step 5: Specify an Email Address

Now that you have the email service set up, you need to specify an email address that users can send email to.

1. Click **Setup ➤ Develop ➤ Email Services**, then click the name of the email service MyEmailService.
2. Click **New Email Address**.
3. Set the `Email address` to `mileagetracker`.
4. Set `Active` to Selected.
5. Set the `Context User` to a user whose permissions the email service will assume when processing emails.
6. Leave `Accept Email From` blank.
7. Click **Save**.

The Force.com platform then generates a unique email address based on your entries.

Step 6: Test the Email Service

From any email client, compose a message to the email address generated in Step 5: Specify an Email Address on page 56. If you are in the online user interface, you may be able to just click on the email address that was generated.

1. For the email fields, define the following:
   - Set the subject to `Pat Stumuller`
   - Set the body of the email to `Miles: 50`
2. Send the email.
3. Log in to your Developer Edition organization.
4. Go to the Mileage tab and click **Go** to display all mileage records.
5. You should see an entry for the mileage record you just created via email:

Summary

In this tutorial, you created an Apex class that implemented the `InboundEmailHandler` interface. You also defined an email service to process inbound emails to Salesforce using this Apex class.

It is important to remember that when you create an email service, you must associate it with an Apex class that handles the inbound emails. You should also ensure that the Apex classes that are working with email properly process any variation of the email text. In this example, the `getFieldValue` method did this.

Tutorial #10: Downloading and Deploying an App Using Code Share

Level: Intermediate; **Duration:** 20-30 minutes

Force.com Code Share (http://developer.force.com/codeshare) provides a directory of open-source code projects that support collaborative development, testing and deployment. Many of these applications run on the Force.com platform, and you can download the source code and metadata and deploy these applications to your own Developer Edition, sandbox or production environment.

One such application, stored on Google Code, is the Mileage Tracker application built in Tutorial #1: Creating an App with Point-and-Click Tools on page 3 or Tutorial #2: Creating an App with the Force.com IDE on page 9.

In this tutorial, you will learn how to download the source code for the Mileage Tracker application from the Google Code subversion repository and deploy it to your Force.com organization.

Prerequisites

Basic Knowledge

For this tutorial, it helps to have basic knowledge of Java and Eclipse, but it is not required.

Software Requirements

- Eclipse 3.3: http://wiki.apexdevnet.com/index.php/Force.com_IDE_Installation_for_Eclipse_3.3.x
- Force.com IDE
- Subclipse: http://wiki.apexdevnet.com/index.php/Google_Data_APIs_Toolkit_Setup#Installing_Subclipse

Getting Started

If you have already created the Mileage Tracker application you must first delete all the components you created—including the custom object, the tab, and the application—so that the names don't conflict with those in this tutorial.

Step 1: Download Source Code From a Repository

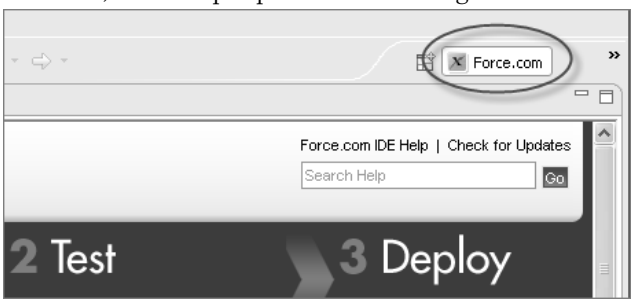
In this step, you will download the source code for the Mileage Tracker application—which lets users track miles driven—from a Google Code Subversion (SVN) repository.

1. In the Force.com IDE, click the **SVN Repository Exploring** perspective icon in the upper right corner. If this perspective is not available, click **Window ➤ Open Perspective ➤ Other** and choose **SVN Repository Exploring**.
2. If there are any repository locations (indicated by a yellow cylinder) in the **SVN Repository Explorer** area, delete them.
3. Right-click (Ctrl-click on a Mac) in the **SVN Repository Explorer** area and select **New ➤ Repository Location**.
4. Specify the repository's URL: `http://forcedotcomworkbook.googlecode.com/svn/trunk/`
5. Click **Finish**. A node symbol for the Google Code repository should now appear in the SVN Repository Explorer area.
6. Expand the **v14.0** repository one level so you can see the `src` folder.
7. Right-click the **v14.0** folder and choose **Checkout**.
8. Select **Check out as a project in the workspace**.
9. Enter `Mileage Tracker Repository` as the project name.
10. Accept the defaults and click **Finish**.

Step 2: Verify the Source Code

In this step, you will verify the metadata components that were downloaded in the previous step.

1. In the upper right-hand corner of the IDE, click the perspective list to change to the Force.com IDE perspective.



   You should now have a Force.com IDE project with the Mileage Tracker source code.
2. In the Package Explorer, expand the **Mileage Tracker Repository** folder to verify that you have the following files:

   - `applications/Mileage_Tracker.app`
   - `classes/MileageUtil.cls`
   - `objects/Mileage__c.object`
   - `tabs/Mileage__c.tab`
   - `triggers/MileageTrigger.trigger`

Note: Other files and folders may be included in your project.

Step 3: Deploy the Project to a Force.com Organization

Now that you have all the source code, you can deploy the application.

1. Connect the project to your Developer Edition organization.
   a. In the Package Explorer, right-click the **Mileage Tracker Repository** folder and choose **Force.com ➤ Add Force.com Nature**. This takes a few moments to run. A warning message displays when the process finishes, letting you know that you need to enter connection properties that define how to connect the project to your Developer organization.
   b. In the Package Explorer, right-click the **Mileage Tracker Repository** folder and choose **Properties**.
   c. Click the **Force.com** node and specify the following:
      - `Username`: enter your Developer Edition organization username
      - `Password`: enter your Developer Edition password
      - `Environment`: select **Production/Developer Edition Organization**
   d. Click **OK** to display the Refresh Project Contents window.
   e. Click **No** to prevent having code from the Developer Edition overwrite the code you just downloaded from the repository.
   f. Click **OK** to close the window.
2. Save (deploy) the application to the server you just specified.
   a. In the Package Explorer, right-click the **Mileage Tracker Repository** folder and choose **Force.com ➤ Save to Server**.
   b. Click **Yes** in the Confirm Save window.

Step 4: Test the Mileage Tracker Application

Now that you have deployed the application, you can test it.

1. Select the Mileage Tracker application from the Force.com app menu.
2. Click the **Mileage** tab.
3. Click **New**.
4. Enter the following values for these fields:
   - `Date`: Enter today's date.
   - `Miles`: `1000`
   - `Contact`: `Pat Stumuller`
5. Click **Save**. You should get two errors, one at the object level and one at the field level. These errors demonstrate that an Apex trigger was fired and correctly enforced a validation rule. In Tutorial #5: Adding Business Logic with Apex on page 29, validation logic was added to the application that limits mileage to 500 miles per day.
6. Change the value for **Miles** to be `499` or less. When the number of miles you enter does not exceed 500 miles for the day, you will no longer get an error when you click **Save**.

Summary

In this tutorial, you learned how to use Force.com Code Share to download existing application source code from a repository into your IDE. You also learned how to deploy the source code that you downloaded to your Developer Edition organization using the IDE.

Tutorial #11: Packaging and Distributing an Application

Level: Beginner; **Duration:** 15-20 Minutes

Packages are like suitcases that can contain your components, code, or applications. Packages are the preferred way to distribute an application to many organizations.

The Force.com platform provides two types of packages: unmanaged and managed. Unmanaged packages are used to distribute copies of the bundled components, including all of the source code. These are typically free packages where upgrading is not required. Managed packages are the preferred way to release applications on Force.com AppExchange. With managed packages, customers can upgrade easily to new versions of the application. Manage packages also provide intellectual property protection and license enforcement to the publisher.

To migrate changes from a test environment such as a sandbox to production, see the *Force.com Migration Tool Guide* at http://wiki.apexdevnet.com/index.php/Migration_Tool_Guide.

Prerequisites

Basic Knowledge
For this tutorial, it helps to have basic knowledge of Salesforce administration, but it is not required.

Mileage Tracker App
You first need to create the basic Mileage Tracker application as described in Tutorial #1: Creating an App with Point-and-Click Tools on page 3 or Tutorial #2: Creating an App with the Force.com IDE on page 9.

Software Requirements
You must have a second Developer Organization to test the package installation.

Getting Started
If you plan to package Apex classes with your application, you must write test cases that cover 75% of the code, and all of your triggers must have some code coverage.

Step 1: Create a Package

In this first step, we are going to create a new package for our application.

1. Log into your Developer Edition organization.
   a. Launch your browser and go to `https://login.salesforce.com`.
   b. Enter your Developer Edition username (in the form of an email address) and password.
2. Click **Setup ➤ Create ➤ Packages**.
3. Click **New**.
4. Set the `Package Name` to `Mileage Tracker`.
5. Click **Save**.

Step 2: Add Components to Your Package

A package must contain all the components in the application. Most components are automatically included when the application is added to the package. However, some components such as reports and dashboards, that are not explicitly tied to an application, must be manually included.

1. Click **Setup ➤ Create ➤ Packages**.
2. Click the `Mileage Tracker` package.

Note: If you have completed Tutorial #5: Adding Business Logic with Apex, and created an Apex trigger on the Mileage object, the trigger is automatically included when the Mileage Tracker application is added. Therefore, you must have tests written for this trigger to successfully add the Mileage Tracker application.

3. In the Package Components section, click **Add**.
4. Select **Custom App** from the `Component Type` drop-down list.
5. Click the check box next to `Mileage Tracker`.
6. Click **Add To Package**.
7. Repeat steps 3-6 for the following components.

Note: To add the Workflow rule, you need to have completed Tutorial #4: Using Workflow and Approvals. To add Apex classes, you need to have completed Tutorial #5: Adding Business Logic with Apex, Tutorial #9: Adding Email Services and Tutorial #6: Adding Tests to Your Application. If you have not completed these tutorials, you can continue without adding their components.

Component Type	Component Name
Visualforce Page	MyMileagePage
Workflow Rule	Large Mileage Rule
Apex Class	EmailtoApex
Apex Class	MileageTrackerTestSuite
Apex Class	MileageUtil

Step 3: Convert to a Managed Package

All packages are unmanaged by default. To make this package managed, you must register for a namespace prefix and select this package as the managed package. A namespace prefix is an identifier attached to all your components that makes them globally unique across all Salesforce customers. Namespace prefixes are only used with managed packages.

Note: Managed packages can only be created from Developer Edition organizations. A Developer Edition organization can only create and upload one managed package. If you are not able to perform this tutorial in a Developer Edition organization without a specified managed package, skip this step and continue with an unmanaged package.

1. Click **Setup ➤ Create ➤ Packages**.
2. Click **Edit** under `Developer Settings`.
3. Read the information on namespace prefixes and click **Continue**.
4. Enter a `Namespace Prefix`.
5. Click **Check Availability**. If the namespace prefix is available, continue. If the namespace prefix is not available enter another namespace prefix and try again.
6. Select **Mileage Tracker** from the `Package to be managed` drop-down list.
7. Click **Review My Selections**.
8. Review your selection and click **Save**.

Step 4: Upload the Package

The managed package is ready for upload. Managed packages can be uploaded in two states: Managed - Beta or Managed - Released. Managed - Released is used after a package is tested and ready to be distributed to customers. Once a package is

released, certain changes are no longer possible, such as deleting a custom object or field. This is required to allow customers to install new versions of the package.

During development, testing, and beta customer evaluation, a package should be uploaded as Managed - Beta to allow the developer to continue to make changes. Managed - Beta packages can only be installed in Developer Edition or Sandbox organizations and are not upgradeable.

1. Click **Setup ➤ Create ➤ Packages**.
2. Click the Mileage Tracker package.
3. Click **Upload**.
4. Set the `Version Name` to `1.0`.
5. Select Managed - Beta for the `Release Type`.
6. Click **Continue**.
7. Wait for the page to refresh with an `Installation URL`, then copy and save the URL to a text file. The installation URL is needed in the next step. Salesforce also sends the installation URL to the email address associated with the account.

Step 5: Install the Package

To install and test your managed package, you need a second Developer Edition organization. If you do not have a second Developer Edition organization, register for one at: http://wiki.apexdevnet.com/events/regular/registration.php.

1. Go to the Installation URL generated in the previous step.
2. Enter your second Developer Edition username and password.
3. Review the components in the package and click **Continue**.
4. Click **Next** to approve the API access.
5. If asked, keep the default security level and click **Next**.
6. Click **Install**.
7. The package is now installed and ready for testing.

Summary

Congratulations, you have successfully created and installed a Managed - Beta package containing a custom app. Due to your unique namespace prefix all components in your managed package are globally unique across all Salesforce users. Once you've tested your custom app in another Developer Organization, you can upload a Managed - Released version of your package. You can publish Managed - Released packages on Force.com AppExchange to sell or share with other Salesforce users or you can distribute your installation URL directly to your customers.

developerforce™

Go to http://developer.force.com for more developer resources.

Corporate Headquarters
The Landmark @ One Market
Suite 300
San Francisco, CA, 94105
United States

1-800-NO-SOFTWARE
www.salesforce.com

Latin America
+1-415-536-4606

Japan
+81-3-5785-8201

Asia/Pacific
+65-6302-5700

Europe, Middle East & Africa
+4121-6953700

BK_ForcePlatformWorkbook_v2_0_100208