



Salesforce.com: Summer '10

Adobe Flash Builder for Force.com Quickstart



Last updated: June 8, 2010

© Copyright 2000-2010 salesforce.com, inc. All rights reserved. Salesforce.com is a registered trademark of salesforce.com, inc., as are other names and marks. Other marks appearing herein may be trademarks of their respective owners.

Table of Contents

Creating Your First Force.com Flex App with Force.com Flex.....	2
Prerequisite: Create the Merchandise Object and Fields.....	3
Step 1: Configure Your Salesforce.com Personal Information.....	5
Step 2: Create an Offline Briefcase Configuration.....	5
Step 3: Generate Your Enterprise WSDL and Download a Client Certificate.....	6
Step 4: Install and Launch Force.com Flex.....	6
Step 5: Create a Force.com Flex Project.....	6
Step 6: Set the Force.com Flex Application Component Attributes.....	9
Step 7: Create the Inventory Tracker Window Component.....	9
Step 8: Testing the Inventory Tracker App.....	15
Summary.....	18

Creating Your First Force.com Flex App with Force.com Flex

Force.com Flex is a set of tools that let you develop Force.com Flex apps—desktop applications that operate in the Adobe Integrated Runtime (AIR) environment or Web applications that leverage Force.com logic and database capabilities. Force.com Flex desktop apps can operate both *online* and *offline*, making them ideal for users who do not always have Internet connectivity but still need to access data in your Force.com apps.

This tutorial demonstrates how to use Force.com Flex to build a basic Force.com Flex app called Inventory Tracker that provides offline access to merchandise data. The app will enable managers to track inventory on tablet PCs while on-foot in a warehouse that has inconsistent Wi-Fi, and will include the following standard Force.com Flex app features:

- A login screen that prompts users for their credentials and established connectivity with Force.com when online.
- A conflict resolution interface that lets users easily resolve data conflicts that occur when values in the Force.com Flex app violate validation rules or conflict with changes made by other users.
- A status bar that notifies users whether the Force.com Flex app is connected to the Internet, and indicates the number of unresolved data conflicts and errors.

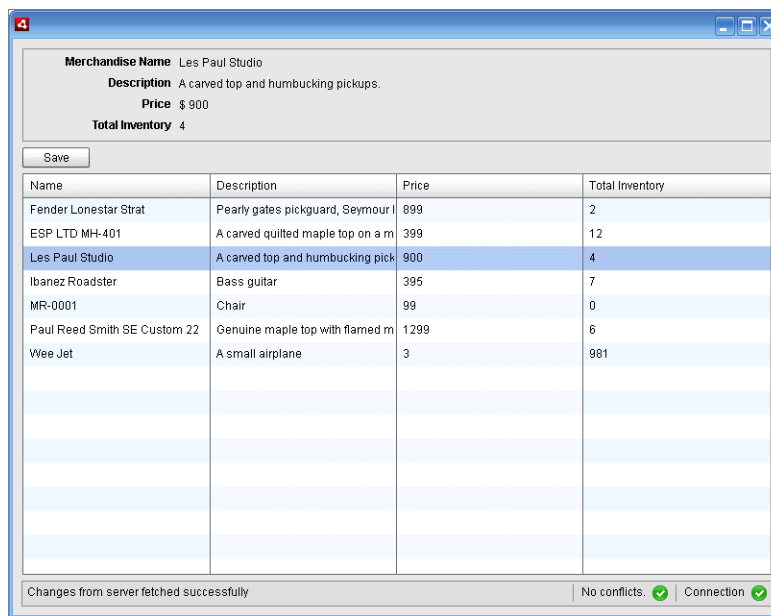


Figure 1: The Inventory Tracker Force.com Flex App

To create the app, we'll use Adobe Flash technologies such as MXML and ActionScript components alongside Force.com Flex classes that Force.com Flex generates; however, you don't need to know MXML and ActionScript to create the Inventory Tracker app— this tutorial contains all the code you need to get your first Force.com Flex app up and running.



Caution: Force.com Flex is currently only available as a developer preview and may change significantly before it is generally available. During the developer preview, do not use Force.com Flex to create apps for production environments.

Prerequisite: Create the Merchandise Object and Fields

Force.com Flex apps are based on the objects and fields in Salesforce.com. Before you begin using Force.com Flex to build Force.com Flex apps, you must create the data model in Salesforce.com.

For the Inventory Tracker app, the only necessary data model change is the addition of a custom object called Merchandise and its custom fields on the object. Create the them as follows:

1. Create the Merchandise object.
 - a. Log into Salesforce.com as an administrator with the “Customize Application” permission.
 - b. Click **Setup** ► **Create** ► **Objects**.
 - c. Click **New Custom Object**.

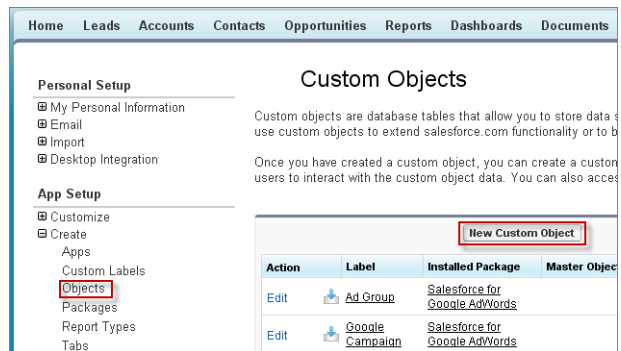


Figure 2: New Custom Object Button

- d. Fill out the fields on the page according to the following table:

Field Name	Value	Description
Label	Merchandise	The singular version of the name of the object as it appears in the user interface.
Plural Label	Merchandise	The plural version of the name of the object as it appears in the user interface.
Record Name	Merchandise Name	The name used in page layouts, list views, related lists, and search results in Salesforce.com.
Data Type	Text	The type of field (text or auto-number) for the record name. Records that have unique IDs instead of names use auto-numbers. An auto-number is a unique sequential number assigned automatically. It is always a read-only field.

Leave all other values as they are.

Figure 3: The Merchandise Custom Object

- e. Click **Save** to finish creating your new object. The Merchandise object detail page appears.
2. Create the `Description` field.
 - a. Click the **New** button in the Custom Fields & Relationships related list on the Merchandise object detail page.
 - b. Scroll down to the Custom Fields & Relationships related list on the Merchandise detail page.
 - c. Click **New** to launch the New Custom Field wizard.
 - d. For Data Type, select `Text` Area and click **Next**.
 - e. In the Field Label field, enter `Description`.
 - f. In the Field Name field, leave the default value `Description__c`.
 - g. Click **Next**, accept the defaults, and click **Next** again.
 - h. Click **Save & New** to save the `Description` field and return to the first step of the wizard.
 3. Create the `Price` field.
 - a. Click the **New** button in the Custom Fields & Relationships related list.
 - b. For Data Type, select `Currency`, and click **Next**.
 - c. In the Field Label field, enter `Price`.
 - d. In the Field Name, leave the default value `Price__c`.
 - e. In the Length field, enter 16.
 - f. In the Decimal Places field, enter 2.
 - g. Leave the defaults for the remaining fields, and click **Next**.
 - h. In the next step, accept the defaults, and click **Next**.
 - i. In the next step, click **Save & New** to save the `Price` field and to return to the first step of the wizard.
 4. Create the `Total Inventory` field.
 - a. Click the **New** button in the Custom Fields & Relationships related list.
 - b. For Data Type, select `Number` and click **Next**.

- c. In the `Field Label` field, enter `Total Inventory`.
- d. In the `Field Name` field, leave the default value `Total Inventory__c`.
- e. Accept the defaults for the remaining fields, and click **Next**.
- f. In the next step, accept the defaults, and click **Next**.
- g. Click **Save** to create the `Inventory` field and to return to the `Merchandise Custom Object` page.

Step 1: Configure Your Salesforce.com Personal Information

Force.com Flex apps leverage the `Connect Offline` technology in Force.com. To develop or use a Force.com Flex app, you must have the `Offline User` option selected in your Salesforce.com personal information.

1. In Salesforce.com, click **Setup** ► **Manage Users** ► **Users**.
2. Click **Edit** next to your name.
3. Select the `Offline User` checkbox.
4. Click **Save**.

Step 2: Create an Offline Briefcase Configuration

Your Force.com desktop app accesses only Force.com data specified in the offline briefcase configuration assigned to you in Salesforce.com. The `Inventory Tracker` app needs to access merchandise data, so create an offline briefcase configuration that includes the `Merchandise` object.

1. In Salesforce.com, click **Setup** ► **Desktop Administration** ► **Offline Briefcase Configuration**.
2. Click **New Offline Briefcase Configuration**.
3. Enter a name for your offline briefcase configuration, such as `Inventory Tracker App Data`.
4. Select the `Active` checkbox.
5. Select your name in the `Available Members` list and click **Add** to move it to the `Assigned Members` list.
6. Click **Save**. The `Offline Briefcase Configuration` detail page appears.
7. Click **Edit** in the `Data Sets` related list.
8. Click **Add...**
9. Select `Merchandise` and click **OK**.
10. Select `Merchandise Name` in the `Order By` picklist.
11. Click **Done**.

Tell Me More....

Offline briefcase configurations are sets of parameters that determine the records available in Force.com Flex apps and Force.com `Connect Offline`—a client application that lets you access a subset of Force.com data using the same browser-based interface as the online system but without an Internet connection. Use `Connect Offline` to view, edit, create, and delete accounts, activities, contacts, opportunities, leads, and custom object records (including relationship groups). You can also add and update products and schedules on opportunities.

You can create multiple briefcase configurations and associate each with different users and profiles to simultaneously suit the needs of various types of offline users. For example, one configuration might include leads and opportunities for sales representatives, while another configuration includes accounts and related opportunities for account executives.

Step 3: Generate Your Enterprise WSDL and Download a Client Certificate

Web services are a common form of integration, and Force.com provides a robust set of functionality to support both incoming and outgoing Web services calls. Force.com also, automatically, generates a Web service end point that allows access to the data in your application. "Enterprise WSDL" refers to the Web Services Definition Language (WSDL) that describes your Force.com objects. This is typically all that's needed when integrating with another system as it fully describes all the objects and the Web service.

Force.com Flex lets you import your enterprise WSDL into development projects. When you import your WSDL, Force.com Flex generates ActionScript classes for every object in your enterprise WSDL, allowing you to reference Force.com objects in your ActionScript code.

We will import our enterprise WSDL into our development project later to create a class for the Merchandise object. For now, save a local copy of your current enterprise WSDL. Also, download a Salesforce.com client certificate, which authorizes client applications to access Force.com.

1. In Salesforce.com, click **Setup** ► **Develop** ► **API**.
2. Click **Generate Enterprise WSDL**.
3. In your browser, click **File** ► **Save Page As...**
4. Name the file `enterprise.wsdl` and click **Save**.
5. Click **Generate Client Certificate** to generate and save the client certificate for your machine.

Step 4: Install and Launch Force.com Flex

Force.com Flex installs as a standalone version of Adobe® Flash® Builder™, Adobe's integrated development environment (IDE) based on the Eclipse platform.

1. Download the Force.com Flex zip file from <http://developer.force.com/flashbuilder>.
2. Extract the contents of the zip file.
3. Double-click the installer.
4. When the installation is complete, double-click the Force.com Flex shortcut on your desktop.

Step 5: Create a Force.com Flex Project

After you install Force.com Flex, a new project type called Force.com Flex Project is available in Flash Builder. Force.com Flex projects let you import your enterprise WSDL and automatically generate ActionScript classes for your Salesforce.com objects. Flash Builder also provides code hints and compiler warnings based on your WSDL.

1. In Flash Builder, select **File** ► **New Project**. The New Project wizard opens.
2. Select the Force.com Stratus Project in the Flash Builder folder and click **Next**.

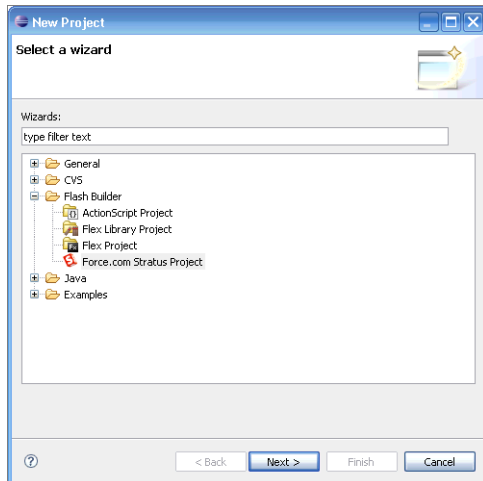


Figure 4: New Project Wizard, Step 1

3. Enter `Inventory Tracker` as the project name, select `Desktop` (runs in Adobe AIR), and click **Next**.

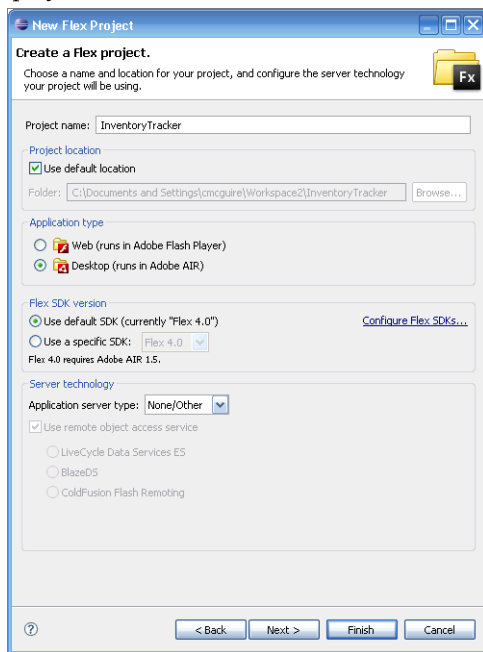


Figure 5: New Project Wizard, Step 2

4. Accept the default settings for the output folder and click **Next**.
5. Enter `com.salesforce.InventoryTracker` in the `Application ID` field. The AIR runtime environment and the operating system to identify the application. To guarantee a unique application ID, use reverse domain name notation. Click **Finish** to continue.

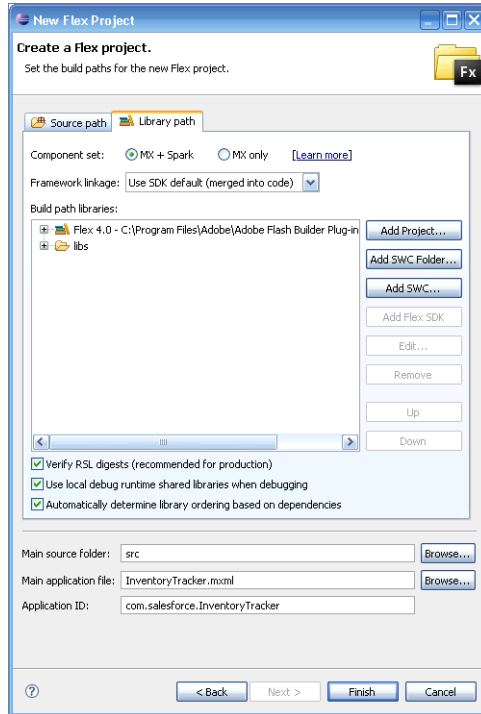


Figure 6: New Project Wizard, Step 3

6. Select your enterprise WSDL, then click **Next**.

After you create a Force.com Flex project, the following directory structure appears in your Flash Builder Package Explorer.

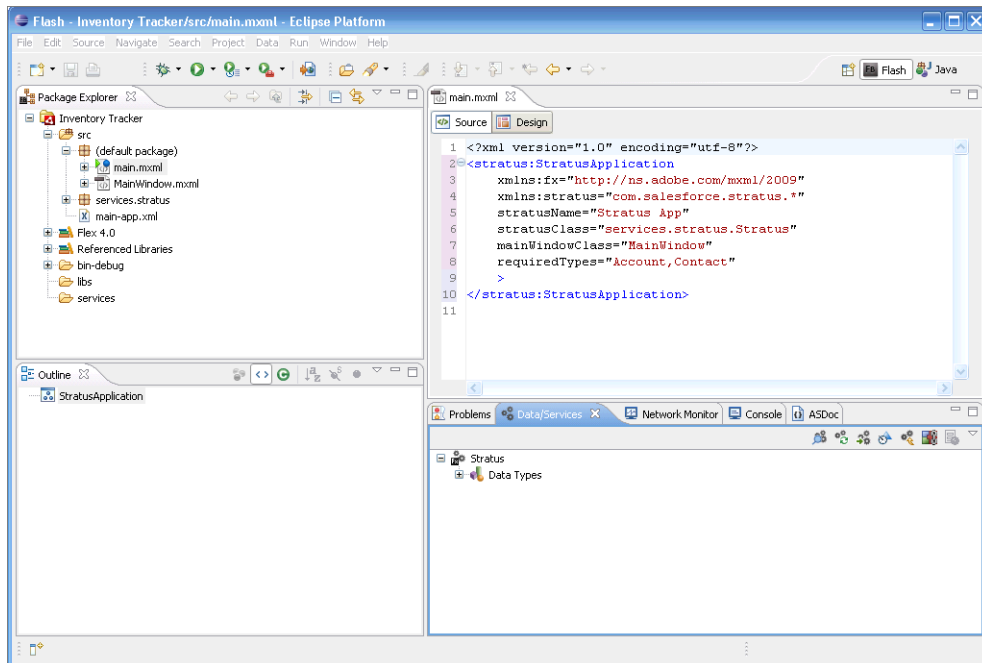


Figure 7: Force.com Flex Project Directory Structure

Step 6: Set the Force.com Flex Application Component Attributes

Every Force.com Flex app is contained within a Force.com Flex Application component. The Force.com Flex Application component connects the Force.com Flex app to Force.com, provides the Force.com login and authentication functionality, and handles the initial synchronization of data between Force.com and the Force.com Flex app.

Force.com Flex projects have a default `main.mxml` file that contains a Force.com Flex Application template you can modify for your Inventory Tracker app. Modify the attributes of the `stratus:StratusApplication` element in this file.

1. Open the `main.mxml` file located in **src** > **(default package)** if it is not already open.
2. Set the `stratusName` attribute to `"Inventory Tracker"`. This controls the name that appears in the title bar of the application.
3. Set the `requiredTypes` attribute to `"Merchandise__c"`. This attribute determines which of the objects in your Force.com Flex app the Force.com Flex app references.
4. Your final code should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<stratus:StratusApplication
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:stratus="com.salesforce.stratus.*"
  stratusName="Inventory Tracker"
  stratusClass="services.stratus.Stratus"
  mainWindowClass="MainWindow"
  requiredTypes="Merchandise__c"
>
</stratus:StratusApplication>
```

5. Save the file.

Step 7: Create the Inventory Tracker Window Component

The user interface of the Inventory Tracker will consist of two parts. The lower part will contain a list of merchandise records at the bottom. When you select a record in that list, the details of the record will appear in the top part. You will be able to click on the record details to edit them. A **Save** button below the record details will let you commit those changes to Force.com.

The logic for the main functionality of the Inventory Tracker app consists of ActionScript and MXML that utilize two Adobe component sets, Spark and MX, as well as Force.com Flex components. Force.com Flex components are components generated by Force.com Flex, and simplify the process of accessing Force.com logic and data in your code.

You'll enter the application logic in the default `WindowedApplication` component that Force.com Flex generates when you create a Force.com Flex project. The default `WindowedApplication` component is in the file `DefaultDesktopProject.mxml`. The default code is:

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/mx"
  xmlns:flexforforce="http://flexforforce.salesforce.com"
  title="Salesforce.com FlexForForce"
  showStatusBar="false">

  <fx:Declarations>
    <s:TraceTarget includeCategory="true"
      includeLevel="true"
```

```

        includeTime="true"
        level="{LogEventLevel.INFO}"
        filters="{['com.salesforce.*']}" />

<flexforforce:F3DesktopApplication id="app"
    statusChanged="statusChangedHandler(event)"
    loginComplete="loginCompleteHandler(event)"
    loginFailed="loginFailedHandler(event)"
    sessionExpired="sessionExpiredHandler(event)"
    requiredTypes="Account,Contact" />
</fx:Declarations>

<fx:Script>
    <![CDATA[
        import com.salesforce.events.LoginFaultEvent;
        import com.salesforce.events.LoginResultEvent;
        import com.salesforce.events.SessionExpiredEvent;
        import com.salesforce.events.StatusChangedEvent;

        import mx.controls.Alert;
        import mx.logging.LogEventLevel;
        import mx.managers.CursorManager;

        private var _username : String;
        private var _password : String;

        protected function statusChangedHandler( event : StatusChangedEvent ) : void {
            status = event.message.description;
        }

        protected function loginClickHandler( event : MouseEvent ) : void {
            _username = username.text;
            _password = password.text;
            CursorManager.setBusyCursor();
            app.loginByCredentials( _username, _password );
        }

        protected function loginFailedHandler( event : LoginFaultEvent ) : void {
            CursorManager.removeBusyCursor();
            // By default an alert shows that the login failed.
            // To implement your own logic you can override the
            // default behavior by uncommenting the following line:
            //event.preventDefault();
            currentState = "login";
        }

        protected function loginCompleteHandler( event : LoginResultEvent ) : void {
            CursorManager.removeBusyCursor();
            // When the login is complete the main state should be shown.
            currentState = "main";
        }

        protected function sessionExpiredHandler(event : SessionExpiredEvent) : void {
            // An alert will be shown if the default behavior is not overridden.
            event.preventDefault();

            Alert.show(
                "Your session expired and was automatically renewed.
                Please repeat your last operation.",
                "Session Expired"
            );

            CursorManager.setBusyCursor();
            // When the login session expires the user can be re-logged in automatically.

            app.loginByCredentials( _username, _password );
        }
    ]]>

```

```

    ]]>
</fx:Script>

<s:states>
  <s:State name="login"/>
  <s:State name="main"/>
</s:states>

<s:VGroup includeIn="login" horizontalCenter="0" verticalCenter="0">
  <s:Label text="Salesforce.com Login" fontWeight="bold"/>
  <mx:Form>
    <mx:FormItem label="Username">
      <s:TextInput id="username" text=""/>
    </mx:FormItem>
    <mx:FormItem label="Password" direction="horizontal">
      <s:TextInput id="password" text="" displayAsPassword="true"/>
      <s:Button label="Login"
        enabled="{!app.loginPending}"
        click="loginClickHandler(event)"/>
    </mx:FormItem>
    <mx:ProgressBar indeterminate="true"
      visible="{app.loginPending}"
      label="{status}"/>
  </mx:Form>
</s:VGroup>

<s:Group includeIn="main"
  enabled="{!app.loginPending}"
  width="100%"
  height="100%">

  <s:layout>
    <s:BasicLayout/>
  </s:layout>

  <s:VGroup paddingTop="10"
    paddingLeft="10"
    paddingRight="10"
    paddingBottom="10">

    <s:Label text="Put your components here"/>
  </s:VGroup>

  <flexforforce:StatusBar left="0" bottom="0"/>
</s:Group>

</s:WindowedApplication>

```

The default code includes:

- A Spark `WindowedApplication` component that specifies the namespaces, background color, and dimensions for the Force.com Flex app. The `showStatusBar` attribute, which determines whether the application renders the standard Flash Builder status bar, is set to `false` to allow the application to show the Force.com Flex status bar instead.
- The `F3DesktopApplication` component includes a `loginComplete` attribute that determines the function that executes when a user logs in, as well as a `statusChanged` attribute that determines the function handling status change events and messages that will be displayed in the Force.com Flex status bar component.
- A Spark `Layout` component that arranges the layout elements according to their individual settings.
- A Spark `BasicLayout` component that arranges the layout elements in a vertical sequence, top to bottom, with optional gaps between the elements and optional padding around the sequence of elements.
- A Spark `VGroup` component that creates a vertical grouping of the user interface elements.

- An XML Script component in which you can enter ActionScript. This component contains default ActionScript that initializes the data when the Force.com Flex app opens.
- A Force.com Flex StatusBar component that renders a status bar at the bottom of the Force.com Flex app.

Modify the default code to implement the Inventory Tracker app logic as follows:

1. Remove the Label component:

```
<s:Label text="Your content goes here" />
```

2. Inside the VGroup component, add a Force.com Flex FieldContainer component. Force.com Flex FieldContainer components group data and simplify the process of manipulating several fields in a single operation. The FieldContainer you're creating here will also render the details of the selected record at the top of the screen.

Set the FieldContainer component width to 100% and for the ID, enter `_editFieldContainer`. You'll reference the FieldContainer ID later in the code.

```
<flexforforce:FieldContainer id="_editFieldContainer" width="100%">
</flexforforce:FieldContainer>
```

3. Within the FieldContainer component, add one LabelAndField component for each field of the Merchandise object you want to display, and reference each field by its internal Force.com API name. A LabelAndField component is a Force.com Flex component that renders a Salesforce.com field value with its label. The fields you create with the FieldAndLabel component automatically function in Force.com Flex apps the same way they do on the Salesforce.com user interface. For example, date fields display a calendar when clicked. The fields also respect field dependencies, and automatically provide inline editing, hover details, error notification, and Info icon help text.

```
<flexforforce:LabelAndField field="Merchandise__c.Name" />
<flexforforce:LabelAndField field="Merchandise__c.Description__c" />
<flexforforce:LabelAndField field="Merchandise__c.Price__c" />
<flexforforce:LabelAndField field="Merchandise__c.Total_Inventory__c" />
```

4. After adding the LabelAndField components, be sure to add the closing FieldContainer element if you haven't already done so.

```
</flexforforce:FieldContainer>
```

5. Now add a standard Spark button, label it **Save**, and configure it to execute a function called `onEditSaveClick`, which you'll write later.

```
<s:Button label="Save" click="onEditSaveClick()" />
```

6. Below the **Save** button in the Inventory Tracker user interface, you want to render a list of merchandise records. You'll use a standard ActionScript DataGrid component to render the list. Enter the code as shown below.

```
<mx:DataGrid id="dataGrid" width="100%" height="100%" dataProvider="{_gridDataProvider}"
itemClick="onDataGridItemClick()">
  <mx:columns>
    <mx:DataGridColumn dataField="Name" />
    <mx:DataGridColumn dataField="Description__c" headerText="Description" />
    <mx:DataGridColumn dataField="Price__c" headerText="Price" />
    <mx:DataGridColumn dataField="Total_Inventory__c" headerText="Total Inventory" />
  </mx:columns>
</mx:DataGrid>
```

Note that the `dataProvider` attribute references a variable that you'll create later, and the `itemClick` attribute references a function that you'll create later.

- The final component in the `VGroup` element is the `StatusBar` component, which is part of the default code. Leave this component as is, and make sure you add the closing `VGroup` element.

```
<flexforforce:StatusBar/>
</s:VGroup>
```

- The Inventory Tracker app user interface is now in place, but you need to add the `ActionScript` that retrieves `Salesforce.com` data and saves the data in the `Force.com Flex` app. Begin the `ActionScript` by importing the classes you need for the `ActionScript` in the `<![CDATA[` section inside the `<fx:Script>` element. The code will look like this:

```
import mx.collections.ArrayCollection;
import mx.data.IManaged;
import com.salesforce.flexforforce.F3DesktopWrapper;
import services.flexforforce.Merchandise__c;
import com.salesforce.flexforforce.StratusMessage;
```

The classes are as follows:

- `ArrayCollection`—An `ActionScript` wrapper that exposes an array as a collection that can be accessed and manipulated.
- `IManaged`—An `ActionScript` interface that provides the contract for a managed object.
- `F3DesktopWrapper`—A `Force.com Flex` component that wraps the functionality of Adobe's Data Management Service (DMS), a key component of Adobe AIR platform that provides a model to manage client-server data synchronization.
- `Merchandise__c`—The `ActionScript` class of the `Merchandise` object generated by `Force.com Flex`.
- `F3Message`—A `Force.com Flex` component that provides standard contextual error and information messages, such as data conflicts.

- Create a variable to hold the merchandise data. The code will use this variable to display the data in the list.

```
[Bindable] private var _gridDataProvider:ArrayCollection = null;
```

- Create a variable that your code can use to access the DMS functionality.

```
private var _flexforforceWrapper:F3DesktopWrapper;
```

- The `onWindowComplete()` function in the default code queries the database when the applications loads. Modify this function to specifically return merchandise data. The complete function looks like this:

```
public function onWindowComplete():void {
    new StratusMessage(StratusMessage.STATUS_INFO, "Application
initialized").showAsStatus();
    _flexforforceWrapper = F3DesktopWrapper.getInstance();
    _flexforforceWrapper.query("select * from Merchandise__c", new StratusResponder(
        function(rows:ArrayCollection):void { _gridDataProvider = rows; },
        null
    ));
}
```

You'll recall that you referenced a function called the `WindowComplete` attribute in the `Window` element at the beginning of this step in the tutorial.

12. Create a function named `onEditSaveClick` that copies the changes a user makes to the data back into memory from the user interface.

```
private function onEditSaveClick():void {
    _editFieldContainer.fieldCollection.updateObject(new StratusResponder(commitToDB,
null));
}
```

This is the functionality that a user will trigger by clicking the **Save** button. This function also performs a simple validation of the data the user enters. If the data is valid, the function calls the `commitToDB` function.

13. Create the `commitToDB` function, which commits the data in memory to the database, and clears the data from the `FieldContainer`.

```
private function commitToDB(managedObject:IManaged):void {
    _flexforforceWrapper.save(managedObject, new StratusResponder(
function(result:F3Message):void { _editFieldContainer.fieldCollection.clear(); },
null
));
}
```

14. Create a function that renders the `FieldContainer` data in the Force.com Flex app. Refer to the `FieldContainer` by the ID you set for it earlier (`_editFieldContainer`).

```
private function onDataGridItemClick():void {
    var merchandise:Merchandise__c = _dataGrid.selectedItem as Merchandise__c;
    _editFieldContainer.fieldCollection.render(merchandise);
}
```

15. Save the file.

Your final code should look like this.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Window
xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/halo"
xmlns:flexforforce="com.salesforce.flexforforce.*"
showStatusBar="false"
width="800"
height="600"
backgroundColor="#e7e7e7"
windowComplete="onWindowComplete()">

<s:layout>
<s:VerticalLayout/>
</s:layout>

<s:VGroup width="100%" height="100%" paddingBottom="10" paddingLeft="10" paddingRight="10"
paddingTop="10">
<flexforforce:FieldContainer id="_editFieldContainer" width="100%">
<flexforforce:LabelAndField field="Merchandise__c.Name" />
<flexforforce:LabelAndField field="Merchandise__c.Description__c" />
<flexforforce:LabelAndField field="Merchandise__c.Price__c" />
<flexforforce:LabelAndField field="Merchandise__c.Total_Inventory__c" />
</flexforforce:FieldContainer>
<s:Button label="Save" click="onEditSaveClick()" />
<mx:DataGrid id="_dataGrid" width="100%" height="100%" dataProvider="{_gridDataProvider}"
itemClick="onDataGridItemClick()">
<mx:columns>
```

```

    <mx:DataGridColumn dataField="Name"/>
    <mx:DataGridColumn dataField="Description__c" headerText="Description"/>
    <mx:DataGridColumn dataField="Price__c" headerText="Price"/>
    <mx:DataGridColumn dataField="Total_Inventory__c" headerText="Total Inventory"/>
  </mx:columns>
</mx:DataGrid>
<flexforforce:StatusBar/>
</s:VGroup>

<fx:Script>
<![CDATA[
  import mx.collections.ArrayCollection;
  import mx.data.IManaged;
  import com.salesforce.flexforforce.F3DesktopWrapper;
  import com.salesforce.flexforforce.F3Message;
  import services.flexforforce.Merchandise__c;

  [Bindable] private var _gridDataProvider:ArrayCollection = null;
  private var _flexforforceWrapper:F3DesktopWrapper;

  public function onWindowComplete():void {
    new F3Message(F3Message.STATUS_INFO, "Application initialized").showAsStatus();
    _flexforforceWrapper = F3DesktopWrapper.getInstance();
    _flexforforceWrapper.query("select * from Merchandise__c", new StratusResponder(
      function(rows:ArrayCollection):void { _gridDataProvider = rows; },
      null
    ));
  }

  private function onEditSaveClick():void {
    _editFieldContainer.fieldCollection.updateObject(new StratusResponder(commitToDB, null));
  }

  private function commitToDB(managedObject:IManaged):void {
    _flexforforceWrapper.save(managedObject, new StratusResponder(
      function(result:F3Message):void { _editFieldContainer.fieldCollection.clear(); },
      null
    ));
  }

  private function onDataGridItemClick():void {
    var merchandise:Merchandise__c = _dataGrid.selectedItem as Merchandise__c;
    _editFieldContainer.fieldCollection.render(merchandise);
  }
]}>
</fx:Script>
</s:Window>

```

Step 8: Testing the Inventory Tracker App

The Inventory Tracker app is now ready to test.

1. Verify that your machine has an active Internet connection.
2. In Flash Builder, right-click on the Inventory Tracker project and select **Run as... ► Desktop Application**. The Inventory Tracker app login screen appears.

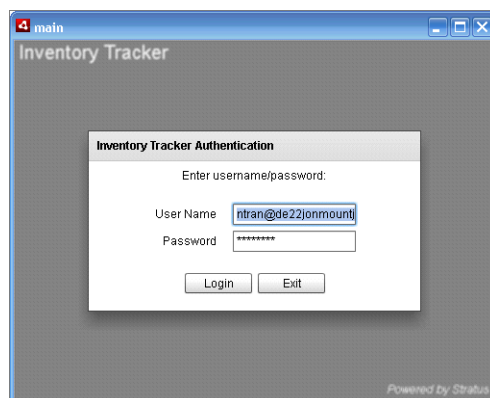


Figure 8: The Inventory Tracker App Login Screen

3. In the login screen, enter the username and password you use to log into your Force.com Flex app, and click **Login**. The Inventory Tracker user interface appears.

Name	Description	Price	Total Inventory
Fender Lonestar Strat	Pearly gates pickguard, Seymour I	899	2
ESP LTD MH-401	A carved quilted maple top on a m	399	12
Les Paul Studio	A carved top and humbucking pick	900	4
Ibanez Roadster	Bass guitar	395	7
MR-0001	Chair	99	0
Paul Reed Smith SE Custom 22	Genuine maple top with flamed m	1299	6
Wee Jet	A small airplane	3	981

Figure 9: The Inventory Tracker App

4. Select a record in the list and change its data, then click **Save**.
5. Log into your Force.com app and verify that your change appears.
6. To test the data conflict resolution capability, change a field in your Force.com app and click **Save**, then change the same field in the Inventory Tracker app to a different value. When you click **Save** in the Inventory Tracker app, notice that the status bar alerts you that there is a conflict.

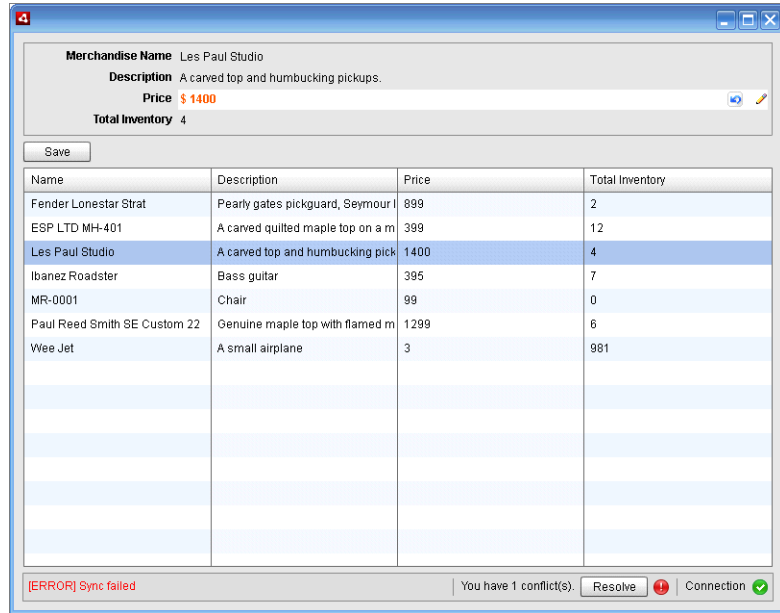


Figure 10: Data Conflict Notification

7. Click **Resolve** in the status bar to launch the conflict resolution interface.

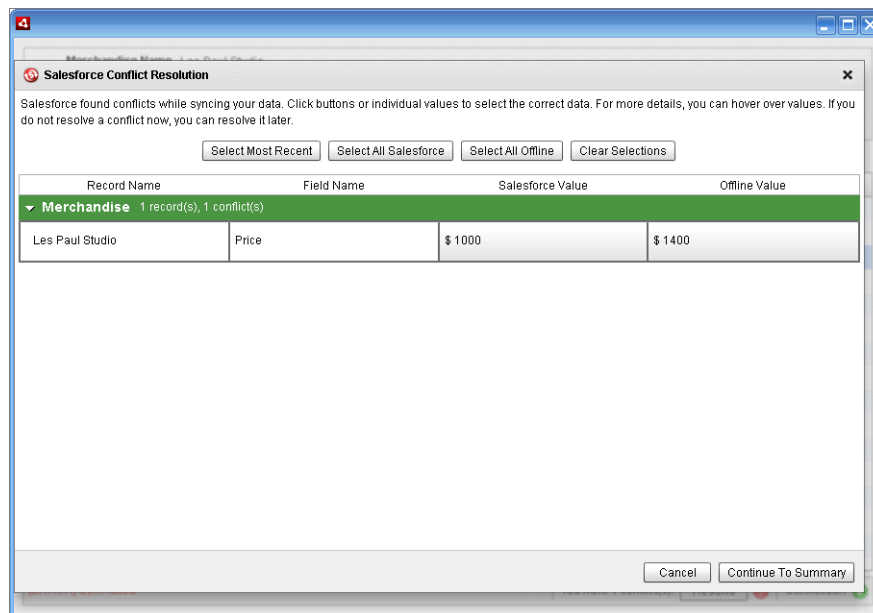


Figure 11: Conflict Resolution Interface

The conflict resolution interface lets you see the values in both Salesforce.com and the Inventory Tracker app. Select the value you want to keep, then click **Continue To Summary**.

Summary

In this tutorial, you created a simple Force.com Flex app that gives users the ability to view and edit Force.com data both *with* and *without* an Internet connection. It demonstrates the fundamentals of Force.com Flex without much coding. As you can probably imagine, you can use Force.com Flex to create far more powerful Force.com Flex apps to enhance the ways in which users interact with your Force.com apps.