

Porting the Adobe Developer Center Flex 3 Dashboard to Flex 4

By Greg Lafrance

glafrance@stardustsystems.com

Stardust Systems Tutorials

<http://www.StardustSystems.com>
<http://www.StardustSystems.com/blog>

Trademarks

Stardust Systems and the Stardust Systems logo are trademarks of Stardust Systems. Such trademarks may be registered in the United States or in other jurisdictions, including internationally. This manual may include trademarks, service marks, or trade names of Adobe Systems Incorporated, Inc. and other companies. Such trademarks, service marks, or trade names may be registered in the United States or in other jurisdictions, including internationally.

Third-Party Information

This manual contains information such as links to third-party websites that are not under the control of Stardust Systems, and Stardust Systems is not responsible for the content on any linked site. If you access a third-party website mentioned in this manual, then you do so at your own risk. Stardust Systems has provided these links only as a convenience, and the inclusion of the link does not imply that Stardust Systems endorses or accepts any responsibility for the content on those third-party sites.

The software described in this manual is provided under an agreement with Adobe Systems Incorporated, and such software can only be used in accordance with the terms of the agreement provided by Adobe Systems. Software code described and provided in this manual is provided under an agreement with Stardust Systems. The software can only be used in accordance with the terms of the agreement.

No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, photocopying, manual, optical, recording, or otherwise, outside the license agreement accompanying these materials, without the prior written permission of Stardust Systems. Stardust Systems claims copyright in this program and documentation as an unpublished work, revisions of which were first licensed on the date indicated in the foregoing notice. Claim of copyright does not imply waiver of other rights of Stardust Systems and its subsidiaries.

Information in this manual may change without notice and does not represent a commitment on the part of Stardust Systems.

NOTICE OF LIABILITY

This information in these training materials is distributed on an "AS IS" basis, without warranty of any kind, either express or implied. While every precaution has been taken in the preparation of these materials, neither Stardust Systems nor its licensors shall have any liability to any person or entity with respect to liability, loss, or damage caused or alleged to be caused directly or indirectly by the instructions contained in these materials or by the computer software and hardware products described herein.

First Edition: February 2011 - Stardust Systems - Cupertino, CA 95014 USA

Copyright © 2011 Stardust Systems. All rights reserved.

Introduction

This tutorial walks you step-by-step through the process of porting the Flex 3 Dashboard example application posted here on the Adobe Flex Developer's Center to Flex 4:

[Adobe Flex 3 Dashboard Application](#)

The Flex 3 Dashboard example application is a great showcase for Flex; representing data in charts, grids, forms, etc, with data presented in pods that can be dragged about the UI. It's a great starting point for incorporating Flex data visualization functionality in your applications.

The tutorial begins with the goal of importing the Flex 3 Dashboard into Flash Builder 4 and getting the application into a state where it can be launched, while changing as little as possible. Flash Builder uses the 4.x SDK, so importing and launching the Dashboard required a few changes. From there you progress top-down through the application container hierarchy converting the Dashboard to use the Flex 4 Spark containers and controls.

This tutorial will expose you to important aspects of Flex 4, such as the new Spark layout scheme and some of the new containers that come with Flex 4. You will also see how to create custom skin classes in MXML to implement styles previously done using CSS. Perhaps most important, you will see how to take an existing application created in Flex 3, and strategically convert the application step-by-step to Flex 4, replacing MX containers and components to the new Spark containers and components where possible.

Throughout the tutorial one goal will be to keep the application error-free and able to be launched. Nothing is worse than making your way through a lengthy tutorial, only to find that at the end nothing works and you have no idea how to fix things.

This tutorial is lengthy however. The Dashboard code base includes 18+ files, not including data and graphics files, and Flex 4 brings significant changes in the layout, skinning, and component architecture of Flex applications. Presenting the conversion of porting this application from Flex 3 to Flex 4 is difficult, and there are many steps, so proceed carefully and follow the instructions closely.

This tutorial does not attempt to improve the overall quality of the Dashboard application architecture. It also does not produce an application visually identical to the Flex 3 Dashboard application (though it comes very close). The primary goal is simply to provide an example of modifying a non-trivial application created in Flex 3 to Flex 4.

Not interested in going through the tutorial, and just want the final, ported Flex 4 application? You can experience the Flex 4 version of the Dashboard application, which you will create in this tutorial, here:

[Flex 4 Dashboard Application](#)

You can right-click the application, select View Source, and then view the source code online, or download the Flex project for import into Flash Builder.

Note: for some reason, the Flex 3 Dashboard application posted on the Adobe examples site displays differently when compared with the application when you download the source code from that exact site and compile it using the Flex 3 with SDK 3.4. This tutorial will convert the Flex 3 Dashboard application to Flex 4 so it displays as it does when you compile it using the Flex 3 SDK version 3.4, not as it displays as posted on the Adobe examples site (see figure 1).



<http://examples.adobe.com/flex3/devnet/dashboard/main.html>



Compiled from source code downloaded from examples.adobe.com

Figure 1. The Dashboard displays differently on the examples.adobe.com site.

As you proceed through this tutorial, if you have ideas or suggestions, or if you feel a section of instructions are unclear or insufficient, please do not hesitate to email us at customerservice@stardustsystems.com. As you might imagine, we cannot respond to questions about the tutorial, but you can always post questions on one of the many Flex forums.

Prerequisites

To complete this tutorial you need Flash Builder 4 installed (preferably Flash Builder 4.1, as it already has SDK 4.1), as well as the Flex 4.1 SDK (or a later version if one is available, though this tutorial has only been tested with SDK 4.1). Version 4.1 of the Flex SDK is necessary because it includes important improvements necessary to avoid run-time errors when launching the Dashboard application.

Getting Flash Builder 4

You can buy Flash Builder 4.1 here:

[Get Flash Builder 4.1](#)

If you're just checking out Flash Builder 4.1 you can download the trial version here:

[Flash Builder 4.1 Trial Version](#)

Getting the Flex SDK 4.1

If you have Flash Builder 4 and do not have SDK 4.1 installed, download the Flex SDK 4.1 or the latest SDK and add it to your Flash Builder 4:

[Flex 4.1 SDK](#)

In the "**Latest Milestone Release Builds**" section, download the latest "**Adobe Flex SDK**" and "**Adobe Add-ons**" files, but do not download the "**Open Source Flex SDK**" file.

After downloading the .zip files, create a new folder named "**4.1**" to the Flash Builder **sdks** directory (Windows example):

```
C:\Program Files\Adobe\Adobe Flash Builder 4\sdks
```

and then extract the contents of the two zip files to that folder.

Adding the Flex 4.1 SDK to Flash Builder

After installing the Flex 4.1 SDK to you need to add it to Flash Builder. In Flash Builder, select **Window – Preferences**, then expand the Flash Builder preferences category and click on the **Installed Flex SDKs** item (see figure 2):

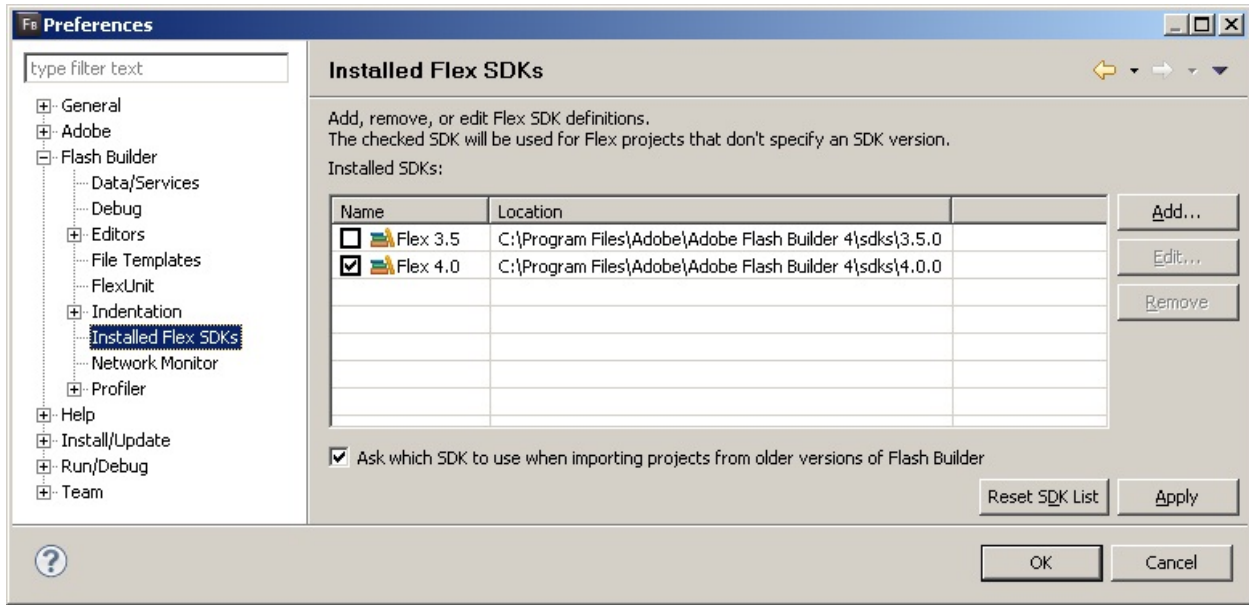


Figure 2. You add installed Flex SDKs in the Preferences window.

Click the **Add** button, navigate to the folder containing the Flex 4.1 SDK in the Flash Builder install, and click OK in the Add Flex SDK window (see figure 3).

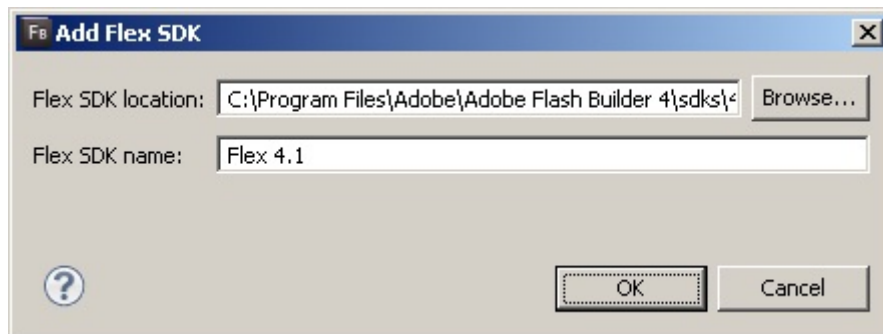


Figure 3. Selecting the Flex SDKs directory.

Ensure the "Ask which SDK to use when importing projects from older versions of Flash Builder" checkbox is checked, and then click OK in the Preferences window to finish adding the SDK.

Dashboard SharedObject File Location

While progressing through this tutorial, you might switch a pod to display a line chart instead of a column chart, change the position of pods within the Dashboard, minimize/maximize pods, etc, and run-time errors might occur from time to time, possibly putting the Dashboard in an unstable state.

The Dashboard application saves information on the location and the state of each pod to a Flex SharedObject, which is similar to the cookies web pages use. Flex SharedObjects are files saved to the hard drive and have a file extension of .sol. You can reset the Dashboard to its initial state by deleting the SharedObject file on your machine. This resets the state of each pod and also the position of each pod within the Dashboard.

Please remember this, because deleting this SharedObject can be important when things just don't look right, and it does no harm. If you also have Flex Builder 3 installed and have imported, compiled, and launched the Flex 3 Dashboard for comparison, you will also want to delete that SharedObject (stored in a different location). The location of the Dashboard application SharedObject is OS dependent:

Operating System	Location
Windows 95/98/ ME/2000/XP	c:/Documents and Settings/username/Application Data/Macromedia/ Flash Player/#SharedObjects
Windows Vista	c:/Users/username/AppData/Roaming/Macromedia/Flash Player/#SharedObjects
Macintosh OS X	/Users/username/Library/Preferences/Macromedia/FlashPlayer/#SharedObjects/ web_domain/path_to_application/application_name/object_name.sol
Linux/Unix	/home/username/.macromedia/Flash_Player/#SharedObjects/ web_domain/path_to_application/application_name/object_name.sol

As an example, here is the location of the Flex 3 and Flex 4 SharedObjects on the test machine used to create this tutorial:

Flex 3 Dashboard (imported into Flex Builder 3):

C:\Users\Greg\AppData\Roaming\Macromedia\Flash Player\#SharedObjects\5QRD7XDU\localhost\Users\Greg\Documents\
Flex Builder 3\Dashboard\bin-debug\main.swf\com.esria.sample.dashboard.sol

Flex 4 Dashboard (imported into Flash Builder 4):

C:\Users\Greg\AppData\Roaming\Macromedia\Flash Player\#SharedObjects\5QRD7XDU\localhost\Flex4Dashboard\
Dashboard\bin-debug\main.swf\com.esria.sample.dashboard.sol

General Code Conversion Approach

The general approach for conversion of the Dashboard code is to proceed from the top application object and proceed down through the various levels of the container hierarchy. However, changes in one file often make it necessary to make changes in other files to avoid errors and still be able to launch the application.

Other times, to keep things simple or to reduce the level of confusion, you may switch between editing a few, related files. Still, for the most part the tutorial leaves few, if any, loose threads.

As much as possible, while porting the Dashboard to Flex 4, this tutorial will attempt to maintain the application in a state where it can be launched, with no compile errors (warnings are acceptable). Occasionally you may experience run-time errors, but at times this will be unavoidable, and this too will be kept to a minimum.

Some sections include many steps for converting code, and missing or improperly applying one step can make it difficult to proceed. Take your time, read each step carefully, and keep in mind these hints on the step-by-step instructions:

- When "replacing" one MXML tag with another, only replace the tag name and namespace prefix, not the entire tag, unless instructions indicate to do so.
- Many steps are repeated in multiple files throughout this article series, so subsequent instructions may simplify steps you have already performed.
- The reference point for steps will often be a function name, the id of an MXML tag, or more generalized text such as "near the top of the file", etc.

Having Flex Builder 3 also installed, with the Flex 3 Dashboard project imported and compiling successfully, is highly recommended, as you will want to frequently compare the UI and functionality of the Flex 4 version you create with the Flex 3 version. You should have the Flex SDK version 3.4 installed.

Importing the Dashboard Project

To begin the tutorial, you will download the Flex 3 Dashboard project archive .zip file, and then import the project into Flash Builder 4.

Downloading the Flex 3 Dashboard Archive

Visit the link below and click the link in the lower-left corner of the page to download the Flex 3 Dashboard source code archive zip file. You do not need to download the Flex 3 SDK.

[Flex 3 Dashboard Source Code](#)

Note: after importing the Flex 3 Dashboard application into Flash Builder and making initial changes to fix any errors and launch the application, save the imported project files to another location for reference.

After downloading the Flex 3 Dashboard zip file, there are usually two steps to importing the project:

1. Select to import from an archive file or from a project folder.
2. Select the Flex SDK version for the project.

Selecting to Import from Archive File or Project Folder

Importing directly from the Dashboard zip archive file is usually preferable to unzipping the file and then importing. In Flash Builder, select **File – Import – Flash Builder Project...** and select to import the Dashboard project using the zip file archive, or using the project folder if you unzipped the archive somewhere (see Figure 4).

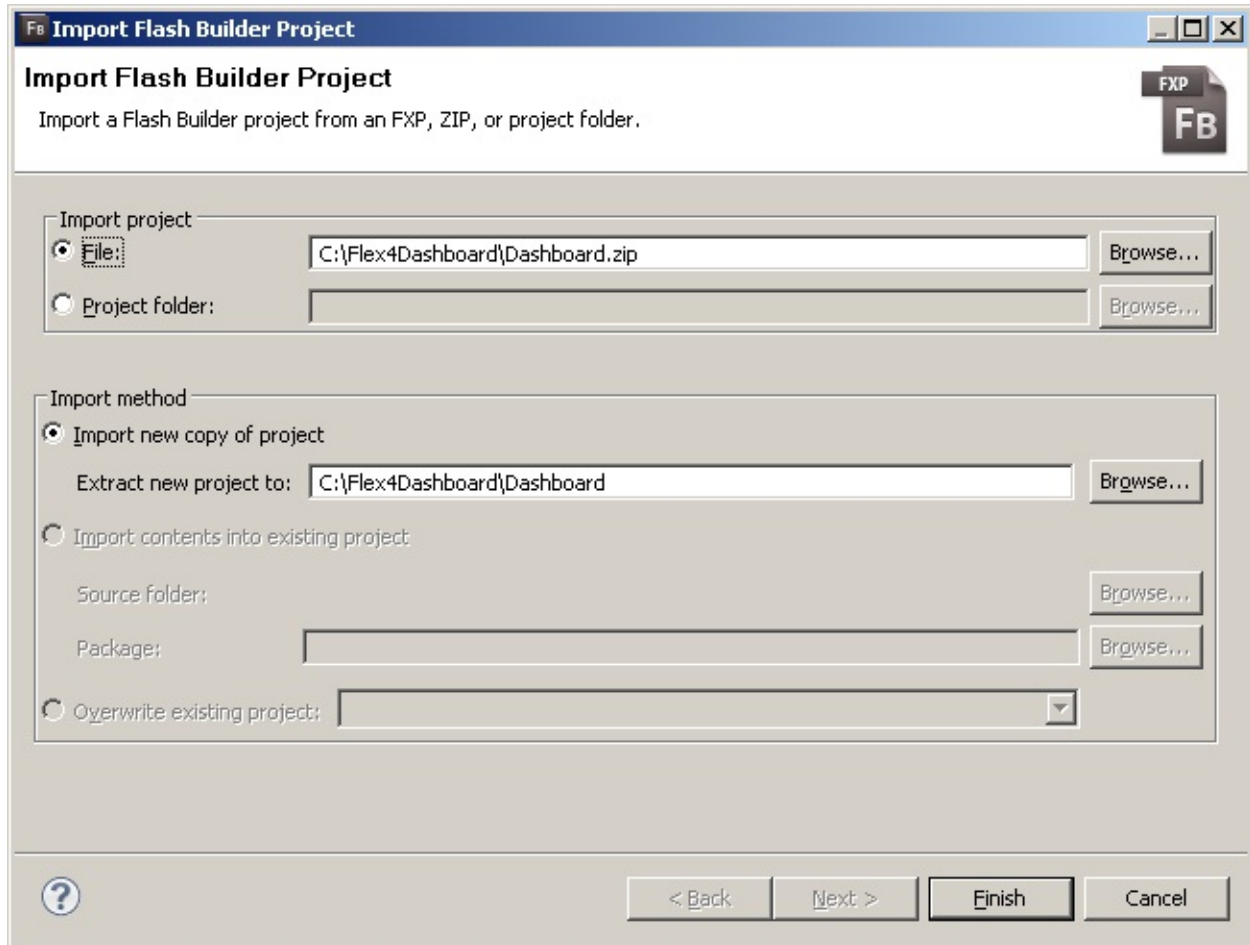


Figure 4. You can select to import from an archive file or from a project folder.

You may use either method. Click the **Finish** button after making your selection.

Note: in rare cases, the Import Flash Builder Project may display an error that the zip archive or project folder containing the unzipped files does not contain a valid project. In that case, unzip the archive folder somewhere if you have not already done so. Then select **File – New – Flex Project**, name the project **Dashboard**, and select the unzipped Dashboard project folder. Click Next until you select **main.mxml** as the main application file, then click **Finish**.

Selecting the Flex SDK Version

After clicking the Finish button, you should be presented with a window prompting you to choose the Flex SDK version for the project (see Figure 5). The Dashboard project requires Flex SDK 4.1 (or a later version if one is available, though this tutorial has only been tested with SDK 4.1).

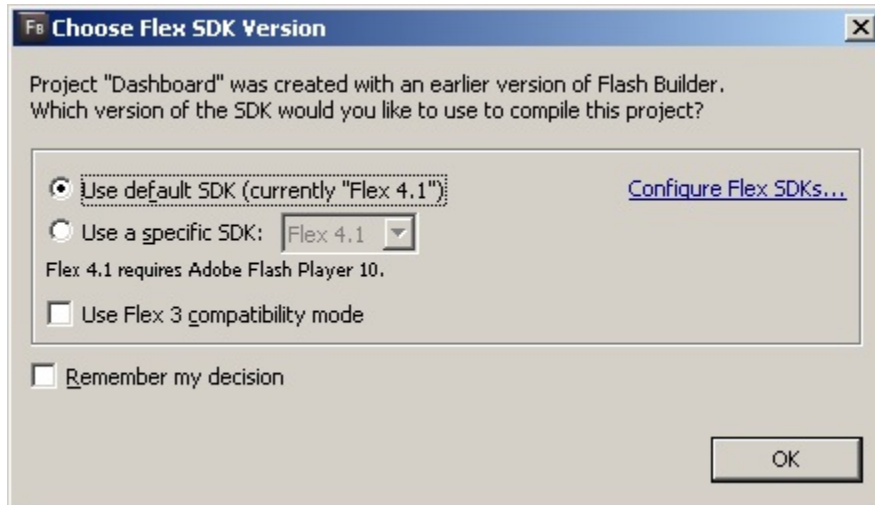


Figure 5. You should choose Flex SDK 4.1 or later.

If your default SDK is not Flex 4.1, select the **"Use a specific SDK"** radio button and select the Flex 4.1 SDK. Ensure the checkbox labeled **"Use Flex 3 compatibility mode"** is NOT selected, because you will be converting the application to Flex 4. Click OK to continue.

If you see the a warning that the project will be upgraded (see Figure 6), just click OK.



Figure 6. You can just click OK if you see the project upgrade warning.

Following the import you should see the project source file tree in Flash Builder 4 (see Figure 7):

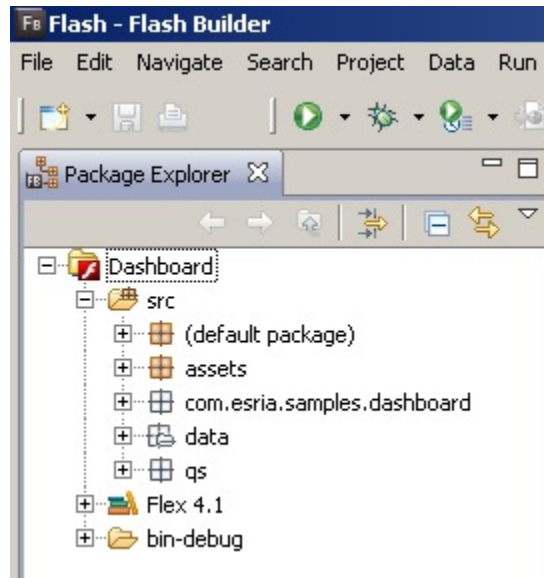


Figure 7. The project source file tree after import.

Fixing Initial Errors and Warnings

Following a successful import of the project, you will need to make several code changes to correct for differences between the Flex 3 SDK and Flex 4 SDK. Note: some errors mentioned in later sub-sections will not be displayed until the first few errors have been addressed, so you may not see them yet.

Adding an Empty libs Folder

The first error complains of a missing libs folder (see Figure 8):

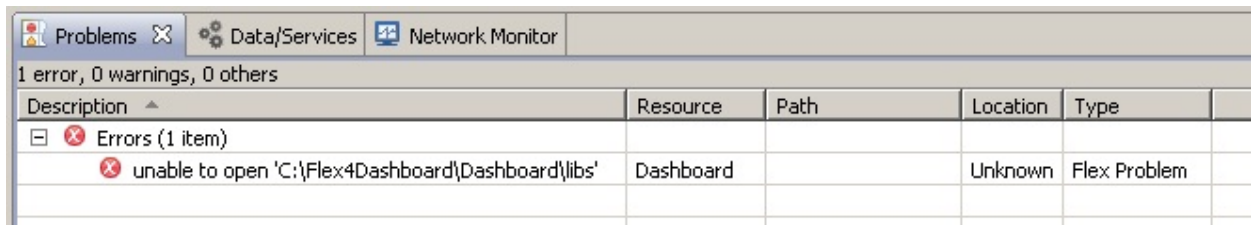


Figure 8. The libs folder is missing after importing the project, which causes an error.

The libs folder is created automatically by Flash Builder for new projects, but for imported projects that do not contain a libs folder, the folder must be created manually. Simply right click the project top-level folder "Dashboard", select **New – Folder**, and create a new folder named "libs" (see Figure 9). The folder can remain empty but it must exist to avoid the error.

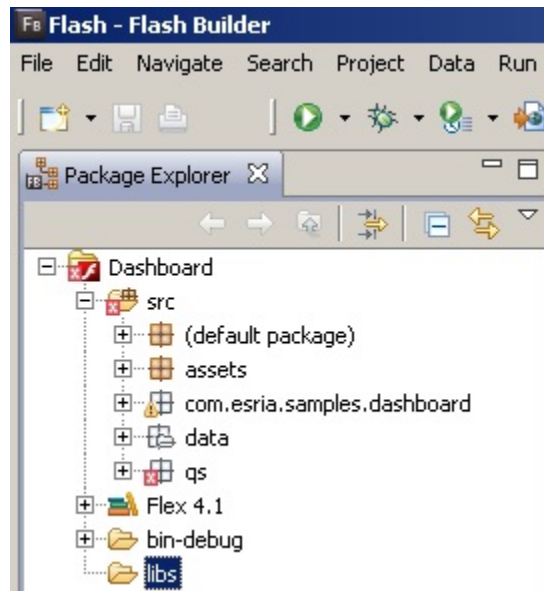
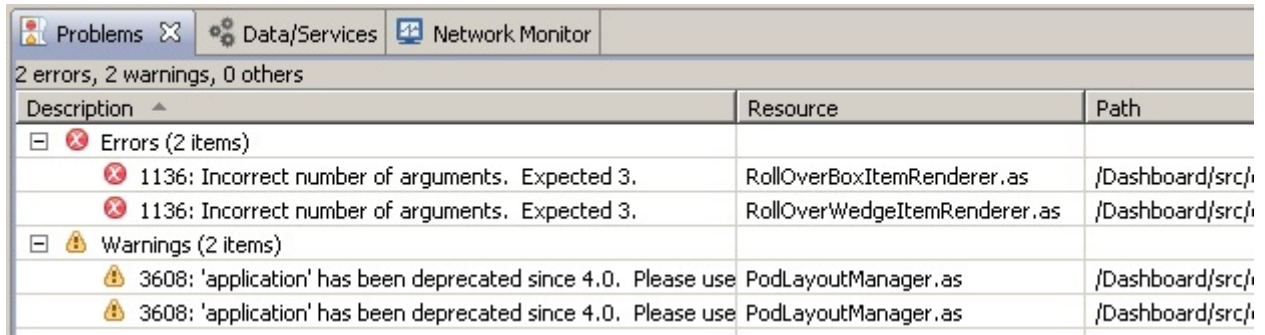


Figure 9. The libs folder has been created manually to eliminate an error.

SDK Related Errors and Warnings

After adding the "libs" folder the error for the missing folder should be gone, and you should have two new errors and two warnings (see Figure 10):



The screenshot shows the 'Problems' window in an IDE. The title bar includes 'Problems', 'Data/Services', and 'Network Monitor'. Below the title bar, it says '2 errors, 2 warnings, 0 others'. The main area is a table with three columns: 'Description', 'Resource', and 'Path'. There are four rows of items:

Description	Resource	Path
Errors (2 items)		
1136: Incorrect number of arguments. Expected 3.	RollOverBoxItemRenderer.as	/Dashboard/src/...
1136: Incorrect number of arguments. Expected 3.	RollOverWedgeItemRenderer.as	/Dashboard/src/...
Warnings (2 items)		
3608: 'application' has been deprecated since 4.0. Please use	PodLayoutManager.as	/Dashboard/src/...
3608: 'application' has been deprecated since 4.0. Please use	PodLayoutManager.as	/Dashboard/src/...

Figure 10. Two new errors and two warnings occur after fixing the libs folder error.

The two warnings occur because in the Flex 4 SDK you reference the top-level application object and methods differently. In Flex 3 you would use the `mx.core.Application` class:

```
Application.application.MY_TOP_VAR_OR_METHOD
```

In Flex 4 you use the new `mx.core.FlexGlobals` class:

```
FlexGlobals.topLevelApplication.MY_TOP_VAR_OR_METHOD
```

To eliminate the two warnings:

1. At the top of file `src\com\esri\samples\dashboard\managers\PodLayoutManager.as`:

replace this line: `import mx.core.Application;`

with this line: `import mx.core.FlexGlobals;`

2. In the file where the warnings occur:

replace: `Application.application`

with this: `FlexGlobals.topLevelApplication`

After making these two changes the warnings no longer be present.

The two errors are due to API changes in Flex 3.5 (the Dashboard application was originally created using an earlier version of the Flex 3 SDK, before SDK 3.5 was released). The changes involve the `IStroke` and `IFill` interfaces. The problem is that the number of arguments passed to two methods defined in these interfaces has changed.

Double click each error and the relevant Dashboard code file will open at the line where you need to make the change to eliminate the error.

Dashboard/src/qs/charts/renderers/RollOverBoxItemRenderer.as

This first error occurs because the `IStroke` interface method **apply()**, which prior to Flex 3.5 took a single argument now takes three arguments:

replace this: `stroke.apply(g);`

with this: `stroke.apply(g, rc, new Point(rc.left,rc.top));`

Then add this line to the top of the file with the other imports:

```
import flash.geom.Point;
```

Dashboard/src/qs/charts/renderers/RollOverWedgeltemRenderer.as

This next error occurs because the `IFill` interface method `begin()`, which prior to Flex 3.5 took two arguments now takes three arguments:

replace this: `f.begin(g,rc);`

with this: `f.begin(g,rc, new Point(rc.left,rc.top));`

Re-create the HTML Templates

If you get a new error that the HTML wrapper cannot be created, simply right click the error message and select "**Recreate HTML Templates**" (see Figure 11).

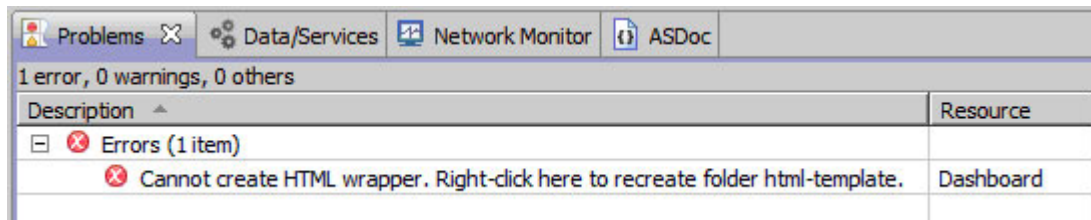


Figure 11. An error related to the HTML wrapper.

You should now have new errors and warnings. The next section will eliminate these, allowing the application to be launched for the first time in Flash Builder (see Figure 12).

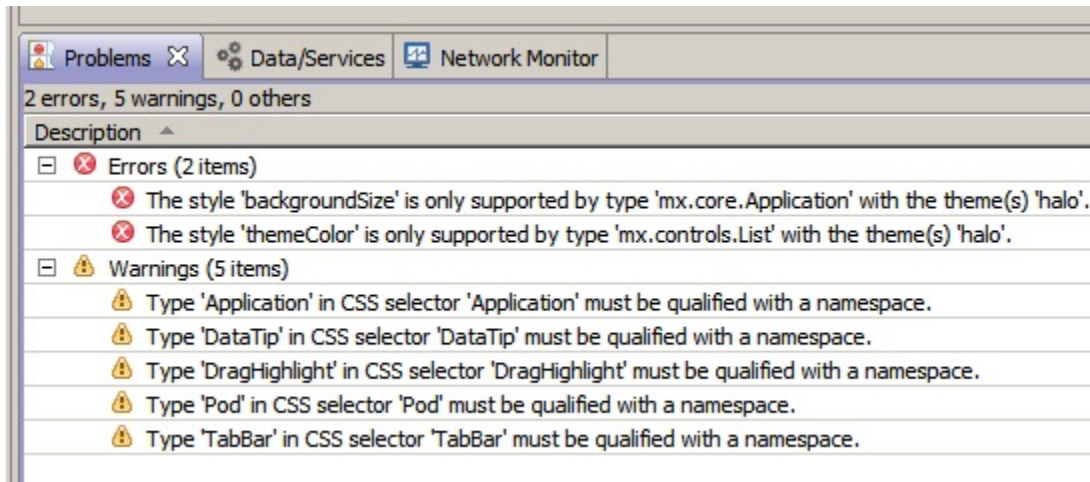


Figure 12. New errors and warnings.

Initial Changes to main.mxml

In this section you make initial changes to the main application file main.mxml to eliminate new errors so the application can be launched for the first time in Flash Builder. You also begin converting the application containers to use the new Spark containers.

Remove the Style Tag

The styles currently implemented in file src/assets/styles.css will be implemented in the Flex 4 version of the Dashboard in custom skin class, so first you will disassociate the application from that CSS file. Open the main application file src/main.mxml in the default package folder and delete this line:

```
<mx:Style source="/assets/styles.css" />
```

Do not delete file styles.css as you will refer to it while creating custom skins.

Changes to Use the Spark Application Container

Next, you make changes to use the new Spark application container instead of the Flex 3 MX application container.

1. In main.mxml change the namespace prefix in the opening and closing Application tags from mx to s, to use the new Spark application class:

```
<mx:Application          <s:Application
```

2. The Flex 3 Dashboard `<mx:Application>` tag defaults to vertical layout, but the Flex 4 Spark `<s:Application>` tag defaults to BasicLayout (the Flex 4 equivalent of absolute layout) so add the following in main.mxml just before the `<mx:TabBar>` component:

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

Change Root Tag Namespaces

Then replace the one Flex 3 namespace definition with the three Flex 4 namespace definitions in the opening Application tag:

replace this:

```
xmlns:mx="http://www.adobe.com/2006/mxml"
```

with these:

```
xmlns:fx="http://ns.adobe.com/mxml/2009"  
xmlns:s="library://ns.adobe.com/flex/spark"  
xmlns:mx="library://ns.adobe.com/flex/mx"
```

Script Tag Namespace Prefix Change

Change the namespace prefix in the opening and closing `<mx:Script>` tags to use the Flex 4 `fx` namespace prefix instead of the `mx` namespace prefix.

```
<mx:Script                <fx:Script
```

Remove Style Property Settings in the Application Container

Several style properties currently set in the Application tag will be set later in custom skins, or will be moved to the layout class.

1. Remove the `backgroundSize` and `backgroundColor` styles from the `<s:Application>` tag.
2. Move the `horizontalAlign`, `paddingLeft`, `paddingRight`, `paddingBottom`, and `paddingTop` style properties from `<s:Application>` to the `<s:VerticalLayout>` tag.
3. Remove the `themeColor` style property from the `<mx:List>` tag in file `src/com/esria/samples/dashboard/view/ListContent.mxml`.

Note: you can double-click the `themeColor` error in the **Problems** view in Flash Builder to automatically open the `ListContent.mxml` file to remove the style property and eliminate the error.

Re-create the HTML Templates

If you get a new error that the HTML wrapper cannot be created, simply right click the error message and select **"Recreate HTML Templates"** (see Figure 13).

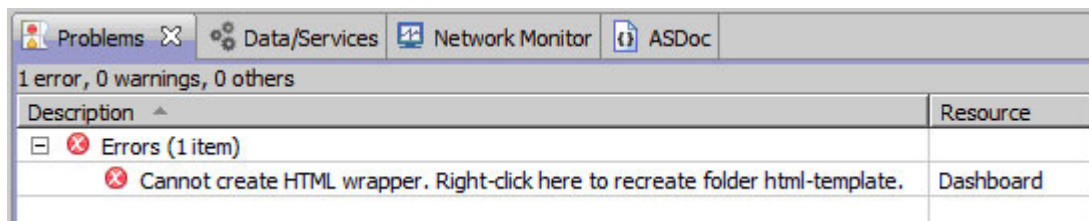


Figure 13. An error related to the HTML wrapper.

Launch the Dashboard Application

Now you should be able to compile and launch the Dashboard application with no errors. Compared with the Flex 3 version, the application has some display issues, such as the background image not displaying (see Figure 14), but these will be eliminated as you continue to port the application to Flex 4.

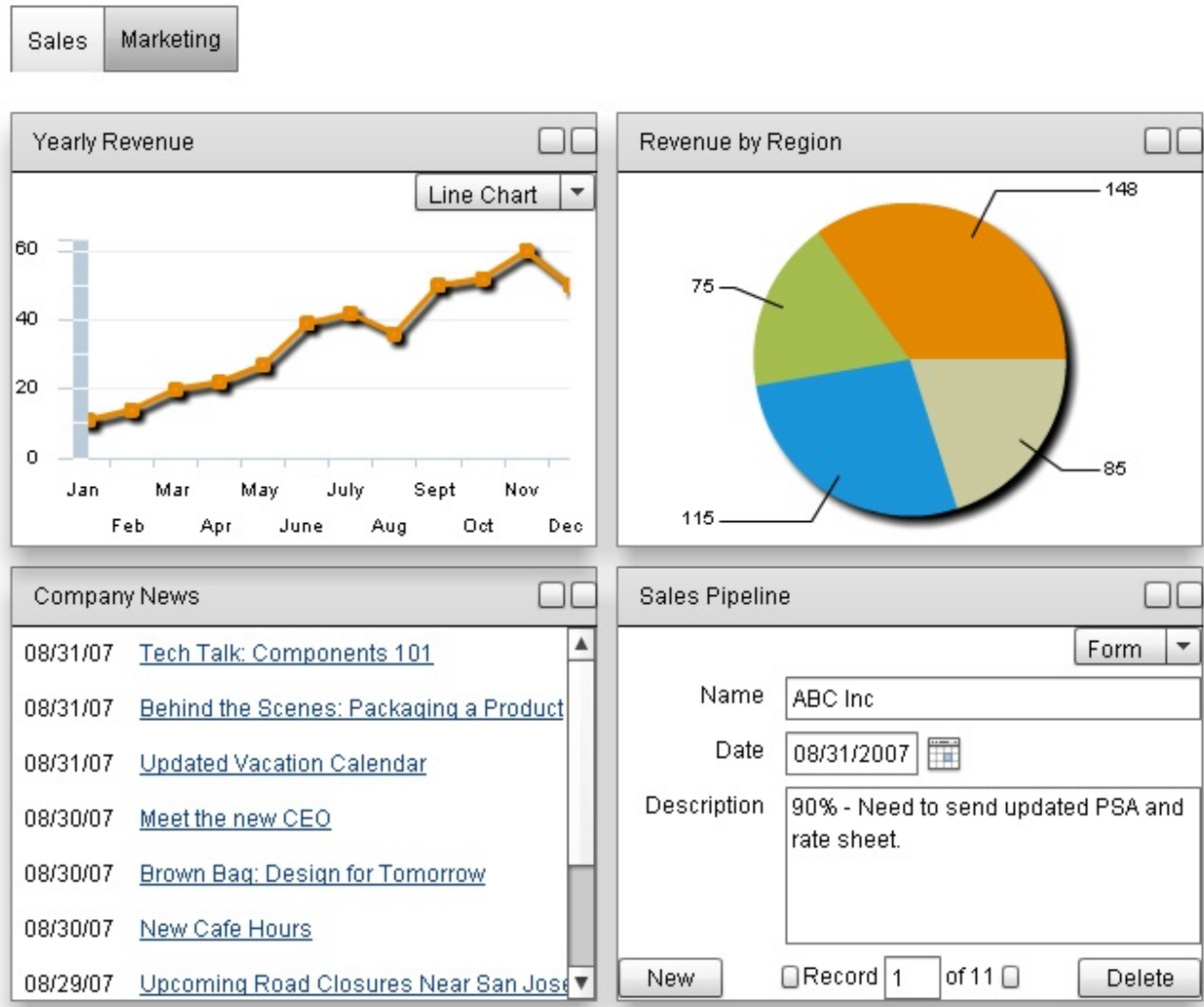


Figure 14. The application launches, but has some display issues (partial screenshot).

Compare the previous graphic with the Flex 3 Dashboard application (see Figure 15).

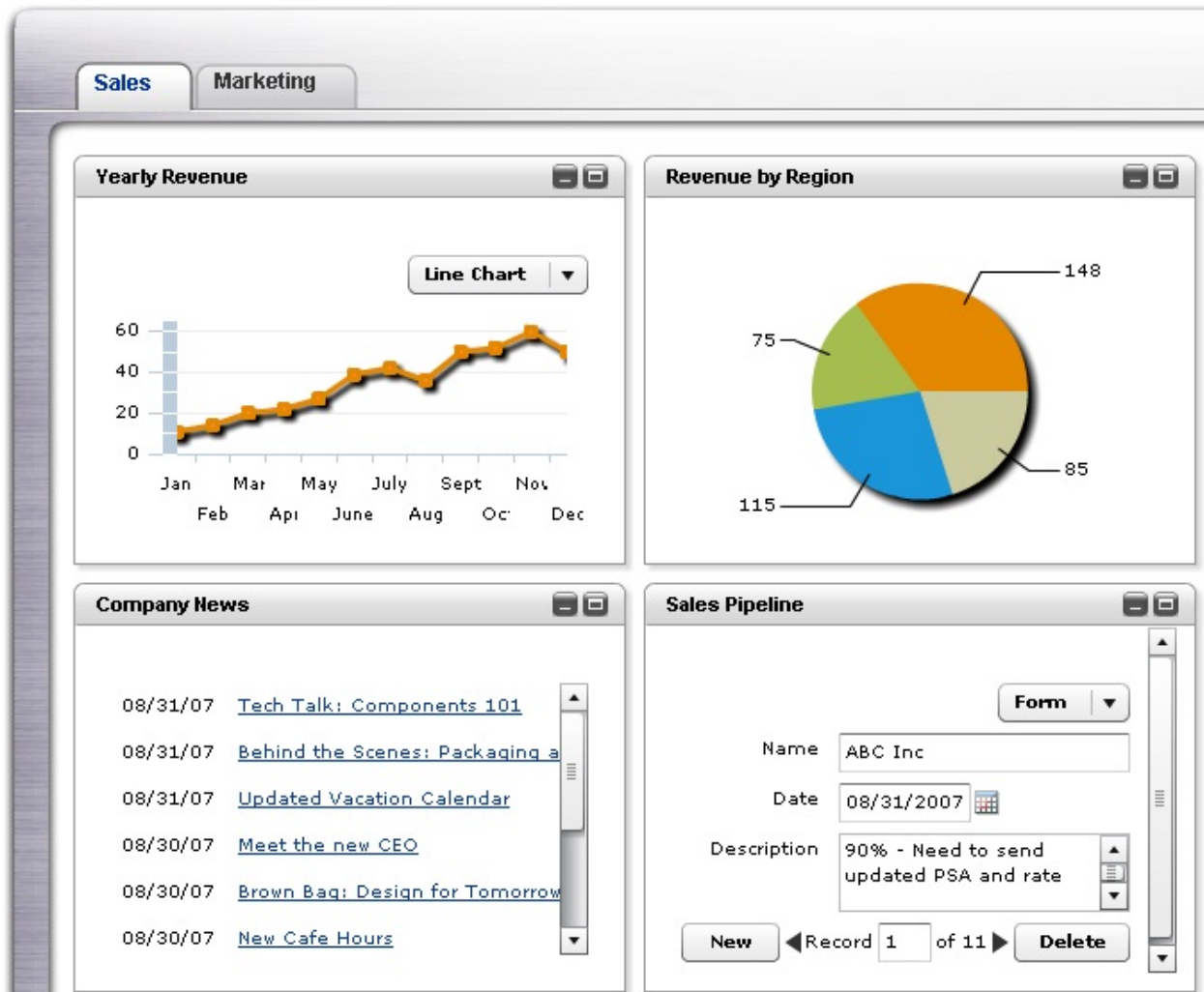


Figure 15. The Flex 3 Dashboard application (partial screenshot).

Now that initial errors have been eliminated, allowing the Dashboard application to be launched, the next step is to implement a custom skin for the Application container.

Application Container Custom Skin

The various styles defined in the styles.css file in the Flex 3 Dashboard will be implemented in custom skin classes in the Flex 4 version of the application. The first custom skin class you create will be for the Application container, and will restore the application background image.

Create a Folder for Custom Skins

First create a new folder named "skins" for custom skins by selecting the "**src**" folder in the Flash Builder **Package Explorer** and then selecting **File - New – Folder**. It should be at the same level as the default package, assets folder, and the other application folders (see figure 16).

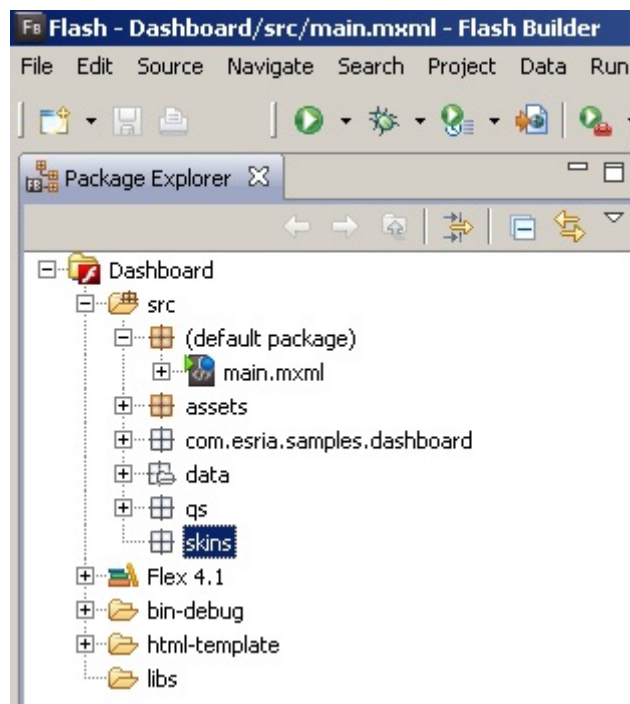


Figure 16. The skins folder will contain custom skins for the Dashboard application.

Initially Create the Custom Skin Class

Next, ensure the skins folder is highlighted, select **File – New – MXML Skin**, and in the New MXML Skin window enter values in the "**Name**", "**Host component**", and "**Create as copy of**" text inputs so the window appears as in figure 17. Ensure the "**Remove ActionScript styling code**" checkbox is **checked**.

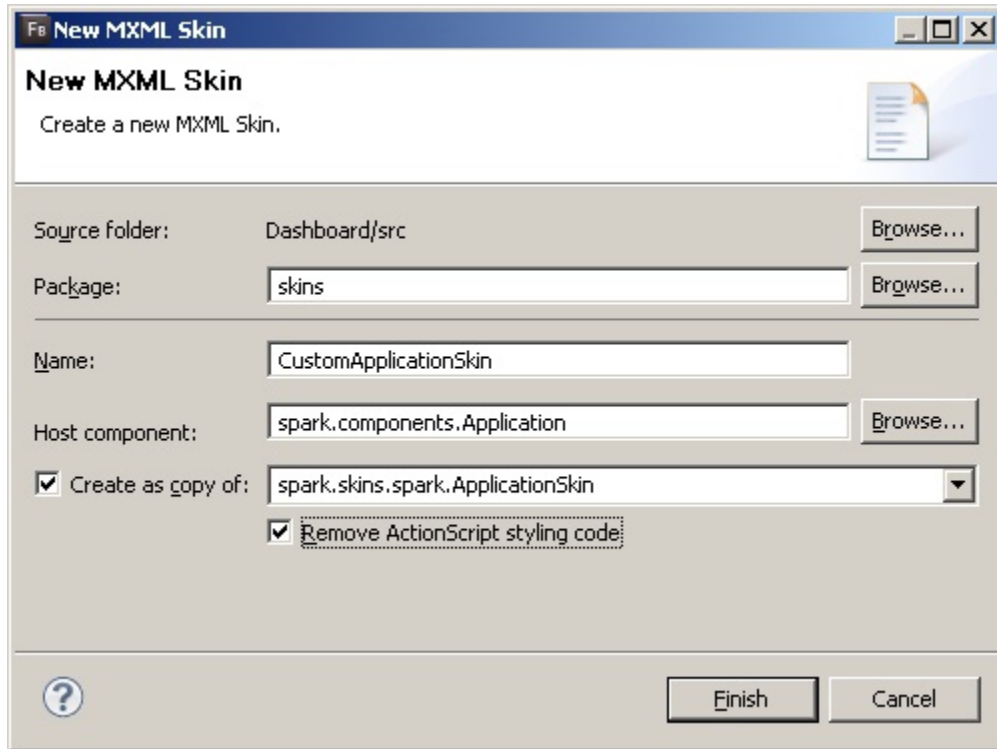


Figure 17. The Application container skin is created for the Dashboard application.

After clicking the Finish button, a new file CustomApplicationSkin.mxml is placed in the src/skins folder.

Apply the Custom Skin File to the Application

To use this custom skin class, which you have not modified yet, add the following to the <s:Application> tag at the top of file main.mxml:

```
skinClass="skins.CustomApplicationSkin"
```

Implement Styles in the Custom Skin

Next you modify the skin class to replace styles previously defined in styles.css, and also the backgroundSize style property removed in an earlier section. Refer to the Application type selector in styles.css, as its styles will be implemented in the skin.

To implement the background image in the skin, open file skins/CustomApplicationSkin.mxml and locate the Rect with the id "backgroundRect". Replace the entire Rect tag and its contents with the following BitmapImage tag:

```
<s:BitmapImage top="0" bottom="0" left="0" right="0"
  source="@Embed(source='/assets/application_background.png',
  scaleGridTop='94', scaleGridBottom='791',
  scaleGridLeft='350', scaleGridRight='690')"/>
```

Notice that this skin sets the `top`, `bottom`, `left` and `right` constraint properties to 0, which causes the image to fill the entire application background, and replaces the `backgroundSize` style property removed from the `<mx:Application>` tag. The `background-color` and `background-gradient-colors` styles in `styles.css` and the `Application` tag are unnecessary because the default Flex 4 Spark application background is white. The `color` style in `styles.css` is unnecessary, because the default text color for the application is black.

Launch the Application

Launch the application and you will see the background image applied to the application as it is for the Flex 3 Dashboard application (see figure 18).

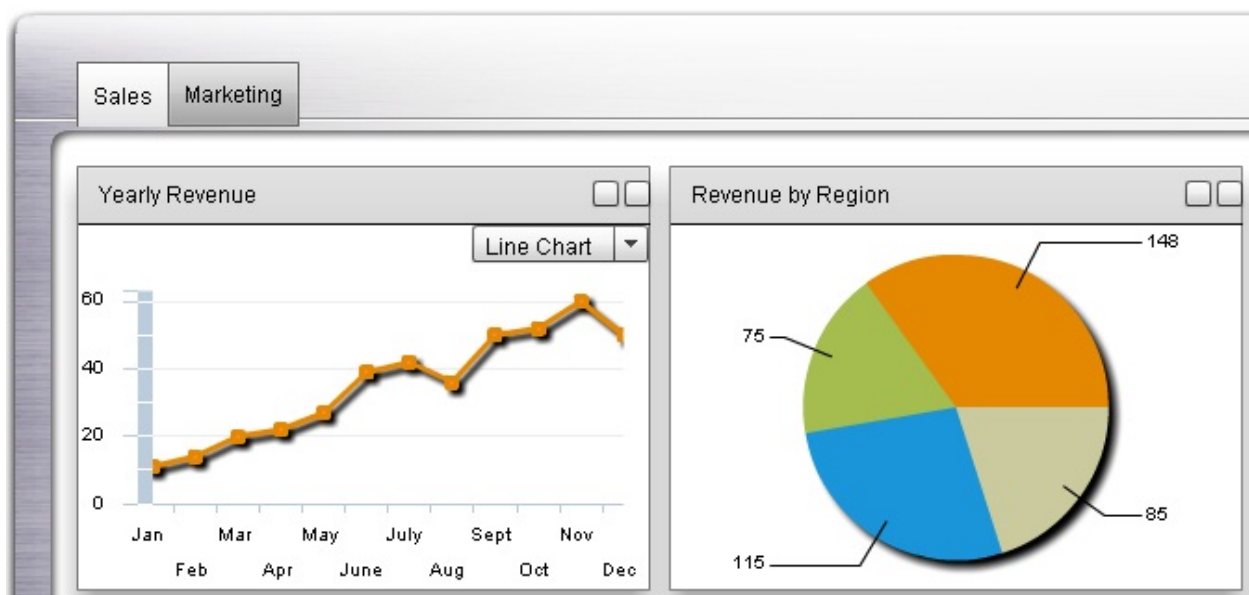


Figure 18. The application background image displays properly (partial screenshot).

Next you will make changes so the application uses the new Spark TabBar component.

Using the Spark TabBar Component

Now you will make initial changes to use the new Flex 4 Spark TabBar component, instead of the Flex 3 MX TabBar component seen in the Flex 3 Dashboard application.

Initial Changes to Use the Spark TabBar

Make the following changes in main.mxml:

1. Change the namespace prefix of the **TabBar** component tag to use the Spark `s` namespace prefix instead of the `mx` prefix.
2. Enclose the TabBar `dataProvider` value in binding curly braces `{viewStack}`.
3. Edit the TabBar to respond to the `change` event instead of to the `itemClick` event.

After completing these steps, the TabBar tag should look similar to the following:

```
<s:TabBar id="tabBar"
  change="onItemClickTabBar(event)"
  height="35" dataProvider="{viewStack}" />
```

Additional Changes Related to Events

Now you need to make additional changes necessary because the Spark TabBar control dispatches a different event when the selected tab is changed.

1. Replace the event import at the top of main.mxml:
replace this: `import mx.events.ItemClickEvent;`
with this: `import spark.events.IndexChangeEvent;`
2. Next search for the definition of function `onItemClickTabBar()` in main.mxml and in the function signature, replace `ItemClickEvent` with `IndexChangeEvent`:

```
private function onItemClickTabBar(e:IndexChangeEvent):void
```
3. Then within the function definition change `e.index` to `e.newIndex` because the `IndexChangeEvent` provides access to the clicked tab through a different property.
4. Next change the following line of code just above the `onItemClickTabBar()` function definition (near the bottom of the `onResultHttpService()` function definition). This is necessary because the code creates an event instance to simulate clicking a tab when the application first launches.

change this:

```
onItemClickTabBar(new ItemClickEvent(ItemClickEvent.ITEM_CLICK, false, false, null, index));
```

to this:

```
onItemClickTabBar(new IndexChangeEvent(IndexChangeEvent.CHANGE, false, false, -1, index));
```

Launch the Application

Now you should have no errors and you should still be able to launch the application. Because the Spark TabBar is now being used, the tabs display differently. Most notable is that the Spark TabBar has the currently selected tab rendered in a darker color than any unselected tabs, while the MX TabBar component renders the selected tab in the lighter color (see figure 19).



Figure 19. *The Spark TabBar renders differently from the MX TabBar component.*

Next you will implement a custom skin for the TabBar, and then a custom skin for the tabs themselves, so the Dashboard application TabBar displays as it does in the Flex 3 version.

TabBar Custom Skin

This section and the next implement two custom skins so the tabs in the TabBar component display as they do in the Flex 3 Dashboard application, one skin for the TabBar component, and one skin for the ButtonBarButton components making up the tabs.

Initially Create the Custom Skin Class

To create the TabBar custom skin, ensure the skins folder is selected, then select **File – New – MXML Skin** and enter the following values in the window:

Name: CustomTabBarSkin
Host component: spark.components.TabBar
Create as copy of: spark.skins.spark.TabBarSkin

Ensure the **"Remove ActionScript styling code"** checkbox is **checked**.

A new file CustomTabBarSkin.mxml should be placed in the `src/skins` folder.

Apply the Custom Skin File to the TabBar

To use this custom skin class, which you have not modified yet, add the following to the `<s:TabBar>` tag at the bottom of file main.mxml:

```
skinClass="skins.CustomTabBarSkin"
```

Remove the Blue Focus Rectangle

Also add `focusThickness="0"` to the skin file `<s:ButtonBarButton>` tag, so the tabs do not display a focus border when leaving and returning to the Dashboard window.

This custom TabBar skin by itself does not change the appearance of the tabs. For that you need to create a custom skin class for the tabs themselves, which you do next.

ButtonBarButton Custom Skin

The TabBar uses instances of the Spark ButtonBarButton component for the individual tabs. A separate custom skin is required for the ButtonBarButton sub-components, which the TabBar custom skin will reference. There are more steps to create this skin, so follow along carefully.

Initially Create the Custom Skin Class

To create the ButtonBarButton custom skin, ensure the skins folder is selected, then select **File – New – MXML Skin**, and enter the following values:

Notice that the value below for the **Create as copy of:** field is not the default class supplied by Flash Builder (**ToggleButtonSkin**), but is **TabBarButtonSkin**.

Name: CustomBarButtonSkin
Host component: spark.components.ButtonBarButton
Create as copy of: spark.skins.spark.TabBarButtonSkin

For this skin, ensure the "Remove ActionScript styling code" checkbox is unchecked.

CustomBarButtonSkin.mxml should be placed in the src/skins folder.

Apply the Custom Skin File in the Custom TabBar Skin

To use this custom skin class, open file skins/CustomTabBarSkin.mxml, and in the `<s:BarButton>` tag at the bottom of the file:

replace this: spark.skins.spark.TabBarButtonSkin
with this: skins.CustomButtonBarButtonSkin

Implement General Styles in the Custom Skin

Next you apply some of the more general styles that were defined in styles.css, specifically the `tabBarButton`, and `tabBarButtonSelected` class selectors.

1. Open CustomBarButtonSkin.mxml and in the "labelDisplay" Label change the `textAlign` property value from `center` to `left`.
2. Implement the padding styles using the `labelDisplay left`, `right`, and `top` style properties. Keep the `left` property value of 10, but change the `right` style property to a value of 20, and change the `top` property to a value of -4.

3. Next add the following style properties to the `labelDisplay Label` tag:

```
color="#333333" fontSize="12" fontFamily="arial"
```

4. Then add `fontWeight="bold"` to the `Label` tag, because the Flex 3 `Label` control text is bold by default, but in Flex 4 the default `fontWeight` is "normal".

Implement Text Color Styles in the Custom Skin

The last three general style properties to implement require an explanation. In the `tabBarButton` style in `file styles.css`, the `text-roll-over-color` property represents the tab text color when the mouse hovers over an unselected tab. In the `tabBarButtonSelected` style, the `text-roll-over-color` property represents the tab text color when the mouse hovers over a tab that is selected, and the `color` property represents the text color when the mouse is not hovering over a selected tab. Although the `text-roll-over-color` style property is used twice in `styles.css`, it is used in two different styles, so there is no conflict. To achieve the same goal in the `CustomBarButtonSkin`, add the following three skin state style property settings to the `labelDisplay Label`:

```
color.upAndSelected="#003399"  
color.overAndSelected="#003399"  
color.over="#858585"
```

The `labelDisplay Label` tag should now look similar to the following:

```
<s:Label id="labelDisplay"  
  textAlign="left"  
  verticalAlign="middle"  
  maxDisplayedLines="1"  
  horizontalCenter="0" verticalCenter="1"  
  left="10" right="20" top="-4" bottom="2"  
  color="#333333" fontSize="12" fontFamily="arial"  
  fontWeight="bold"  
  color.upAndSelected="#003399"  
  color.overAndSelected="#003399"  
  color.over="#858585">  
</s:Label>
```

Implement Graphic Image Styles in the Custom Skin

Lastly you apply the six graphic image styles for the actual tab display. Only two images are used, one image for the `up`, `over`, and `down` states of non-selected tabs, and another for the `up`, `over`, and `down` states of the currently selected tab.

1. First delete everything in the `CustomBarButtonSkin <fx:Script>` tag contents except the following code, which should remain:

```

static private const exclusions:Array = ["labelDisplay"];

override public function get colorizeExclusions():Array {return exclusions;}

override protected function initializationComplete():void
{
    useChromeColor = true;
    super.initializationComplete();
}

```

- Next, delete all of the graphical content up to and including layer 7, so basically you just delete everything from just after the `</s:states>` closing tag to the end of layer 7. Layer 8 containing the `labelDisplay` will remain.
- Then add the following just before the layer 8 `labelDisplay` code, for two `<s:BitmapImage>` components, one for each of the tab images.

```

<s:BitmapImage source="@Embed('/assets/tab_up.png', scaleGridTop='11',
scaleGridBottom='30', scaleGridLeft='10', scaleGridRight='92')"
includeIn="up, over, down" width="{labelDisplayWidth}"/>

```

```

<s:BitmapImage source="@Embed('/assets/tab_selected.png',
scaleGridTop='13', scaleGridBottom='31', scaleGridLeft='9',
scaleGridRight='91')" includeIn="upAndSelected,
overAndSelected, downAndSelected" width="{labelDisplayWidth}"/>

```

Notice the `includeIn` properties, which allows you to specify the states for which each image should display. Also notice that the `width` is set using binding to a variable named `labelDisplayWidth`, which is necessary so the `TabBar` tabs adjust their size as appropriate for the text of each tab. You should have two errors, which will be fixed by the following changes.

- Add the following code to declare the `labelDisplayWidth` variable, and a method to set its value, at the beginning of the `<fx:Script>` tag:

```

[Bindable] private var labelDisplayWidth:Number = 0;

private function setTabWidth():void{
    labelDisplayWidth = labelDisplay.width + labelDisplay.left
        + labelDisplay.right + 3;
}

```

Be careful when pasting the above code, because Flash Builder occasionally gets confused and messes up pasted code. You may need to select **Project – Clean...** from the menu bar to eliminate invalid errors.

5. Then add the following code to the end of the `labelDisplay` `<s:Label>` tag:

```
creationComplete="setTabWidth();" 
```

Adjust the Position of Tab Text in the Custom Skin

Launch the application and notice the tab text is not positioned exactly as it is for the Flex 3 Dashboard application. In the Flex 4 Dashboard, the text is slightly lower than in the Flex 3 Dashboard. (see figure 20).



Figure 20. The tab text is positioned a bit wrong compared to the Flex 3 Dashboard.

1. To correct this minor difference, delete the `horizontalCenter` and `verticalCenter` `labelDisplay` properties, then change the `left` property to 12, and replace the `top` property with `top.selectedStates="0" top.nonSelected="-2"` so the `labelDisplay` `Label` tag is similar to the following:

```
<s:Label id="labelDisplay"
  textAlign="left"
  verticalAlign="middle"
  maxDisplayedLines="1"
  left="12" right="20"
  top.selectedStates="0" top.nonSelected="-2" bottom="2"
  color="#333333" fontSize="12" fontFamily="arial"
  fontWeight="bold"
  color.upAndSelected="#003399"
  color.overAndSelected="#003399"
  color.over="#858585"
  creationComplete="setTabWidth();" >
</s:Label>
```

2. Next, replace the contents of the `<s:states>` tag as follows:

```
<s:states>
  <s:State name="up" stateGroups="nonSelected" />
  <s:State name="over" stateGroups="nonSelected" />
  <s:State name="down" stateGroups="nonSelected" />
  <s:State name="disabled" stateGroups="nonSelected, disabledStates" />
  <s:State name="upAndSelected" stateGroups="selectedStates" />
  <s:State name="overAndSelected" stateGroups="selectedStates" />
  <s:State name="downAndSelected" stateGroups="selectedStates" />
  <s:State name="disabledAndSelected"
    stateGroups="selectedStates, disabledStates" />
</s:states>
```

These changes specify a different position for the tab label when the tab is selected.

Launch the application and compare the tab text position. It is now exactly as it is for the Flex 3 Dashboard application (see figure 21).



Figure 21. *The tab text is now positioned exactly as it is for the Flex 3 Dashboard.*

Finally you are done skinning the TabBar and its sub-components for the tabs. The tabs should display as they do in the Flex 3 Dashboard, even when you hover over the tabs. Next you will make changes so the ViewStack views use the new Spark NavigatorContent container.

Use NavigatorContent for ViewStack Views

Now that the application background and TabBar have been modified to make use of the new Flex 4 Spark components and skinning architecture, you will make Dashboard application code changes so the application ViewStack views are represented using the new Flex 4 Spark NavigatorContent container. Flex 4 does not introduce Spark navigator containers, so you still need to use the MX ViewStack, but you cannot use Spark containers directly as views of a ViewStack. You need to use the Spark NavigatorContent container, or wrap Spark containers in an MX container before using them for the views of a ViewStack.

There is one Viewstack view per tab in the application, and the Flex 3 Dashboard uses the MX Canvas container for the views, which are initialized in the `onResultHttpService()` function in `main.mxml`.

Initial Changes in the `onResultHttpService()` Function

1. Replace this line in the `onResultHttpService()` function:

```
var canvas:Canvas = new Canvas();
```

with this line:

```
var canvas:NavigatorContent = new NavigatorContent();
```

You will keep the instance name "canvas" even though you are now using a NavigatorContent, to avoid trivial changes related to using instance names.

2. Next, replace this line at the top of `main.mxml`:

```
import mx.containers.Canvas;
```

with this line, as your code no longer references the Canvas class:

```
import spark.components.NavigatorContent;
```

3. Then remove these two lines, because Flex 4 Spark containers do not have scrollbars unless you explicitly add them, so the lines are unnecessary:

```
canvas.horizontalScrollPolicy = "off";  
canvas.verticalScrollPolicy = "off";
```

Changes in the PodLayoutManager Class to Use NavigatorContent

Because you are now using NavigatorContent for the views, you will now get an error in `main.mxml`, because the PodLayoutManager class is still expecting a Canvas. To fix this error open file `src\com.esria.samples.dashboard\managers\PodLayoutManager.as` and make changes to use NavigatorContent instead of Canvas.

1. Change this: `private var _container:Canvas;`
to this: `private var _container:NavigatorContent;`

2. Then replace this line at the top of PodLayoutManager.as:

```
import mx.containers.Canvas;
```

with this line, as your code no longer references the Canvas class:

```
import spark.components.NavigatorContent;
```

3. Next change the container setter and getter functions to use NavigatorContent:

```
public function set container(canvas: NavigatorContent):void
{
    _container = canvas;
}

public function get container():NavigatorContent
{
    return _container;
}
```

Changes Due to NavigatorContent and Canvas API Differences

To avoid run-time errors you need to make changes because of the following differences between the NavigatorContent and Canvas container API methods and variables:

Canvas	NavigatorContent	Description
<code>addChild()</code>	<code>addElement()</code>	add an item to the container
<code>setChildIndex()</code>	<code>setElementIndex()</code>	set the index of an item in the container
<code>numChildren</code>	<code>numElements</code>	the number of items in the container

Make the following changes in PodLayoutManager.as:

change this: `container.addChild(pod);`
to this: `container.addElement(pod);`

change this: `container.addChild(dragHighlight);`
to this: `container.addElement(dragHighlight);`

change this: `container.setChildIndex(pod, container.numChildren - 1);`
to this: `container.setElementIndex(pod, container.numElements - 1);`

change this: `container.setChildIndex(dragHighlight, i);`
to this: `container.setElementIndex(dragHighlight, i);`

Changing the Pod Class to Fix an Issue Related to NavigatorContent

For some reason, after making changes to use NavigatorContent rather than a Canvas for the views of the Dashboard ViewStack, several UI issues occur. For example, after implementing NavigatorContent for the ViewStack views, when you click the ComboBox in the ChartContent and FormContent components, the ComboBox opens and then immediately closes.

Another example occurs in LineCharts in the ChartContent component. Clicking on a data point in a LineChart drills down into the chart data, and clicking again should drill back up, but after implementing NavigatorContent for views, clicks on LineCharts to drill up often do not work.

The root cause of the issue seems related to differences in the container hierarchy after making changes for Flex 4. In any case, the following workaround eliminates the problem.

1. Delete this line from the `addEventListener()` function:

```
addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
```

2. In `src\com.esria.samples.dashboard\view\Pod.as`, delete the entire `onMouseDown()` function.

3. Add this line to the `onClickMaximizeRestoreButton()` function, directly before the line that dispatches the `PodStateChangeEvent.MAXIMIZE` event:

```
Group(parent).setElementIndex(this, Group(parent).numElements - 1);
```

4. Add this line to the imports at the top of `Pod.as`:

```
import spark.components.Group;
```

Note that you cast the parent to `Group` and not `NavigatorContent`, because the actual pod parent is the `NavigatorContent contentGroup` skin part, which is of type `Group`.

Once again, you should have no errors, and you should be able to launch the Dashboard application with no compile time or run-time errors, though your latest changes will produce no noticeable change in the user interface display.

Change Pod Class to Extend Spark Panel

The ViewStack views, now represented using the new Spark NavigatorContent class, contain the actual pods of the Dashboard application. This section modifies the Pod class to extend the Spark Panel, rather than extend the MX Panel container as is the case in the Flex 3 Dashboard application.

1. First open file Pod.as and import the Spark Panel rather than the MX Panel:

replace this line: `import mx.containers.Panel;`
with this line: `import spark.components.Panel;`

This results in a number of errors (see figure 22) due to differences in the implementation of the MX and Spark Panel, which will be eliminated as you proceed.

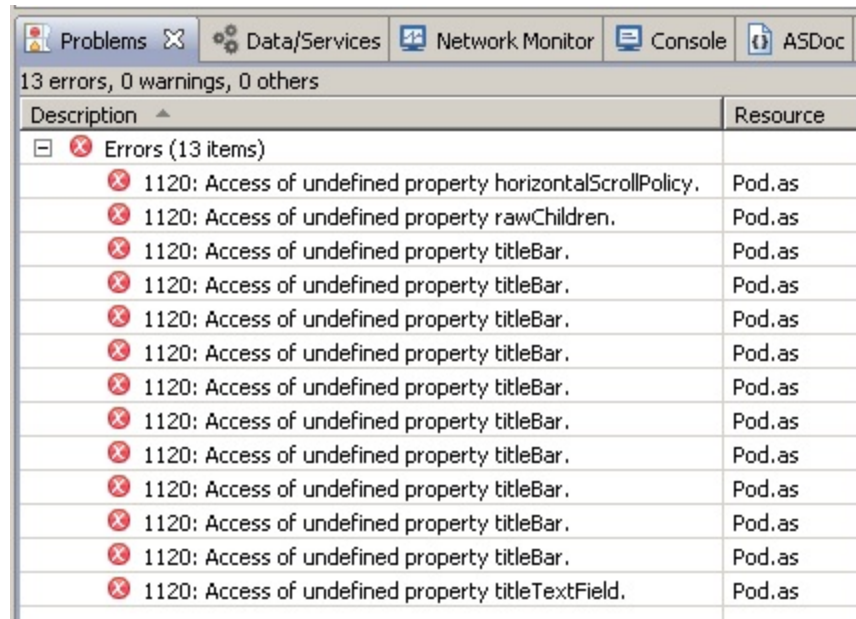


Figure 22. Changing the Pod class to extend the Spark Panel results in errors.

Most of these errors occur because the Spark Panel class does not implement the `titleBar` property. You use a custom skin class to implement the title bar area. The Spark panel also does not implement the `titleTextField` property, and you will also customized this through the skin class.

The error related to the `rawChildren` property occurs due to changes in the Spark component architecture. The MX container class, from which the MX Panel inherits, provides the `rawChildren` property as a way to access the container chrome, such as the container border, etc. You will accomplished this in the Pod custom skin class.

2. Remove this line from the Pod constructor function, as Flex 4 Spark containers do not have scrollbars unless you explicitly add them, so this line is unnecessary:

```
horizontalScrollPolicy = "off";
```

Change Pod Class to Support a Custom Skin

Before creating a custom skin class for the Pod class, you need to add support for the custom skin in the Pod class. You start by declaring "skin parts", visual components declared in the Pod class and implemented in its skin. Notice in the following steps that the skin parts are not declared to be required. If the skin parts were required, and at times this is preferable, then the skin would need to implement all required skin parts, otherwise the compiler will raise errors, as violations of the "contract" between the host component and its skin.

Note however, that even though your Pod skin parts will not be required, they are referenced in functions of the Pod class, so the skin class still needs to implement them. In fact, when using skin states, as with the Flex 4 Pod class to display only the title bar area and hide the control buttons when pods are minimized, changes to some skin states can make it appear that skin parts do not exist, which can cause run-time errors.

Declare Skin Parts in the Pod Class

The first skin part you declare is a container for the minimize and maximize buttons. The Flex 3 Dashboard uses an HBox container, but you will use the new Spark HGroup container. Notice the use of the public access modifier. All skin parts must be public so they will be accessible from the skin class.

1. replace this line: `private var controlsHolder:HBox;`
with these lines: `[SkinPart(required="false")]`
`public var controlsHolder:HGroup;`
2. replace this line: `import mx.containers.HBox;`
with this line: `import spark.components.HGroup;`

The next two skin parts are for the minimize and maximize buttons.

3. replace these lines: `private var minimizeButton:Button;`
`private var maximizeRestoreButton:Button;`
with these lines: `[SkinPart(required="false")]`
`public var minimizeButton:Button;`
`[SkinPart(required="false")]`
`public var maximizeRestoreButton:ToggleButton;`
4. replace this line: `import mx.controls.Button;`
with this line: `import spark.components.Button;`

5. Add this line: `import spark.components.ToggleButton;`

You use the `ToggleButton` control because the `Spark Button` control does not support the "selected" property as the `MX Button` does, and the "selected" property is necessary to switch between two graphical images representing the maximized and restore states for the button display.

The next skin part is for the line dividing the title bar area and the main content area.

6. replace this line: `private var headerDivider:Sprite;`
with these lines: `[SkinPart(required="false")]`
`public var headerDivider:Rect;`

7. replace this line: `import flash.display.Sprite;`

8. with this line: `import spark.primitives.Rect;`

Notice that you have changed the type for the `headerDivider` skin part from `Sprite` to `Rect`.

The next skin part is for an empty rectangle that responds to mouse clicks and to the user dragging the Pod around the Dashboard. Naming it `titleBar` eliminates most of the current errors.

9. Add these lines: `[SkinPart(required="false")]`
`public var titleBar:Group;`

Changes in Functions in the Pod Class

Now that the skin parts have been declared, changes are necessary to several functions in the Pod class.

1. Delete the following text from the `createChildren()` function. Some of it will be replaced with code in the custom skin class, and some of it is unnecessary in your Flex 4 Pod class implementation.

```
if (!headerDivider)
{
    headerDivider = new Sprite();
    titleBar.addChild(headerDivider);
}

if (!controlsHolder)
{
    controlsHolder = new HBox();
    controlsHolder.setStyle("paddingRight", getStyle("paddingRight"));
    controlsHolder.setStyle("horizontalAlign", "right");
    controlsHolder.setStyle("verticalAlign", "middle");
    controlsHolder.setStyle("horizontalGap", 3);
    rawChildren.addChild(controlsHolder);
}
```

```

if(!minimizeButton)
{
    minimizeButton = new Button();
    minimizeButton.width = 14;
    minimizeButton.height = 14;
    minimizeButton.styleName = "minimizeButton";
    controlsHolder.addChild(minimizeButton);
}

if (!maximizeRestoreButton)
{
    maximizeRestoreButton = new Button();
    maximizeRestoreButton.width = 14;
    maximizeRestoreButton.height = 14;
    maximizeRestoreButton.styleName = "maximizeRestoreButton";
    controlsHolder.addChild(maximizeRestoreButton);
}

```

The `createChildren()` function should now be similar to the following:

```

override protected function createChildren():void
{
    super.createChildren();
    addEventListeners();
}

```

2. Next, delete the entire `updateDisplayList()` function, as its code is unnecessary after porting the Dashboard application to Flex 4.

Delete Unnecessary Styles

1. Delete this line from the `minimize()` function, as it will be unnecessary:

```
setStyle("borderSides", "left top right");
```
2. Delete this line from the `onClickTitleBar()` function, as it will be unnecessary:

```
setStyle("borderSides", "left top right bottom");
```
3. Delete this line from the Pod constructor function, as it will be unnecessary:

```
setStyle("titleStyleName", "podTitle");
```

Changes to Display Pods in their Minimized State

The next few changes are necessary to support displaying pods in their minimized state, where only the title bar area is visible, and the pod content and the minimize and maximize buttons are hidden. First you declare a new skin state "minimized".

1. Add the following immediately before the Pod class declaration:

```
[SkinState("minimized)];
```

The Pod class declaration should now be as follows:

```
[SkinState("minimized)];  
public class Pod extends Panel
```

2. Then add the following function to Pod.as:

```
override protected function getCurrentSkinState():String {  
    var returnState:String = "normal";  
    if (windowState == WINDOW_STATE_MINIMIZED) {  
        returnState = "minimized";  
    }  
    return returnState;  
}
```

This function allows you to defines the application logic that determines when the Pod class should enter the minimized state.

3. Next add a call to `invalidateSkinState()` in the `minimize()` function, so the function is similar to the following:

```
public function minimize():void  
{  
    windowState = WINDOW_STATE_MINIMIZED;  
    invalidateSkinState();  
    height = MINIMIZED_HEIGHT;  
    showControls = false;  
}
```

4. Also add a call to `invalidateSkinState()` at the end of the `onClickMaximizeRestoreButton()` function.

These two calls of the `invalidateSkinState()` function cause pod class instances to enter the minimized skin state.

One Change Due to Difference in MX and Spark Panel API

Before implementing the custom pod skin class, you make one additional change to avoid one run-time error due to differences between the MX and Spark Panel API methods.

In main.mxml in the `addPods()` method:

change this: `pod.addChild(podContent);`
to this: `pod.addElement(podContent);`

You should now have no compile time errors, but you still cannot launch the application because the Pod class references some skin parts that have not been created yet. Launch or refresh the application and you should get this run-time error:

```
TypeError: Error #1009: Cannot access a property or method of a null object reference.  
at com.esria.samples.dashboard.view:Pod/addEventListeners()  
[C:\Flex4Dashboard\Dashboard\src\com\esria\samples\dashboard\view\Pod.as:81]
```

The run-time error is occurring at this line of code in the `addEventListeners()` function:

```
titleBar.addEventListener(MouseEvent.CLICK, onMouseDownTitleBar);
```

The run-time occurs because the `titleBar` skin part has been declared but has not yet been instantiated. It will be implemented in the Pod custom skin class in the next section. Anytime you get run-time errors in the Dashboard application, it's a good idea to delete the SharedObject .sol file to reset the application. The location of the Dashboard SharedObject .sol file is operating system specific, and is described in the section entitled "**Location of the Dashboard SharedObject File**".

Now that the Pod class has been modified to support the custom skin, the skin class will be implemented.

Pod Class Custom Skin

Now you implement the custom skin to port the Pod class user interface to use the Flex 4 Spark components and skinning architecture.

Initially Create the Custom Skin Class

To begin, you first create the custom skin class file.

Ensure the skins folder is highlighted, select **File – New – MXML Skin**, and then enter the following values in the New MXML Skin window:

Name: CustomPanelSkin
Host component: com.esria.samples.dashboard.view.Pod
Create as copy of: spark.skins.spark.PanelSkin

Ensure the **"Remove ActionScript styling code"** checkbox is **unchecked**.

Apply the Custom Skin File to the Pod Class

1. To use this custom skin class, add the following line to the top of file Pod.as:

```
import skins.CustomPanelSkin;
```

2. Then add this line to the end of the Pod class constructor:

```
setStyle("skinClass", Class(CustomPanelSkin));
```

Implement Skin Parts in the Custom Skin

Next you implement skin parts in the custom skin, then implement the Pod styles defined in the Flex 3 Dashboard. Each skin part must have the same id in the custom skin that was defined in the Pod class.

Add Skin Parts for the Title Bar and the Minimize/Maximize Buttons Container

First, in file CustomPanelSkin.mxml paste the following directly after the titleDisplay `<s:Label>` control, but inside the same group containing the titleDisplay Label:

```
<s:Group id="titleBar" left="0" right="0" top="0" bottom="0"/>  
  
<s:HGroup id="controlsHolder" right="9" top="4"  
    horizontalAlign="right" verticalAlign="middle" gap="3">  
    <s:Button id="minimizeButton"/>  
    <s:ToggleButton id="maximizeRestoreButton"/>  
</s:HGroup>
```

Change the Name and Color of the Divider Skin Part

1. Then, change the id of the Rect that currently has an id of "tbDiv" to have an id of "headerDivider".
2. Set the color of the divider separating the Pod title bar area from the content area by setting the color of the SolidColor fill for the headerDivider Rect to 0x999999.

Now you implement styles in the skin class from styles podTitle and Pod in file styles.css.

Implement titleDisplay Styles

1. Find the titleDisplay Label control in the skin class, and add `fontSize="11"` `fontFamily="arial"`.
2. Change the `left` property to have a value of 12.
3. Add the style `Kerning="off"` to achieve the same Panel title string position and rendering as the Flex 3 Dashboard.
4. Implement the Pod `header-height` style, by setting the `titleDisplay minHeight` property to 22.

When you are done, the Label tag should be similar to the following:

```
<s:Label id="titleDisplay" maxDisplayedLines="1"
  left="12" right="3" top="1" bottom="0" minHeight="22"
  verticalAlign="middle" textAlign="start" fontWeight="bold"
  fontSize="11" fontFamily="arial" kerning="off">
```

Implement Padding Styles in the contentGroup

Implement the padding style properties in `contentGroup` Group container by adding the following layout code directly after the opening Group tag. The padding properties have been adjusted for a layout similar to that of the Flex 3 Dashboard.

```
<s:layout>
  <s:VerticalLayout paddingTop="32" paddingLeft="20"
    paddingRight="20" paddingBottom="20"/>
</s:layout>
```

Some Styles do not Need to be Set

The Spark Panel has square bottom corners by default, so you do not need to implement the `rounded-bottom-corners` style. The Spark Panel also has a default border that is a 1px solid line, and a default background color of white, so you also do not need to implement the `border-thickness`, `border-style`,

and `background-color` Pod styles. The Spark Panel skin does not define a title bar highlight fill, so the Pod `highlight-alphas` style property does not need to be set.

Set the Border Styles

For a border color like the Flex 3 Dashboard, add two lines inside the `initializationComplete()` method, after the line setting the `useChromeColor` property:

```
setStyle("borderColor", 0x999999);  
setStyle("borderAlpha", 1);
```

You cannot simply set the color and alpha of the `borderStroke SolidColorStroke` object, because to support run-time border styling the `updateDisplayList()` method includes these lines:

```
borderStroke.color = getStyle("borderColor");  
borderStroke.alpha = getStyle("borderAlpha");
```

If you were to set the color and alpha directly in the `borderStroke SolidColorStroke` object, the above two lines in `updateDisplayList()` would wipe out your values.

Set the Corner Radius Style

This is also the case for the `corner-radius` style, so add the following line in the `initializationComplete()` method for the Pod top corner radius:

```
setStyle("cornerRadius", 6);
```

Adjust the Pod Drop Shadow

Replace the `RectangularDropShadow` in the file with the following:

```
<s:RectangularDropShadow id="dropShadow" blurX="10" blurY="10" alpha="1" distance="5"  
    angle="0" color="#999999" left="0" top="5" right="5" bottom="0"/>
```

Adjust the Pod Title Bar Colors

Implement the `Pod header-colors` style controlling the gradient fill of the title bar area by replacing the two `GradientEntry` lines in the `tbFill Rect` with the following:

```
<s:GradientEntry color="0xFFFFFFFF" ratio=".1"/>  
<s:GradientEntry color="0xCCCCCC" ratio=".8"/>
```

Except for the minimize and maximize buttons, which you implement in the next section, the Pod title bar area is now looking very similar to the Flex 3 Dashboard (see figure 23).



Flex 4 Dashboard



Flex 3 Dashboard

Figure 23. *Except for the buttons, the title bar looks very similar to the Flex 3 Dashboard.*

Pod Minimize and Maximize Custom Skins

Now you will create two custom button skin classes for the maximize and minimize buttons located in the Pod title bar area. These skins will implement the `maximizeRestoreButton` and `minimizeButton` styles from `file styles.css`.

Initially Create the Custom Skin Class

Ensure the skins folder is highlighted, select **File – New – MXML Skin**, and enter the following values.

For these two skins, the "Remove ActionScript styling code" checkbox should be checked.

Maximize Button

Name: CustomMaximizeButtonSkin
Host component: spark.components.ToggleButton
Create as copy of: spark.skins.spark.ToggleButtonSkin

Minimize Button

Name: CustomMinimizeButtonSkin
Host component: spark.components.Button
Create as copy of: spark.skins.spark.ButtonSkin

Apply the Custom Skin Files to the Pod Custom Skin

To use these custom skin classes, in `CustomPanelSkin.mxml` set the `skinClass` properties for the buttons in the `CustomPanelSkin` `controlsHolder` `HGroup`:

```
<s:Button id="minimizeButton" skinClass="skins.CustomMinimizeButtonSkin"/>
<s:ToggleButton id="maximizeRestoreButton" skinClass="skins.CustomMaximizeButtonSkin"/>
```

Implement the Minimize and Maximize Buttons in the Custom Skin

Open files `CustomMinimizeButtonSkin.mxml` and `CustomMaximizeButtonSkin.mxml` and do the following:

1. In both files, remove `minWidth="21" minHeight="21"` from the opening `<s:SparkSkin>` tag.
2. In both files, remove everything in layers 1 – 8. Basically, remove everything from just after the ending `</s:states>` tag until just before the ending `</s:SparkSkin>`.
3. In file `CustomMinimizeButtonSkin.mxml` only, add the following, directly after the ending

`</s:states>` tag:

```
<s:BitmapImage includeIn="up, down" width="14" height="14"
    source="@Embed(source='/assets/minimize_up.png')"/>
```

```
<s:BitmapImage includeIn="over" width="14" height="14"
```

```
source="@Embed(source='/assets/minimize_over.png')"/>
```

4. In file CustomMaximizeButtonSkin.mxml only, add the following, directly after the ending `</s:states>` tag:

```
<s:BitmapImage includeIn="up, down" width="14" height="14"  
    source="@Embed(source='/assets/maximize_up.png')"/>
```

```
<s:BitmapImage includeIn="over" width="14" height="14"  
    source="@Embed(source='/assets/maximize_over.png')"/>
```

```
<s:BitmapImage includeIn="upAndSelected, downAndSelected" width="14"  
    height="14" source="@Embed(source='/assets/restore_up.png')"/>
```

```
<s:BitmapImage includeIn="overAndSelected" width="14" height="14"  
    source="@Embed(source='/assets/restore_over.png')"/>
```

Launch the Dashboard and notice that the Pod title bar area is nearly identical to the Flex 3 Dashboard application (see figure 24).



Figure 24. The pod title bar area now looks nearly identical to the Flex 3 Dashboard.

New Run-time Errors

The application launches, and clicking the pod maximize button works as expected, but if you click the pod minimize button or attempt to drag pods in the Dashboard you see run-time errors. The run-time errors resulting from clicking the minimize button occur because the custom skin has not been modified to support minimizing pods. You will do this in the next section. The run-time errors resulting from dragging the pods will be eliminated when you convert the DragHighlight component to Flex 4 in a later section.

Support Minimizing Pods in Custom Skin

Changes are necessary in CustomPanelSkin.mxml to support displaying pods in a minimized state. Some of these steps are potentially confusing, so follow along carefully.

Add the minimized State

Add support for the new "minimized" state by adding this line to the `<s:states>` tag:

```
<s:State name="minimized"/>
```

Exclude Visual Elements Hidden in minimized State

Use the `excludeFrom` property to exclude visual elements that should not display in the minimized state:

```
<s:RectangularDropShadow id="dropShadow" blurX="10" blurY="10"
    alpha="1" distance="5" angle="0" color="#999999" left="0"
    top="5" right="5" bottom="0" excludeFrom="minimized"/>
```

```
<s:Rect id="background" left="1" top="1" right="1" bottom="1"
    excludeFrom="minimized">
```

```
<s:Group id="contentGroup" width="100%" height="100%"
    minWidth="0" minHeight="0" excludeFrom="minimized">
```

Add Checks for Null Objects in updateDisplayList() Function

It is necessary to modify the `updateDisplayList()` function to check for null objects, because in the minimized state, some visual elements are being excluded and will not exist, which will cause null object run-time errors. You can just replace the entire function definition in your CustomPanelSkin.mxml file:

```
override protected function updateDisplayList(unscaledWidth:Number,
    unscaledHeight:Number):void
{
    if (getStyle("borderVisible") == true)
    {
        border.visible = true;
        contents.left = contents.top = contents.right = contents.bottom = 1;
        if(background != null){
            background.left = background.top = background.right =
                background.bottom = 1;
        }
    }
    else
    {
        border.visible = false;
```

```

        contents.left = contents.top = contents.right = contents.bottom = 0;
        if(background != null){
            background.left = background.top = background.right =
            background.bottom = 0;
        }
    }

    if(dropShadow != null){
        dropShadow.visible = getStyle("dropShadowVisible");
    }

    var cr:Number = getStyle("cornerRadius");
    var withControls:Boolean = (currentState == "disabledWithControlBar"
        || currentState == "normalWithControlBar");

    if (cornerRadius != cr)
    {
        cornerRadius = cr;
        if(dropShadow != null){
            dropShadow.tlRadius = cornerRadius;
            dropShadow.trRadius = cornerRadius;
            dropShadow.blRadius = withControls ? cornerRadius : 0;
            dropShadow.brRadius = withControls ? cornerRadius : 0;
        }

        setPartCornerRadii(topMaskRect, withControls);
        setPartCornerRadii(border, withControls);
        if(background != null){
            setPartCornerRadii(background, withControls);
        }
    }

    if (bottomMaskRect) setPartCornerRadii(bottomMaskRect, withControls);
    borderStroke.color = getStyle("borderColor");
    borderStroke.alpha = getStyle("borderAlpha");
    if(backgroundFill != null){
        backgroundFill.color = getStyle("backgroundColor");
        backgroundFill.alpha = getStyle("backgroundAlpha");
    }

    super.updateDisplayList(unscaledWidth, unscaledHeight);
}

```

Now you should be able to minimize, maximize, and restore the Dashboard pods, in the same manner as the Flex 3 application. However, dragging the pods results in a run-time error, which will be eliminated when you convert the DragHighlight component to Flex 4 next.

Modifying the DragHighlight Component

Now you convert the DragHighlight component, which displays a highlight box when pods are dragged, to Flex 4. You will use a Spark BorderContainer to define the visual aspects of the drag highlight defined in the DragHighlight style in styles.css.

Open DragHighlight.mxml and replace the `<mx:Box>` tag with this:

```
<s:BorderContainer xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/mx"
  borderColor="#CCCCCC" borderWidth="3"
  borderStyle="solid" backgroundColor="#F3F3F3"/>
```

This is the only change necessary for the DragHighlight component.

Now you should be able to drag and drop pods in different positions in the dashboard with no errors.

Next, you begin the first of a series of sections that will complete porting the Flex Dashboard application to Flex 4. These sections convert the components containing the actual pod content.

Modify PodContentBase to Extend VGroup

Now you make code changes so the PodContentBase class extends the Spark VGroup. In the Flex 3 Dashboard application, PodContentBase extends the Flex 3 MX VBox container. The PodContentBase class is the base class of the ChartContent, FormContent, ListContent, and PieChartContent components, which represent the containers for the actual data for the pods. Open the PodContentBase.as file and make the following changes.

1. change this line:
to this:

```
public class PodContentBase extends VBox
public class PodContentBase extends VGroup
```
2. change this line:
to this:

```
import mx.containers.VBox;
import spark.components.VGroup;
```

These are the only two changes necessary to the PodContentBase class. Note, launching and manipulating the application now might result in run-time errors, because although you have modified the PodContentBase class, you have not yet modified its sub-classes.

ChartContent Component Changes

This is the first of several sections in which you make changes to the ChartContent, PieChartContent, FormContent, and ListContent components, which extend the PodContentBase class and represent the containers for the actual data for the pods. Some of these steps are potentially confusing, so follow along carefully.

Some steps in these sections are the same, but you will make changes to each component separately, because making changes to all four files at the same time would make it impossible to launch the application due to multiple errors for an extended period of time.

Change Root Tag Namespaces

To begin, in ChartContent.mxml replace the namespace definition in the root tag.

replace this line:

```
xmlns:mx="http://www.adobe.com/2006/mxml"
```

with these lines:

```
xmlns:fx="http://ns.adobe.com/mxml/2009"  
xmlns:s="library://ns.adobe.com/flex/spark"  
xmlns:mx="library://ns.adobe.com/flex/mx"
```

Change Script and Component Tag Namespaces

1. Change the namespace prefix from mx to fx for the `<mx:Script>` tag.
2. Change the namespace prefix from mx to fx for all `<mx:Component>` tags.

Place Effects in a Declarations Tag

Next, enclose the two effects defined in the file within an `<fx:Declarations>` tag:

```
<fx:Declarations>  
  <effects:DrillDownEffect id="drillDownEffect" duration="1500"  
    effectEnd="chart.showDataTips=true" />  
  <effects:DrillUpEffect id="drillUpEffect" duration="1500"  
    effectEnd="chart.showDataTips=true"/>  
</fx:Declarations>
```

In Flex 4, non-visual components such as effects, formatters, validators, etc. must be placed within a `<fx:Declarations>` tag.

Changes to Use Spark DropDownList Control

Next make changes to use the Spark DropDownList instead of the MX ComboBox:

1. delete this line: `import mx.controls.ComboBox;`

2. replace the entire mx ComboBox declaration with this:

```
<s:DropDownList selectedIndex="{_selectedViewIndex}"
  change="onChangeComboBox(event)" width="100"
  fontSize="10" fontWeight="bold" fontFamily="verdana">
  <s:ArrayList>
    <fx:String>Bar Chart</fx:String>
    <fx:String>Line Chart</fx:String>
  </s:ArrayList>
</s:DropDownList>
```

Note, the Spark DropDownList is used because the Spark ComboBox items are editable by default, and the Dashboard items should not be editable. Also, notice that the DropDownList width is specified because the default behavior of the Spark DropDownList in the Dashboard does not respect the size of the largest item.

3. Next, locate the `onChangeComboBox()` function:

change this line: `var index:Number = ComboBox(e.target).selectedIndex;`
to this: `var index:Number = DropDownList(e.target).selectedIndex;`

4. Inside the `onChangeComboBox()` function and change the event type:

change this: `ListEvent`
to this: `IndexChangedEvent`

5. At the top of the file:

change this line: `import mx.events.ListEvent;`
to this line: `import spark.events.IndexChangeEvent;`

Changes to the Chart Font-Family

Add `fontFamily="verdana"` to the `<mx:ColumnChart>` and `<mx:LineChart>` tags.

Change Component and Container Tags

1. Replace all `<mx:HBox>` tags with `<s:HGroup>`.
2. Replace all `<mx:Canvas>` tags with `<s:NavigatorContent>`.

If you launch the application you will get a run-time error because other components have not yet been converted to Flex 4, but if you click the **Dismiss All** or **Continue** buttons in the error window, you will see that the pods displaying line and bar charts are nearly identical to the line and bar chart pods in the Flex 3 Dashboard (see figures 25 and 26).



Flex 4 Dashboard Chart Pods

Figure 25. Flex 4 Dashboard Line and Bar Chart Pods.



Flex 3 Dashboard Chart Pods

Figure 26. Flex 3 Dashboard Line and Bar Chart Pods.

Next you convert the PieChartContent component to Flex 4.

PieChartContent Component Changes

In this section you make changes to the PieChartContent component.

1. To begin, in PieChartContent.mxml replace the namespace definitions.

```
xmlns:fx="http://ns.adobe.com/mxml/2009"  
xmlns:s="library://ns.adobe.com/flex/spark"  
xmlns:mx="library://ns.adobe.com/flex/mx"
```

2. Then change the namespace prefix from mx to fx for the `<mx:Script>` tag and for the `<mx:Component>` tag.
3. Then enclose the two effects defined in the file within an `<fx:Declarations>` tag.

Changes to the Chart Font-Family

Add `fontFamily="verdana"` to the `<mx:PieChart>` tag.

If you launch the application you still get a run-time error because other components have not yet been converted to Flex 4, but if you click the **Dismiss All** or **Continue** buttons in the error window, you will see that the pods displaying the pie charts are identical to the pie chart pods in the Flex 3 Dashboard (see figure 27).

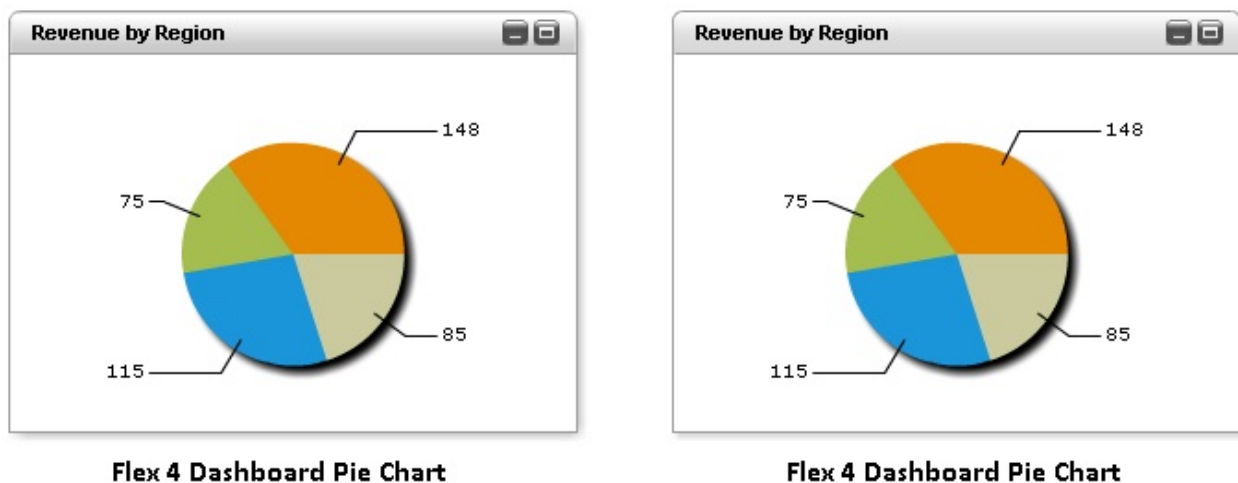


Figure 27. Pie Chart Pod in the Flex 4 and Flex 3 Dashboards.

Next you convert the FormContent component to Flex 4.

FormContent Component Changes

In this section you make changes to the FormContent component. Several steps in this section are identical to steps in the section on modifying the ChartContent component, therefore the steps in this section may be abbreviated. This section contains many steps, and some steps are potentially confusing, so follow along carefully.

Change Root Tag Namespaces

In FormContent.mxml replace the namespace definitions.

```
xmlns:fx="http://ns.adobe.com/mxml/2009"  
xmlns:s="library://ns.adobe.com/flex/spark"  
xmlns:mx="library://ns.adobe.com/flex/mx"
```

Change Script Tag Namespace Prefix

Then change the namespace prefix for the `<mx:Script>` tags to `fx`.

Set Gap in the Root Tag

Add `gap="7"` to the `<PodContentBase>` root tag near the top of the file.

Changes to Use Spark DropDownList Control

1. Replace the entire `mx ComboBox` declaration with this:

```
<s:DropDownList selectedIndex="{selectedViewIndex}"  
  change="onChangeComboBox(event)" width="74"  
  fontSize="10" fontWeight="bold" fontFamily="verdana">  
  <s:ArrayList>  
    <fx:String>Form</fx:String>  
    <fx:String>Grid</fx:String>  
  </s:ArrayList>  
</s:DropDownList>
```

2. In the `onChangeComboBox()` method:

replace this line: `var index:Number = ComboBox(e.target).selectedIndex`

with this line: `var index:Number = DropDownList(e.target).selectedIndex;`

3. Change the event type in the `onChangeComboBox()` function
from `ListEvent` to `IndexChangedEvent`

4. Change this line: `import mx.events.ListEvent;`
to this line: `import spark.events.IndexChangeEvent;`

Changes from HBox to HGroup

1. Replace all `<mx:HBox>` tags with `<s:HGroup>`.
2. Remove the `horizontalGap="0"` property from the `<s:HGroup>` tag.
3. Add the `gap="10"` property to all the `<s:HGroup>` tags.
4. In the `<s:HGroup>` containing the `newButton` and `deleteButton` buttons, set `gap="2"`.

Changes from VBox and Canvas to NavigatorContent

1. Replace all `<mx:VBox>` tags with `<s:NavigatorContent>`.
2. Add the following immediately after the opening `<s:NavigatorContent>` tag:

```
<s:layout>
  <s:VerticalLayout horizontalAlign="center" gap="7"/>
</s:layout>
```

3. Replace all `<mx:Canvas>` tags with `<s:NavigatorContent>`.

Note, do not add a layout tag inside this instance of `<s:NavigatorContent>`.

Changes to Text, Date, and DataGrid Controls

1. Replace all `<mx:Label>` tags with `<s:RichEditableText>`
2. Add the following to all `<s:RichEditableText>`:
`fontSize="10" paddingTop="4" fontFamily="verdana"`
3. Replace all `<mx:TextInput>` tags with `<s:TextInput>`.
4. Add the following to all `<s:TextInput>`:
`fontSize="10" paddingTop="4" fontFamily="verdana"`
5. Replace the `<mx:TextArea>` tag with `<s:TextArea>`.
6. Add the following to the `<s:TextArea>`:
`fontSize="10" paddingTop="4" fontFamily="verdana"`
7. Replace all `<mx:Button>` tags with `<s:Button>`.
8. Add `fontSize="10" fontWeight="bold" fontFamily="verdana"` only to `<s:Button>` tags that have the `label` property defined (the `newButton` and `deleteButton` buttons).

9. Add `fontSize="10" fontFamily="verdana"` to the `<mx:DateField>` tag.
10. Add `fontSize="10" fontFamily="verdana"` to the `<mx:DataGrid>` tag.

The Flex 4 Form pods now look nearly identical to those in the Flex 3 Dashboard application. Next you create custom skins for the left and right buttons.

Create Custom Skins for Left and Right Arrow Buttons

Next you create two custom button skin classes for the left and right buttons allowing the user to move through the data items. These skins will implement the `leftArrowButton` and `rightArrowButton` styles in file `styles.css`.

Copy Existing Skin as the Base for the Two Button Skins

Rather than create brand new skins, you can create two copies of `CustomMinimizeButtonSkin.mxml` and rename them as follows:

```
CustomLeftButtonSkin.mxml
CustomRightButtonSkin.mxml
```

Apply the Custom Skins in the FormContent Component

To use these custom skin classes remove the `width` and `height` properties from the two `Button` tags, and set the `skinClass` properties for the buttons in `FormContent.mxml`:

```
<s:Button
    enabled="{dataProvider.length > 0}"
    skinClass="skins.CustomLeftButtonSkin"
    click="saveData();selectedIndex-=1;"/>

<s:Button
    enabled="{dataProvider.length > 0}"
    skinClass="skins.CustomRightButtonSkin"
    click="saveData();selectedIndex+=1;"/>
```

Edit the Custom Skins

Then edit skin class files to replace the `BitmapImage` components:

1. In file `CustomLeftButtonSkin.mxml` only, replace the two `BitmapImage` tags with the following three tags:

```
<s:BitmapImage includeIn="up, down" width="9" height="12"
    source="@Embed(source='/assets/left_arrow_up.gif')"/>

<s:BitmapImage includeIn="over" width="9" height="12"
    source="@Embed(source='/assets/left_arrow_over.gif')"/>
```

```
<s:BitmapImage includeIn="disabled" width="9" height="12"
  source="@Embed(source='/assets/left_arrow_disabled.gif')"/>
```

2. In file CustomRightButtonSkin.mxml only, replace the two `BitmapImage` tags with the following three tags:

```
<s:BitmapImage includeIn="up, down" width="9" height="12"
  source="@Embed(source='/assets/right_arrow_up.gif')"/>
```

```
<s:BitmapImage includeIn="over" width="9" height="12"
  source="@Embed(source='/assets/right_arrow_over.gif')"/>
```

```
<s:BitmapImage includeIn="disabled" width="9" height="12"
  source="@Embed(source='/assets/right_arrow_disabled.gif')"/>
```

If you launch the application you no longer get run-time errors, and the pods displaying the forms are nearly identical to the form pods in the Flex 3 Dashboard (see figure 28).

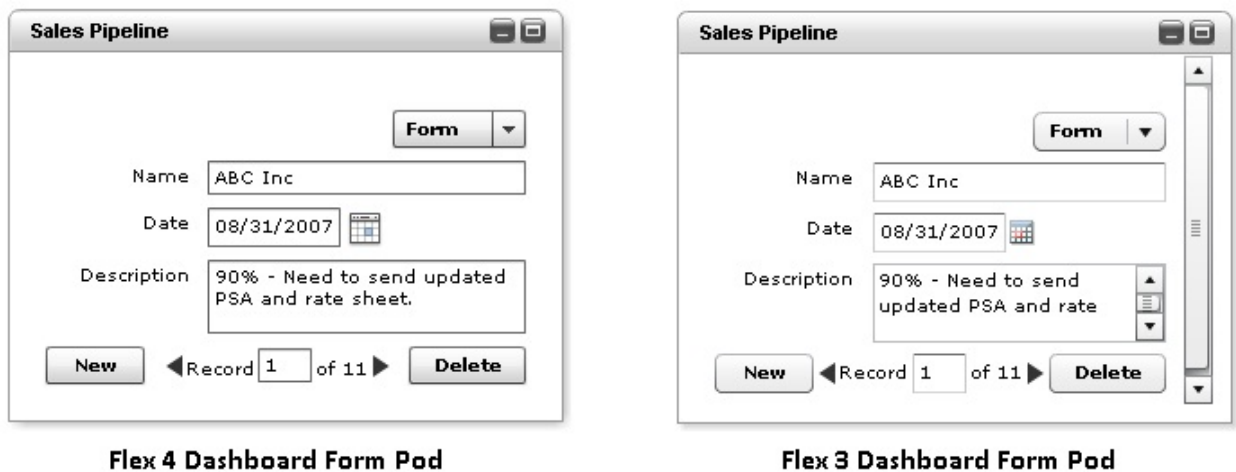


Figure 28. Form Pod in the Flex 4 and Flex 3 Dashboards.

Next you convert the `ListContent` component to Flex 4. This is the last component to convert to Flex 4 in the Dashboard application!

ListContent Component Changes

In this section you make changes to the ListContent component.

Change Root Tag Namespaces

To begin, in ListContent.mxml replace the namespace definitions.

```
xmlns:fx="http://ns.adobe.com/mxml/2009"  
xmlns:s="library://ns.adobe.com/flex/spark"  
xmlns:mx="library://ns.adobe.com/flex/mx"
```

Change Script Tag Namespace Prefix

Then change the namespace prefix for the `<mx:Script>` tag to `fx`.

Use Spark Label Control

```
change this:  import mx.controls.Label;  
to this:      import spark.components.Label;
```

Use Spark List Control

Replace `<mx:List>` with `<s:List>`

Set Border Style

Replace `borderStyle="none"` with `borderVisible="false"` in the `<s:List>` tag.

Set Font Size

Add `fontSize="11"` in the `<s:List>` tag.

Add a Layout Class

Add a closing `</s:List>` tag, and add the following immediately after the opening `<s:List>` tag:

```
<s:layout>  
    <s:VerticalLayout gap="16" paddingTop="6" paddingLeft="5"/>  
</s:layout>
```

Your `<s:List>` tag should now be similar to the following:

```
<s:List  
    id="list"
```

```
width="100%" height="100%"
dataProvider="{dataProvider}"
click="onClickList(event) "
borderVisible="false"
itemRenderer="com.esria.samples.dashboard.renderers.ListPodRenderer"
fontSize="11">
<s:layout>
    <s:VerticalLayout gap="16" paddingTop="6" paddingLeft="5"/>
</s:layout>
</s:List>
```

Custom Skin for ListContent List

The Flex 3 Dashboard ListContent component displays only a vertical scrollbar, and does not display a horizontal scrollbar. It achieves this by setting the `horizontalScrollPolicy` property in the custom item renderer for the ListContent List sub-component to "off". To remove the horizontal scrollbar in the Flex 4 it is necessary to create a custom skin class for the ListContent List sub-component.

The custom skin will also be used to remove the border from the List control.

Initially Create the Custom Skin Class

Ensure the skins folder is highlighted, select **File – New – MXML Skin**. The "Remove ActionScript styling code" checkbox should be **checked**.

Name: CustomListSkin
Host component: spark.components.List
Create as copy of: spark.skins.spark.ListSkin

Apply the Custom Skin File to the ListContent List Component

To use this skin class, set the `skinClass` property for the ListContent `<s:List>` tag:
`skinClass="skins.CustomListSkin"`

Implement a Style and Remove Horizontal Scrollbar in the Custom Skin

1. In the custom skin class file, add `alpha="0"` in the `borderStroke SolidColorStroke` to remove the List border.
2. Add `horizontalScrollPolicy="off"` to the Scroller sub-component opening tag to remove the `horizontalScrollBar`.

ListPodRenderer Component Changes

In this section, you make changes to the ListPodRenderer, used to render list items in the List.

Change Root Tag Namespaces

To begin, in ListPodRenderer.mxml replace the namespace definitions.

```
xmlns:fx="http://ns.adobe.com/mxml/2009"  
xmlns:s="library://ns.adobe.com/flex/spark"  
xmlns:mx="library://ns.adobe.com/flex/mx"
```

Change Script Tag Namespace Prefix

Then change the namespace prefix for the `<mx:Script>` tag to `fx`.

Use ItemRenderer Class

Replace `<mx:HBox>` with `<s:ItemRenderer>`

Remove Properties

Remove the `backgroundColor` and `horizontalScrollPolicy` properties from the `ItemRenderer` tag, as they are not supported by the `ItemRenderer` class.

Add a Layout Class

Add the following immediately following the opening `<s:ItemRenderer>` tag:

```
<s:layout>  
    <s:HorizontalLayout gap="12"/>  
</s:layout>
```

Add Normal and Over States

Add the following immediately following the closing `</s:layout>` tag:

```
<s:states>  
    <s:State name="normal"/>  
    <s:State name="over"/>  
</s:states>
```

Set Styles

Add the following lines to the `onInitialize()` function:

```
setStyle("rollOverColor", "#FFFFFF");  
setStyle("selectionColor", "#FFFFFF");
```

Use the Spark Label and Make Adjustments

1. Replace `<mx:Label>` with `<s:Label>`
2. Remove the `truncateToFit="true"` property from the label, as it is unnecessary.
3. Add the following in the `<s:Label>` tags:

```
fontSize="10" fontFamily="verdana" kerning="off"
```

4. In the `nameLabel <s:Label>`, replace this:

```
rollOver="Label(event.currentTarget).setStyle('textDecoration', 'none');" 
```

with this: `textDecoration.over="none"`

5. In the `nameLabel <s:Label>`, remove this:

```
rollOut="Label(event.currentTarget).setStyle('textDecoration', 'underline');" 
```

If you launch the application you see the pods displaying the lists are nearly identical to the list pods in the Flex 3 Dashboard (see figure 29).



Figure 29. List Pod in the Flex 4 and Flex 3 Dashboards.

Change the URL in the news.xml Data File

If you launch the Dashboard and click the links in the Company News pods, a new browser window opens but the page fails to load, because `www.esria.com` no longer exists. Open file `data/news.xml` and replace all occurrences of `www.esria.com` with `www.adobe.com`.

Wrap-up

That's it, you're finally finished porting the Adobe Flex 3 Example Dashboard application to use the Flex 4 Spark controls and architecture! If you have ideas or suggestions, please do not hesitate to email us at customerservice@stardustsystems.com. As you might imagine, we cannot respond to questions about the tutorial, but you can always post questions on one of the many Flex forums.

The best way to learn even more about the Dashboard application is to think of things it does not do, or limitations or problems you see in the Dashboard, and then try to understand and modify the code. Just remember to always export your project to an archive folder just in case you want to step back in time.

Resources

This article illustrated how to convert the Flex Developer Center demo Dashboard application, written using the Flex 3 SDK, to use the Flex 4 Spark components and architecture. More information and learning resources relevant to the Dashboard demo application:

[Flex 4 Help - Introduction to Charts](#)

[Flex 4 Help - Drilling Down into Data](#)

[Flex 4 Help - Drag and Drop](#)

[Flex Developer Center - What's new in Flex 4](#)

[Flex in a Week video training \(free\)](#)