ActionScript Developer's Guide to Building Mobile Applications

Developing BlackBerry Tablet Applications with Flex 4.5

O'REILLY®

Rich Tretola

Developing BlackBerry Tablet Applications with Flex 4.5

Ready to put your ActionScript 3 skills to work on mobile apps? This hands-on book walks you through the process of creating an Adobe AIR application for BlackBerry Tablets from start to finish, using the Flex 4.5 framework. Move quickly from a basic Hello World application to complex interactions with BlackBerry APIs, and get complete code examples for working with tablet components—including the accelerometer, GPS unit, camera, file system, and multi-touch screen. This is an ideal resource no matter how much Flex experience you have.

- Use Flash Builder 4.5 to create and debug a Flex Mobile project
- Choose a layout option to determine which files Flash Builder autogenerates
- Obtain permissions you need to install your app on a BlackBerry Tablet
- Read and write text files, browse the file system for media files, and create and write to an SQLite database
- Learn how to use native qnx components within your application
- Publish your app to a BlackBerry installer file with Flash Builder

Twitter: @oreillymedia
facebook.com/oreilly

US $29.99 **CAN $34.99**
ISBN: 978-1-449-30556-7
52999
9 781449 305567

O'REILLY®
oreilly.com

Developing BlackBerry Tablet Applications with Flex 4.5

Developing BlackBerry Tablet Applications with Flex 4.5

Rich Tretola

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Developing BlackBerry Tablet Applications with Flex 4.5
by Rich Tretola

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Editor: Mary Treseler
Production Editor: Holly Bauer
Proofreader: Holly Bauer
Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Robert Romano

Printing History:
July 2011: First Edition.

ISBN: 978-1-449-30556-7

[LSI]

1308602843

Table of Contents

Preface

Introduction to BlackBerry Tablet OS

In 2011, Research in Motion (RIM) introduced an entirely new operating system known as the BlackBerry Tablet OS. The first device using this new operating system was released in May of 2011 and was known as the BlackBerry PlayBook. Through Adobe's partnership with RIM, this new operating system was built fully integrated with the Adobe AIR runtime; as a result, the performance of Adobe AIR applications running on BlackBerry tablet devices is outstanding, and RIM has built specific libraries accessible to ActionScript for deep integration within the operating system.

This book walks you through the creation of your first Adobe AIR application using the Flex 4.5 framework and provides examples of how to interact with the device's components. These include the GPS unit, camera, gallery, accelerometer, multi-touch display, and the `StageWebView`, Operating System interactions, native components, and more.

Who This Book Is For

Developing BlackBerry Tablet OS Applications with Adobe Flex 4.5 targets developers of every skill level. It starts with a basic Hello World application and then quickly moves to more complicated examples where the BlackBerry Tablet OS APIs are explored.

Who This Book Is Not For

This book is not for developers who are interested in developing native BlackBerry applications. This book only provides examples of BlackBerry Tablet OS application development using Adobe Flex 4.5 and ActionScript 3.

Conventions Used in This Book

The following typographical conventions are used in this book:

Menu options
: Menu options are shown using the → character, such as File→Open.

Italic
: Italic indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: This is used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: This indicates commands or other text that should be typed literally by you.

`Constant width italic`
: This indicates text that should be replaced with user-supplied values or values determined by context.

This Book's Example Files

You can download the example files for this book from this location:

http://examples.oreilly.com/9781449305567-files/

Where necessary, multiple code samples are provided for each recipe to correspond with the different development environments. Each sample will be separated into a folder for the specific environment. Each application should include the needed code for your environment as well as an application descriptor file.

Using the Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. In addition, answering a question by citing this book and quoting example code does not require permission. However, selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Developing BlackBerry Tablet Applications with Flex 4.5* by rich Tretola (O'Reilly). Copyright 2011 Rich Tretola, 978-1-449-30556-7."

If you think your use of code examples falls outside fair use or the permission given here, feel free to contact us at *permissions@oreilly.com*.

How to Use This Book

Development rarely happens in a vacuum. In today's world, email, Twitter, blogs, coworkers, friends, and colleagues all play a vital role in helping you solve development problems. Consider this book yet another resource at your disposal to help you solve the development problems you will encounter (however, this book does have a big advantage: it's available anytime of the day or night). The content is arranged in such a way that solutions should be easy to find and understand.

Safari® Books Online

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/9781449305567

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

First and foremost, I would like to thanks my wife and best friend, Kim, and my daughters, Skye, Coral, and Trinity, for their love and support. I love you all!

I would like to say thanks to the Adobe Flex team and the members of the Flex CAB who provided early access and support to the AIR 2.6 and Flash Builder 4.5 tools and documentation.

Thank you as well to Mary Treseler from O'Reilly for providing this opportunity.

CHAPTER 1

Hello World

This section will walk you through building your first BlackBerry PlayBook application using Adobe Flash Builder 4.5. If you don't have Flash Builder 4.5, you can get a trial from Adobe at *http://www.adobe.com/products/flashbuilder/*.

Now that you have Flash Builder 4.5 installed, open it and lets get started.

Create a Flex Mobile Project

Create a new Flex Mobile Project by choosing File→New→Flex Mobile Project as shown in Figure 1-1.

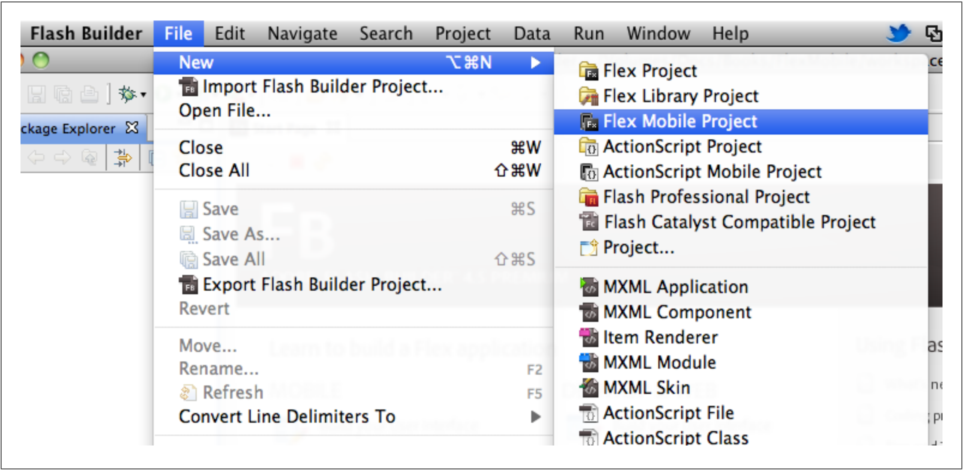
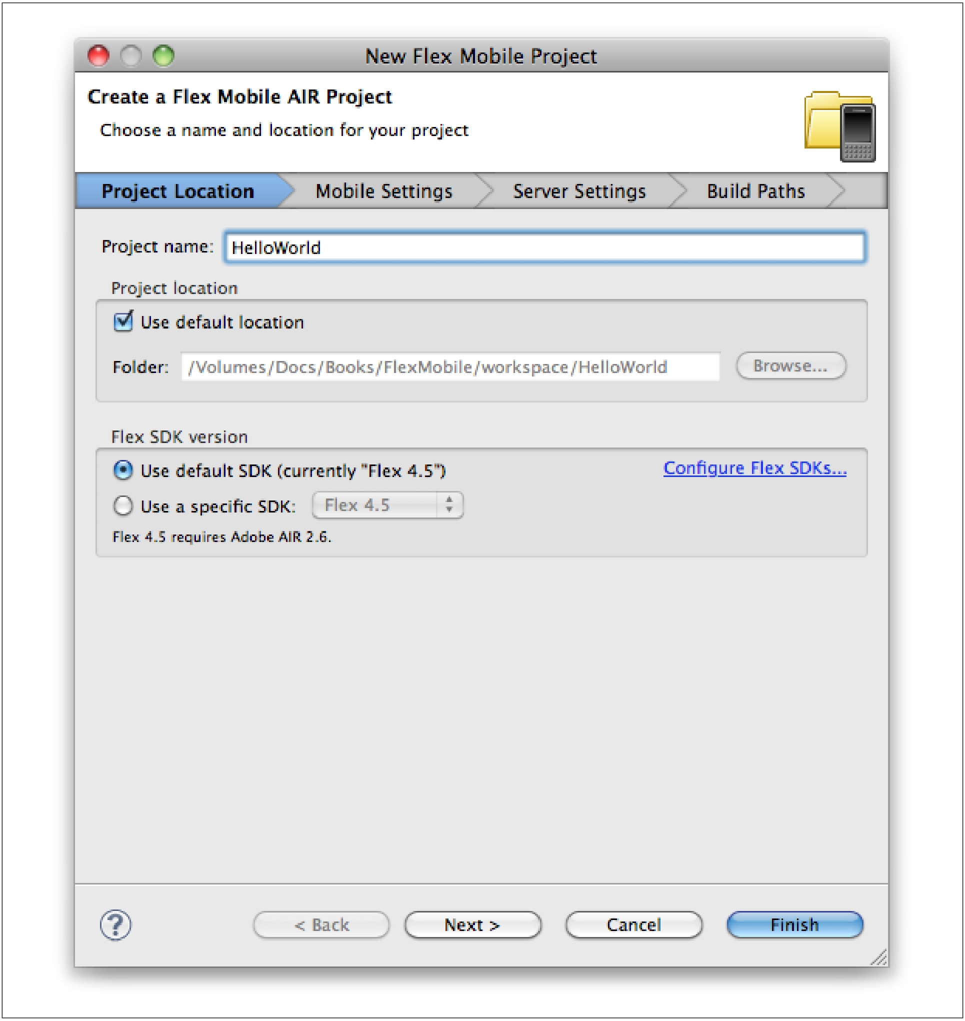


Figure 1-1. Create a Flex Mobile Project

This will open the New Flex Mobile Project wizard, which will walk you through the rest of the project-creation process. The first screen you're presented allows you to set the project name, location, and Flex SDK. Enter the name *HelloWorld* as the Project name and leave the other settings to their defaults. Click Next, as shown in Figure 1-2.


New Flex Mobile Project

Create a Flex Mobile AIR Project

Choose a name and location for your project

Project Location | Mobile Settings | Server Settings | Build Paths

Project name: HelloWorld

Project location

[✓] Use default location

Folder: /Volumes/Docs/Books/FlexMobile/workspace/HelloWorld

Flex SDK version

[●] Use default SDK (currently "Flex 4.5")

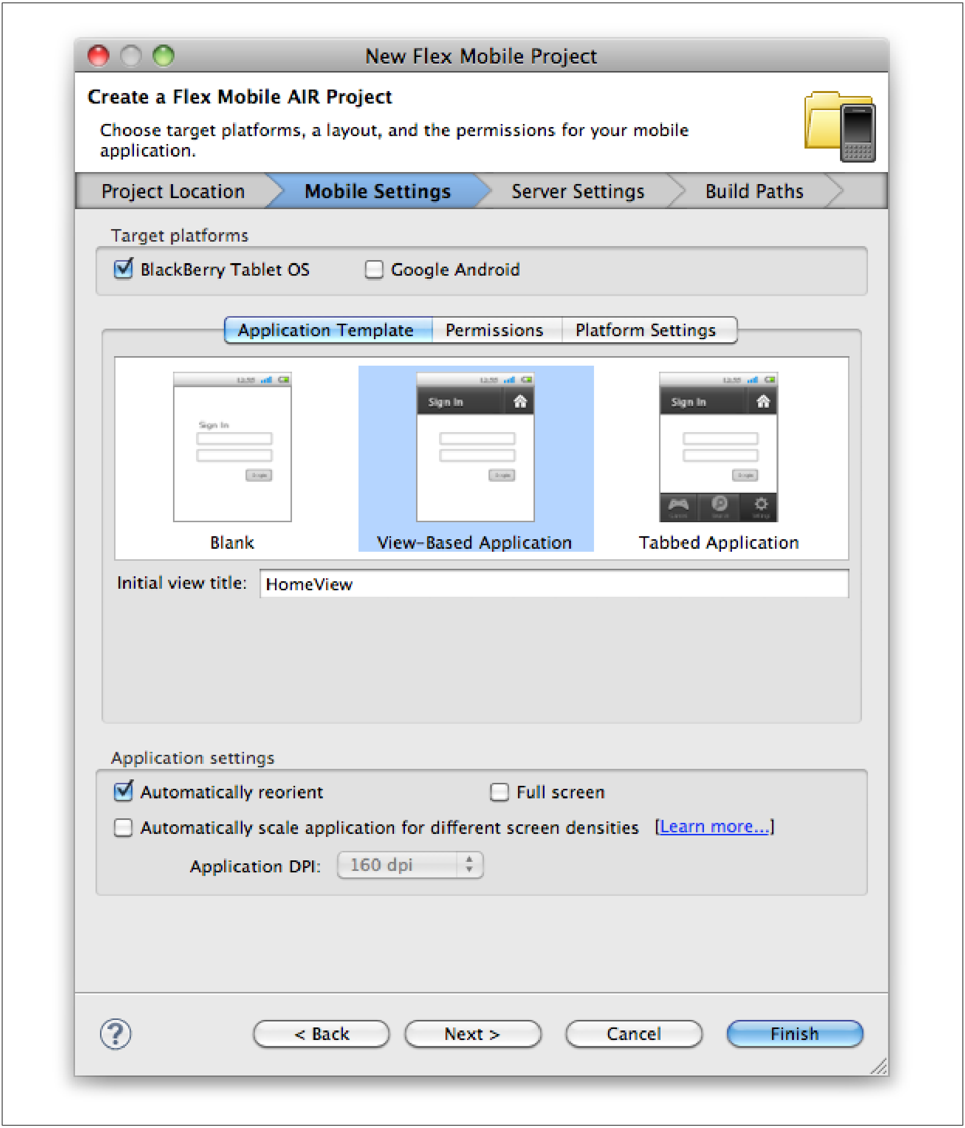
[○] Use a specific SDK: Flex 4.5

Flex 4.5 requires Adobe AIR 2.6.

Figure 1-2. Project Name and Location

The second screen in the new project wizard is where you can select settings specific to the target platform. Since you have installed the Blackberry Tablet OS plug-in, you will see the options for both Google Android and BlackBerry Tablet OS. Select Black-Berry Tablet OS. You also have the option of three different application types: Blank, View-Based Application, and Tabbed Application. For this first project, select View-Based Application as shown in Figure 1-3 and leave the other settings to their defaults.

Next, click on the Permissions tab. Within this tab, you can select the permissions your application will need in order to interact with the BlackBerry Tablet OS–native APIs. For the purposes of this exercise, leave only the access_internet permission selected, as shown in Figure 1-4. Click next.


New Flex Mobile Project

Create a Flex Mobile AIR Project

Choose target platforms, a layout, and the permissions for your mobile application.

Target platforms

[✓] BlackBerry Tablet OS [ ] Google Android

Initial view title: HomeView

Application settings

[✓] Automatically reorient [ ] Full screen

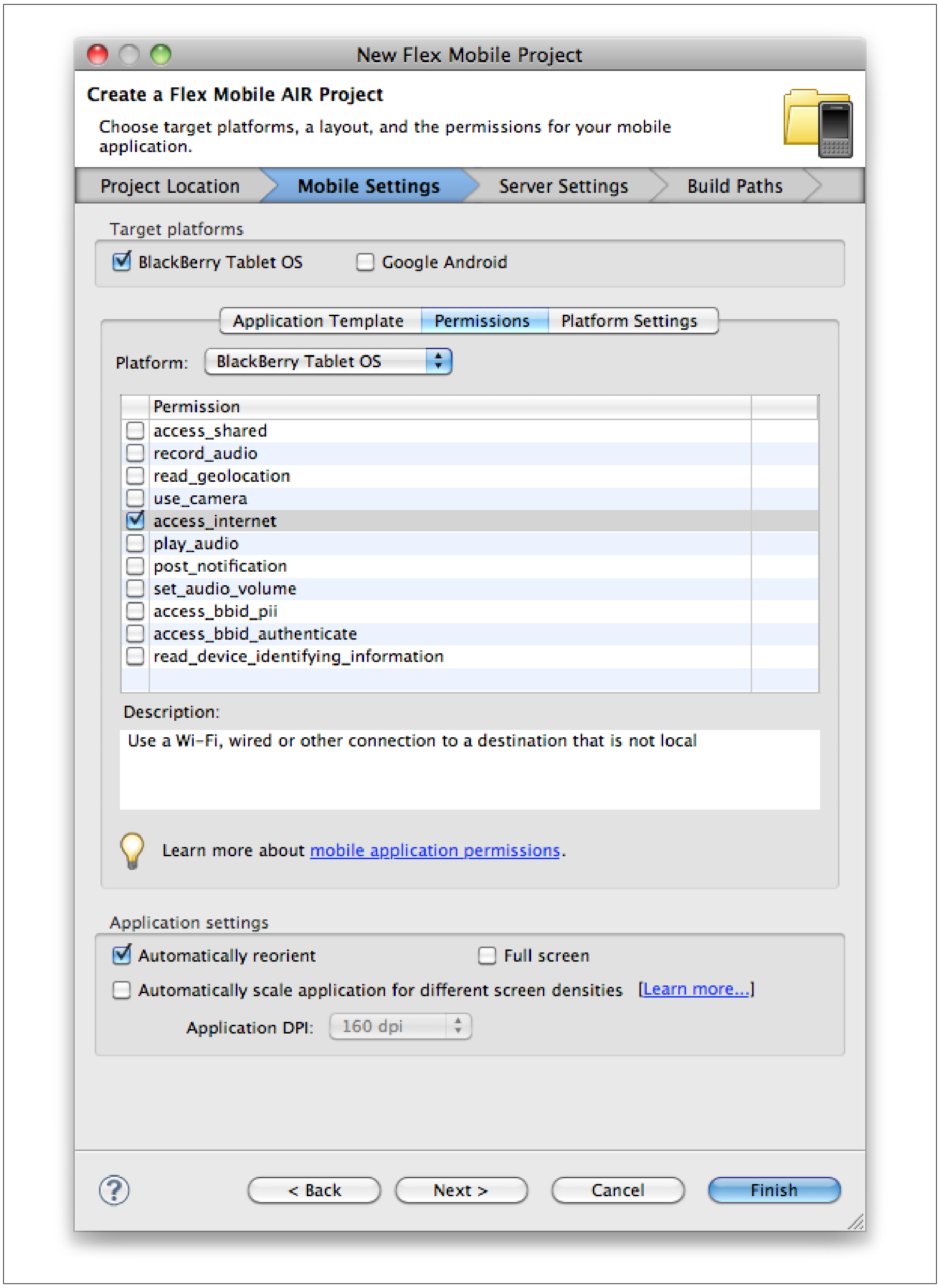
[ ] Automatically scale application for different screen densities [Learn more...]

Application DPI: 160 dpi

Figure 1-3. Select Application Template

The next screen allows for the configuration of an application server and output folder. For this project, we will not be using an application server, so leave it set to None/Other and click next, as shown in Figure 1-5.

The last screen that you will see is the Build Paths screen; this is where you set your Application ID. This setting is very important, as the Application ID will be used to identify your application in BlackBerry App World. To ensure that your application has a unique identifier, the reverse domain naming convention works best.


New Flex Mobile Project

Create a Flex Mobile AIR Project

Choose target platforms, a layout, and the permissions for your mobile application.

Project Location | **Mobile Settings** | Server Settings | Build Paths

Target platforms

[✓] BlackBerry Tablet OS   [ ] Google Android

Application Template | Permissions | Platform Settings

Platform: BlackBerry Tablet OS

	Permission
[ ]	access_shared
[ ]	record_audio
[ ]	read_geolocation
[ ]	use_camera
[✓]	access_internet
[ ]	play_audio
[ ]	post_notification
[ ]	set_audio_volume
[ ]	access_bbid_pii
[ ]	access_bbid_authenticate
[ ]	read_device_identifying_information

Description:

Use a Wi-Fi, wired or other connection to a destination that is not local

Learn more about mobile application permissions.

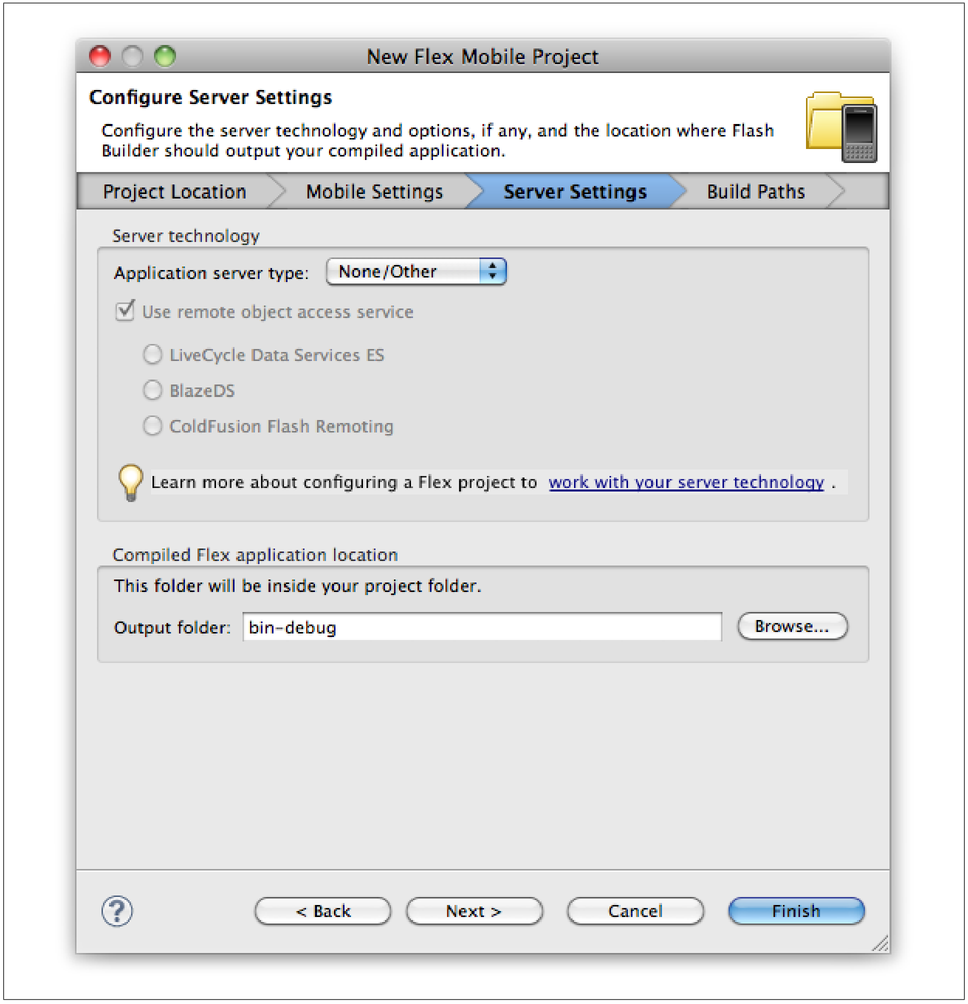
Application settings

[✓] Automatically reorient   [ ] Full screen

[ ] Automatically scale application for different screen densities [Learn more...]

Application DPI: 160 dpi

Figure 1-4. Set BlackBerry Tablet OS Permissions


New Flex Mobile Project

Configure Server Settings

Configure the server technology and options, if any, and the location where Flash Builder should output your compiled application.

Server technology

Application server type: None/Other

[✓] Use remote object access service

( ) LiveCycle Data Services ES

( ) BlazeDS

( ) ColdFusion Flash Remoting

Learn more about configuring a Flex project to work with your server technology .

Compiled Flex application location

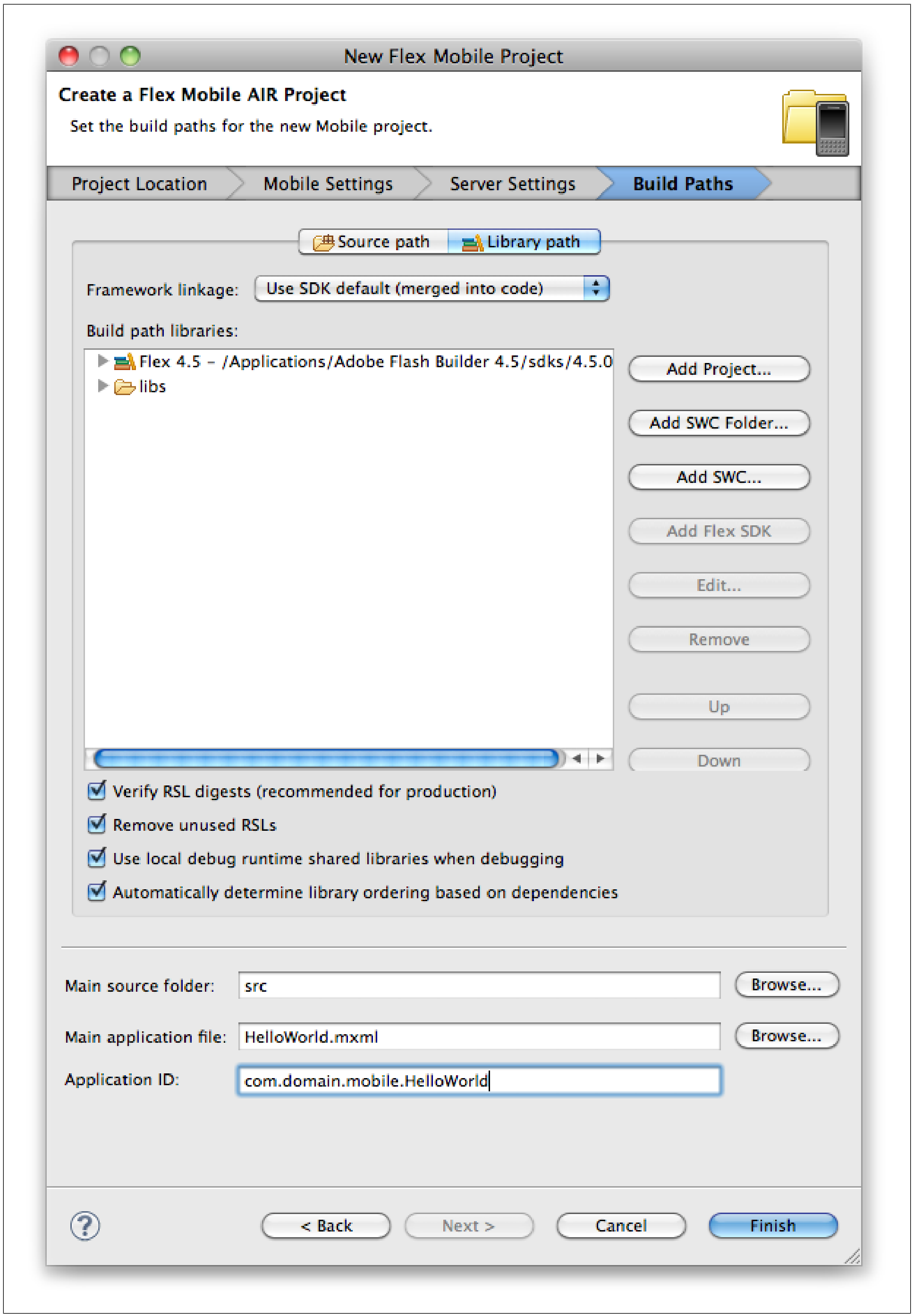
This folder will be inside your project folder.

Output folder: bin-debug

Figure 1-5. Server Settings

Figure 1-6 shows the value of *com.domain.mobile.HelloWorld* as the application ID. By replacing the word domain with a domain that you own, you can ensure that your application ID is unique. Complete this step and click Finish.

Flash Builder will now create your new project and, by default, the *HelloWorldHomeView.mxml* will be created and opened in the workspace, along with the *HelloWorld.mxml* main application file (see Figure 1-7).


New Flex Mobile Project

Create a Flex Mobile AIR Project

Set the build paths for the new Mobile project.

Framework linkage: Use SDK default (merged into code)

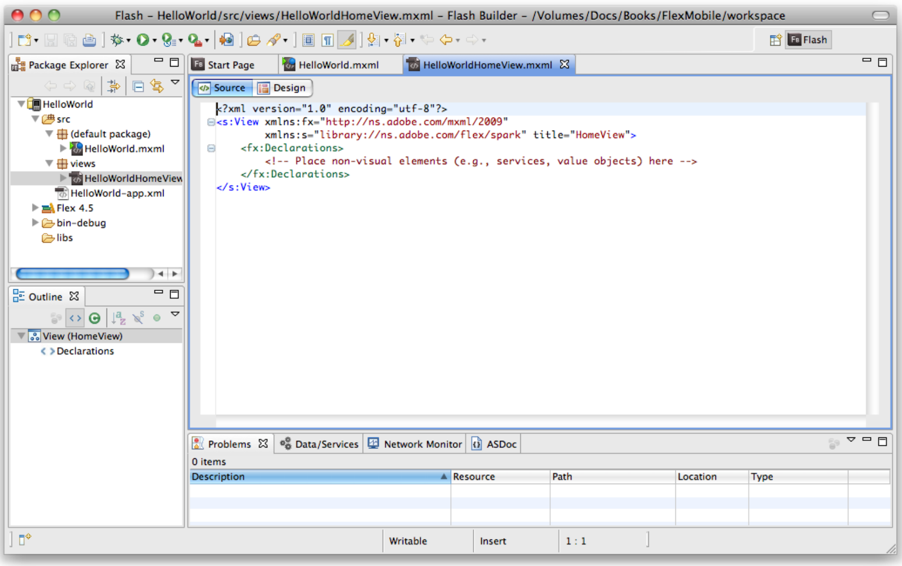
Build path libraries:

Flex 4.5 – /Applications/Adobe Flash Builder 4.5/sdks/4.5.0
libs

- [✓] Verify RSL digests (recommended for production)
- [✓] Remove unused RSLs
- [✓] Use local debug runtime shared libraries when debugging
- [✓] Automatically determine library ordering based on dependencies

Main source folder:	src
Main application file:	HelloWorld.mxml
Application ID:	com.domain.mobile.HelloWorld

Figure 1-6. Application ID



```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="HomeView">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
</s:View>
```

Figure 1-7. New Project has been created

Let's update the contents of the *HelloWorldHomeView.mxml* by adding a Label.

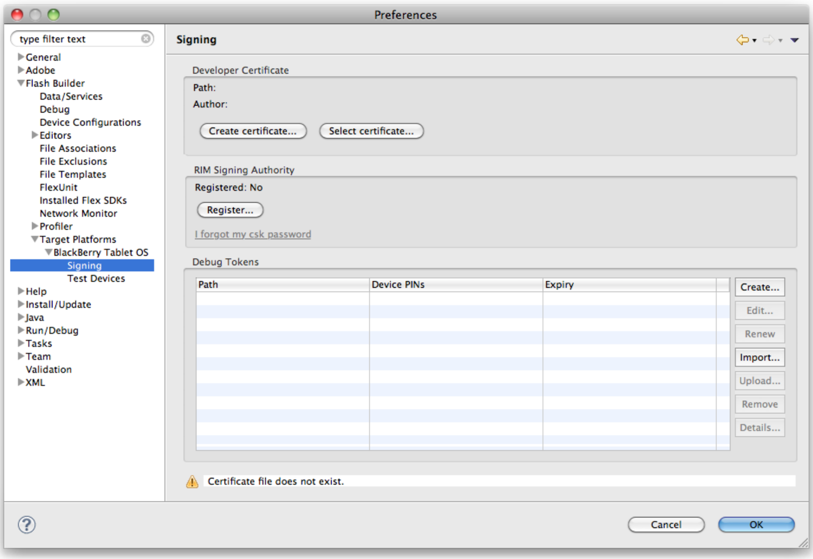
```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="HomeView">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:Label text="Hello World" fontSize="24"
             horizontalCenter="0" verticalCenter="0"/>

</s:View>
```

Before running an application for the first time, you will need to set up either a simulator environment or a device for your testing.

Set Up a Test Environment

If you do not have a device to test with, RIM has made a VMware image available—it's located at *http://us.blackberry.com/developers/tablet/adobe.jsp*. After downloading this disk image, open it within VMware and complete the following steps to test your application.

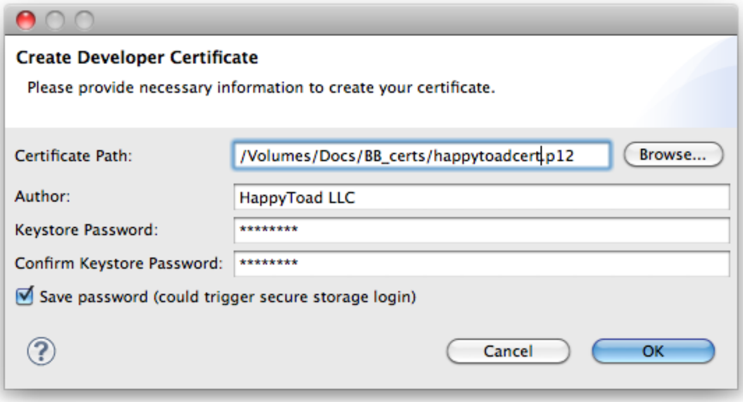
Within Flash Builder, click on the Flash Builder and then on the Preferences menu. Then expand the Flash Builder→Target Platforms→BlackBerry Tablet OS→Signing item within the tree on the left side. See Figure 1-8.



Certificate file does not exist.

Figure 1-8. Signing Screen

Click on the Create certificate to create a new certificate for your workspace. Fill in the form values, then click OK. See Figure 1-9.



Create Developer Certificate

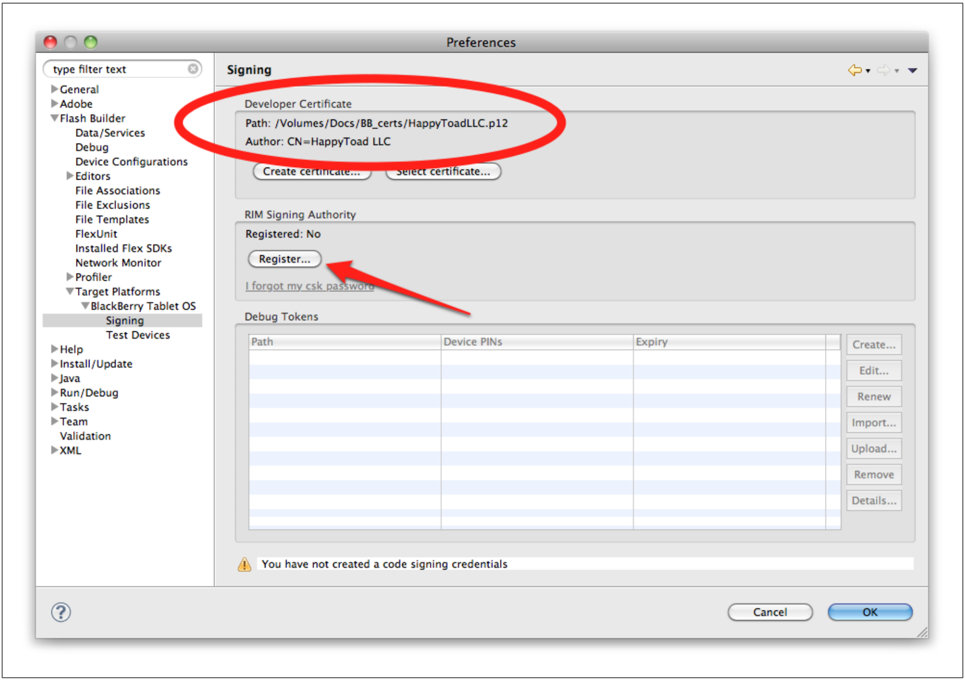
Please provide necessary information to create your certificate.

Certificate Path:	/Volumes/Docs/BB_certs/happytoadcert.p12
Author:	HappyToad LLC
Keystore Password:	********
Confirm Keystore Password:	********

☑ Save password (could trigger secure storage login)

Figure 1-9. Create a Developer Certificate

Next, you will need to register as a developer with RIM by completing the form at *https://www.blackberry.com/SignedKeys/*. Once you have completed this form, you will be sent some code-signing files. To register these with Flash Builder, click on the Register button (see Figure 1-10). Figure 1-11 shows the registration form. Figure 1-12 shows that you have now been registered with the RIM Signing Authority.



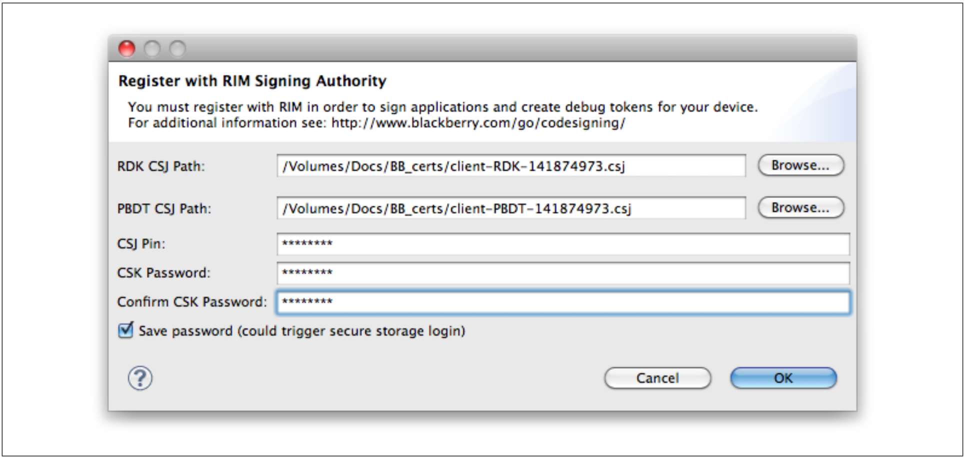
Developer Certificate
Path: /Volumes/Docs/BB_certs/HappyToadLLC.p12
Author: CN=HappyToad LLC

RIM Signing Authority
Registered: No

Debug Tokens

You have not created a code signing credentials

Figure 1-10. Developer Certificate Created

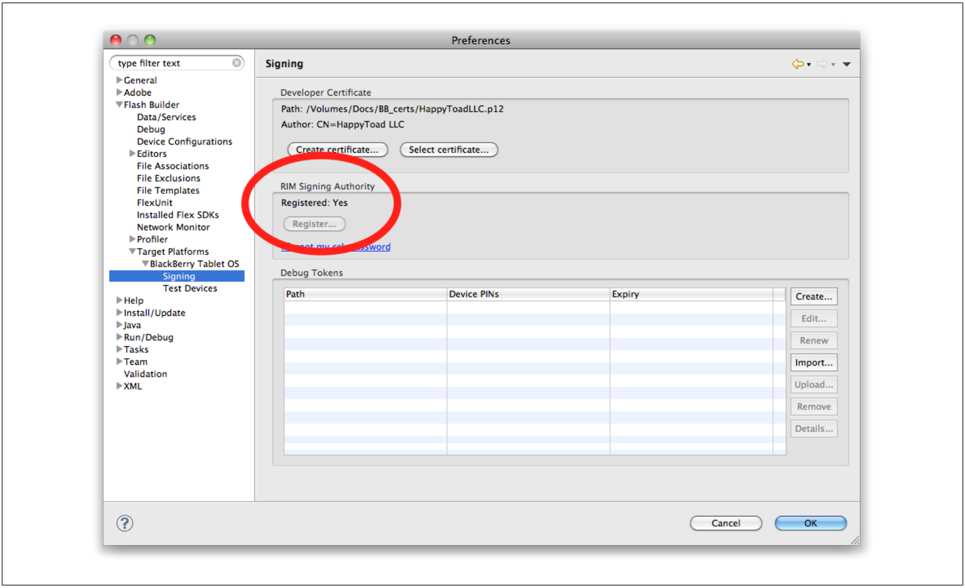


Register with RIM Signing Authority

You must register with RIM in order to sign applications and create debug tokens for your device.
For additional information see: http://www.blackberry.com/go/codesigning/

RDK CSJ Path: /Volumes/Docs/BB_certs/client-RDK-141874973.csj
PBDT CSJ Path: /Volumes/Docs/BB_certs/client-PBDT-141874973.csj
CSJ Pin: ********
CSK Password: ********
Confirm CSK Password: ********
☑ Save password (could trigger secure storage login)

Figure 1-11. Register RIM Signing Authority



Path: /Volumes/Docs/BB_certs/HappyToadLLC.p12

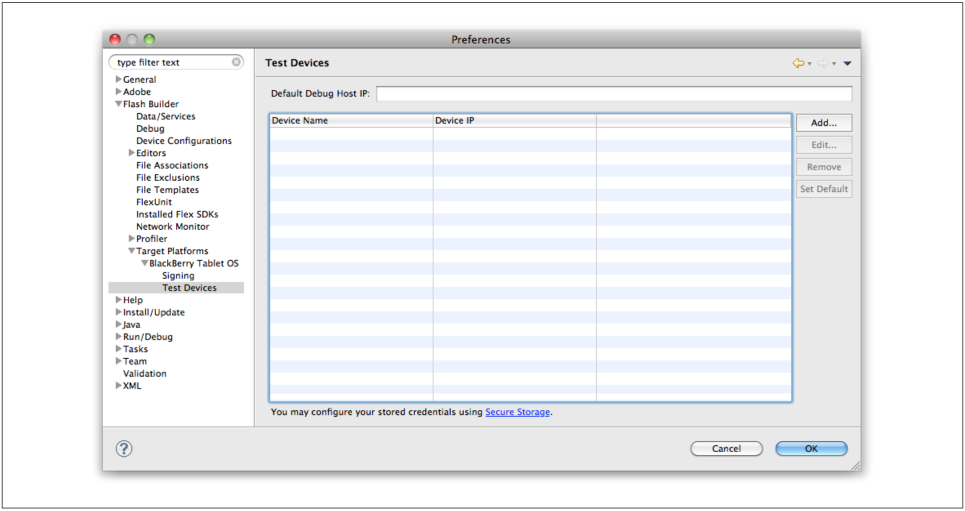
Author: CN=HappyToad LLC

Registered: Yes

Figure 1-12. RIM Signing completed

Now that you have created a certificate and registered yourself with RIM, you need to add a test device. Select the Test Devices item from the left menu.

Click on the Add button. See the setup instructions in "Setup Simulator" on page 11 or "Setup Device" on page 14 to move forward with your testing. See Figure 1-13.



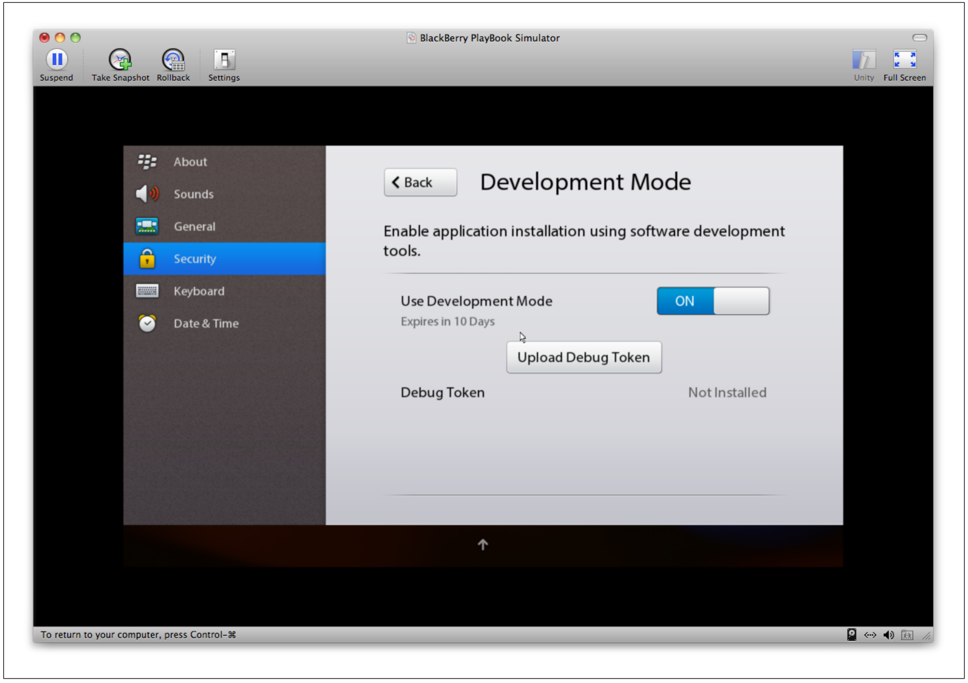
You may configure your stored credentials using Secure Storage.

Figure 1-13. Add a Test Device

Setup Simulator

Open the virtual machine within VMware. Click on the gear in the upper-right corner and then click security. Turn on development mode (see Figure 1-14). Now go back to the home screen and click the little person icon to read your IP address (see Figure 1-15).

Once you have your IP address, complete the Add BlackBerry Tablet OS Test Device setup. Be sure to check the box that says Create a debug token and upload it to this device (see Figure 1-16). Once you say OK, you will be prompted to set the device to accept the debug token. Figure 1-17 shows the warning dialog. Figure 1-18 shows the simulator waiting for the debug token. Figure 1-19 shows that the debug token has been added.



Development Mode

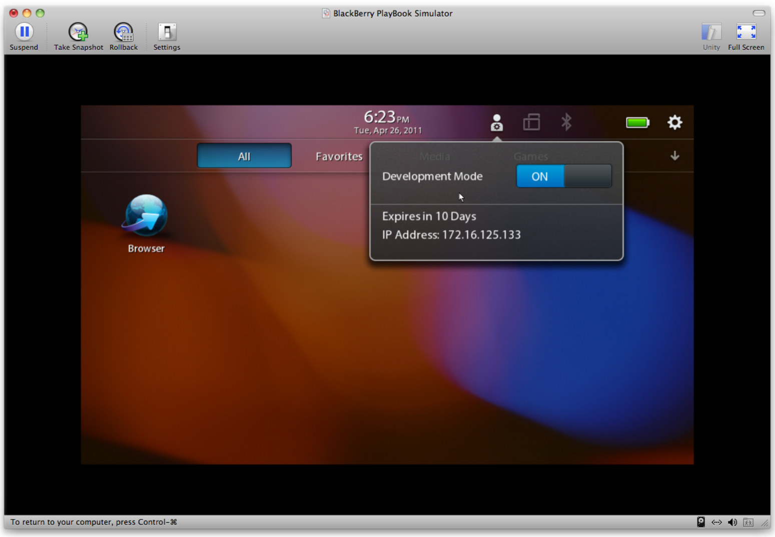
Enable application installation using software development tools.

Use Development Mode — ON
Expires in 10 Days

Debug Token — Not Installed

To return to your computer, press Control-⌘

Figure 1-14. Turn on Development Mode in Simulator

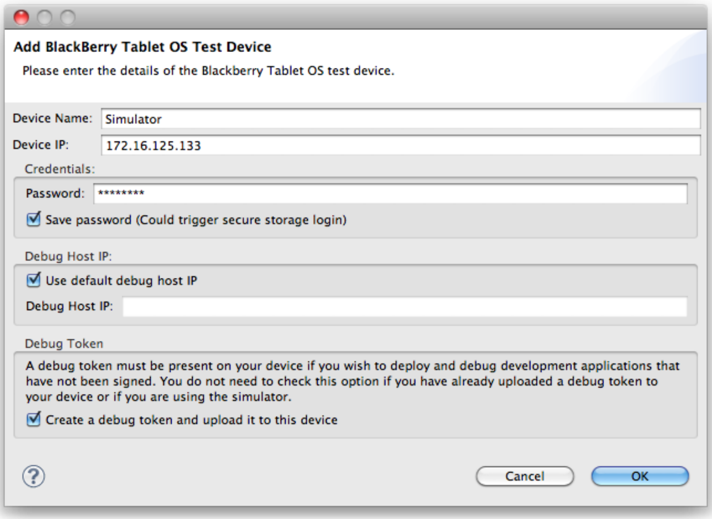


Development Mode ON

Expires in 10 Days

IP Address: 172.16.125.133

Figure 1-15. Read IP Address



Add BlackBerry Tablet OS Test Device

Please enter the details of the Blackberry Tablet OS test device.

Device Name: Simulator

Device IP: 172.16.125.133

Credentials:

Password: ********

☑ Save password (Could trigger secure storage login)

Debug Host IP:

☑ Use default debug host IP

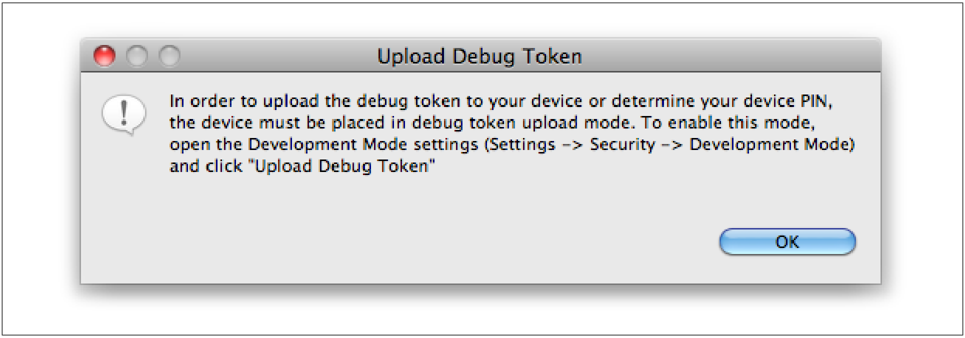
Debug Host IP:

Debug Token

A debug token must be present on your device if you wish to deploy and debug development applications that have not been signed. You do not need to check this option if you have already uploaded a debug token to your device or if you are using the simulator.

☑ Create a debug token and upload it to this device

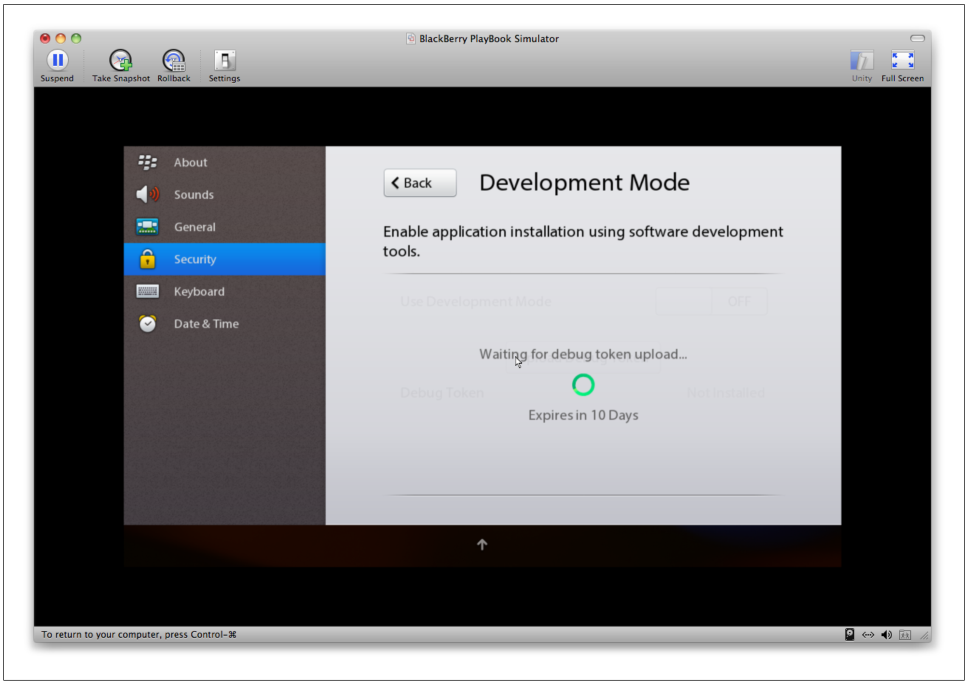
Figure 1-16. Complete Form with IP Address of Simulator



Upload Debug Token

In order to upload the debug token to your device or determine your device PIN, the device must be placed in debug token upload mode. To enable this mode, open the Development Mode settings (Settings -> Security -> Development Mode) and click "Upload Debug Token"

Figure 1-17. Upload Debug Token warning



Development Mode

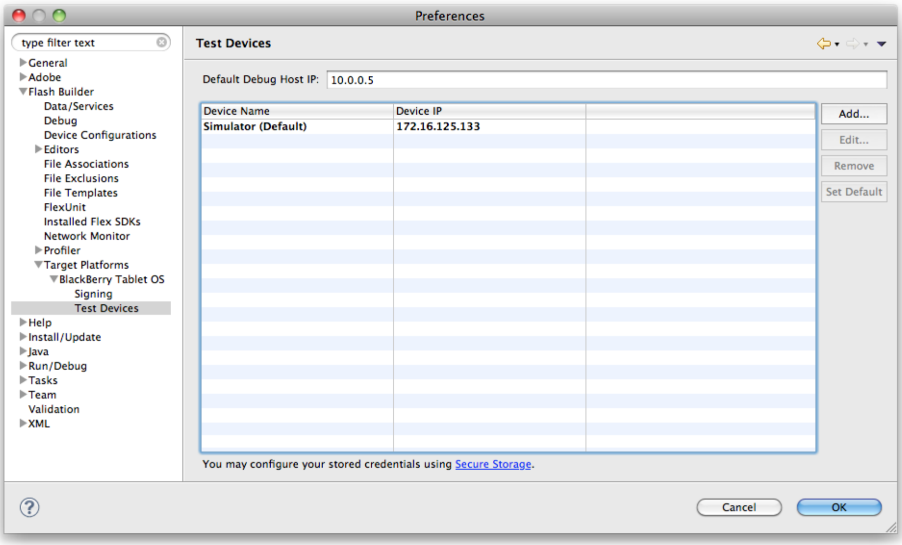
Enable application installation using software development tools.

Waiting for debug token upload...

Expires in 10 Days

To return to your computer, press Control-⌘

Figure 1-18. Waiting for Debug Token



Test Devices

Default Debug Host IP: 10.0.0.5

Device Name	Device IP
Simulator (Default)	172.16.125.133

You may configure your stored credentials using Secure Storage.

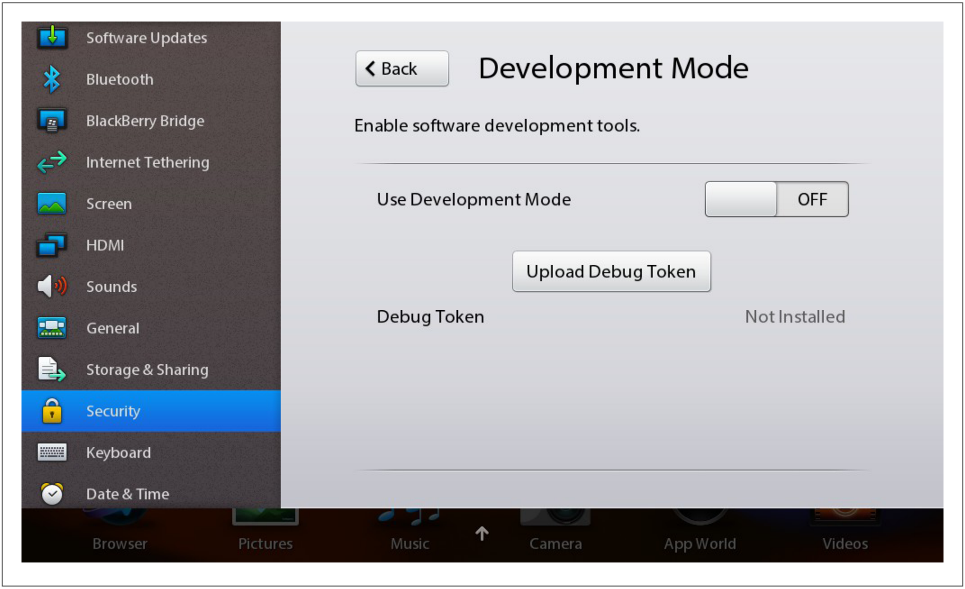
Figure 1-19. Debug Token added

Setup Device

Connect your device via USB. Click on the gear in the upper-right corner and then click security. Turn on development mode (see Figure 1-20). Now go back to the home screen and click the little person icon to read your IP address (see Figure 1-21).

Once you have your IP address, complete the Add BlackBerry Tablet OS Test Device setup. Be sure to check the box that says Create a debug token and upload it to this device (see Figure 1-22). Once you say OK, you're prompted to set the device to accept the debug token. Figure 1-23 shows the warning dialog; Figure 1-24 shows the device waiting for the debug token; Figure 1-25 shows that the debug token has been added; and Figure 1-26 shows the debug token installed on the device.

Once this completes, you're ready to move on to "Reading and setting author information for debug" on page 18, where you will set the author and author ID.



Development Mode

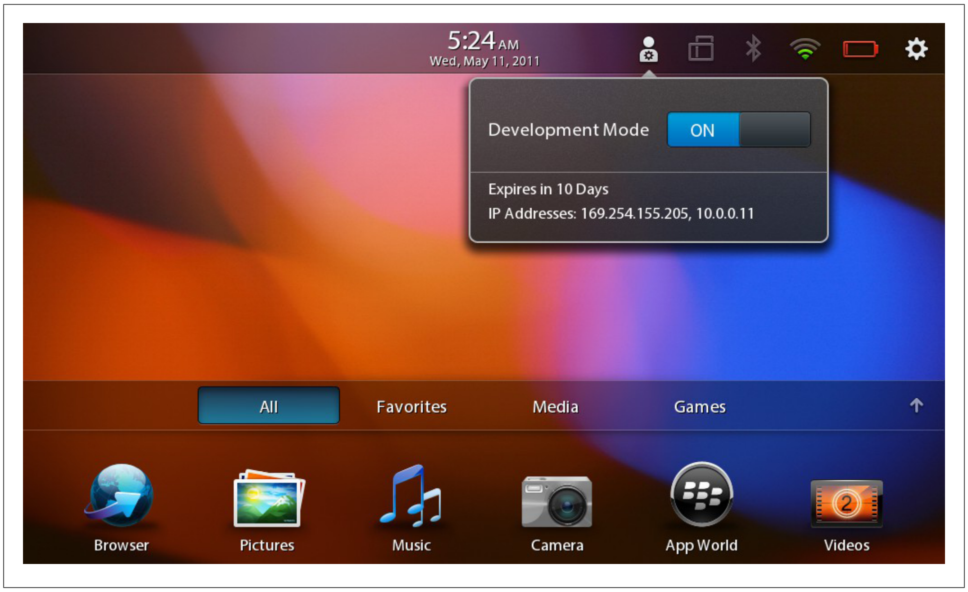
Enable software development tools.

Use Development Mode — OFF

Upload Debug Token

Debug Token — Not Installed

Figure 1-20. Turn on Development Mode in Device

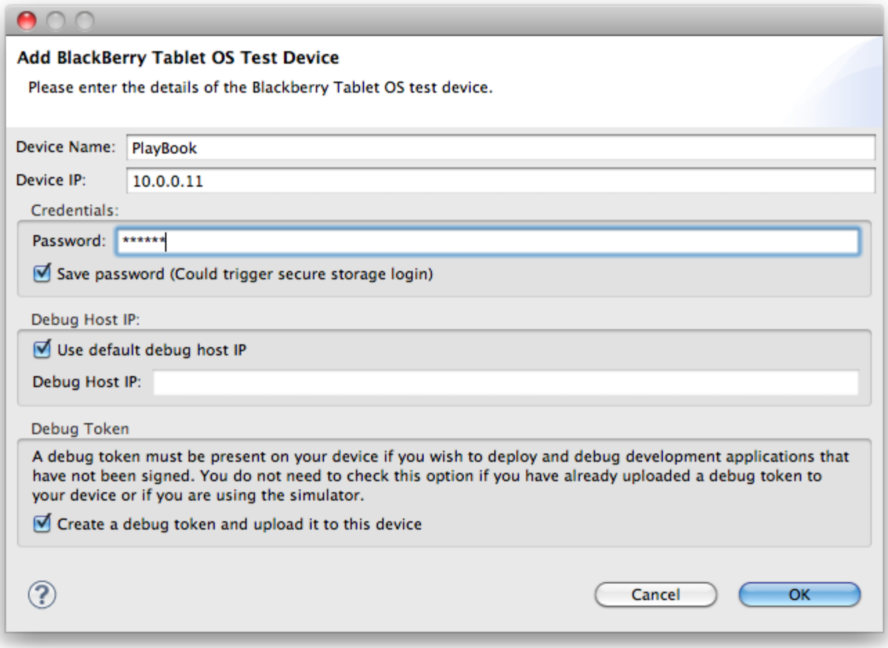


Development Mode — ON

Expires in 10 Days

IP Addresses: 169.254.155.205, 10.0.0.11

Figure 1-21. Read IP Address



Add BlackBerry Tablet OS Test Device

Please enter the details of the Blackberry Tablet OS test device.

Device Name: PlayBook

Device IP: 10.0.0.11

Credentials:

Password: ******

☑ Save password (Could trigger secure storage login)

Debug Host IP:

☑ Use default debug host IP

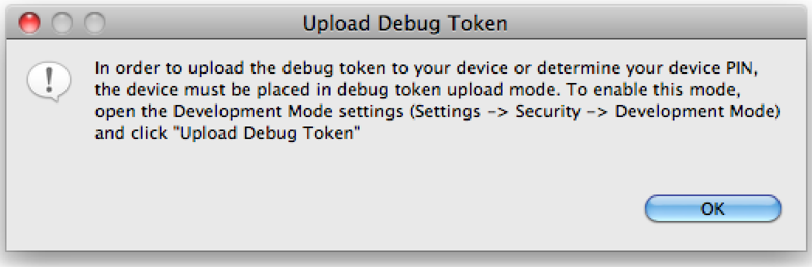
Debug Host IP:

Debug Token

A debug token must be present on your device if you wish to deploy and debug development applications that have not been signed. You do not need to check this option if you have already uploaded a debug token to your device or if you are using the simulator.

☑ Create a debug token and upload it to this device

Figure 1-22. Complete Form with IP Address of Device



Upload Debug Token

In order to upload the debug token to your device or determine your device PIN, the device must be placed in debug token upload mode. To enable this mode, open the Development Mode settings (Settings -> Security -> Development Mode) and click "Upload Debug Token"

Figure 1-23. Upload Debug Token warning

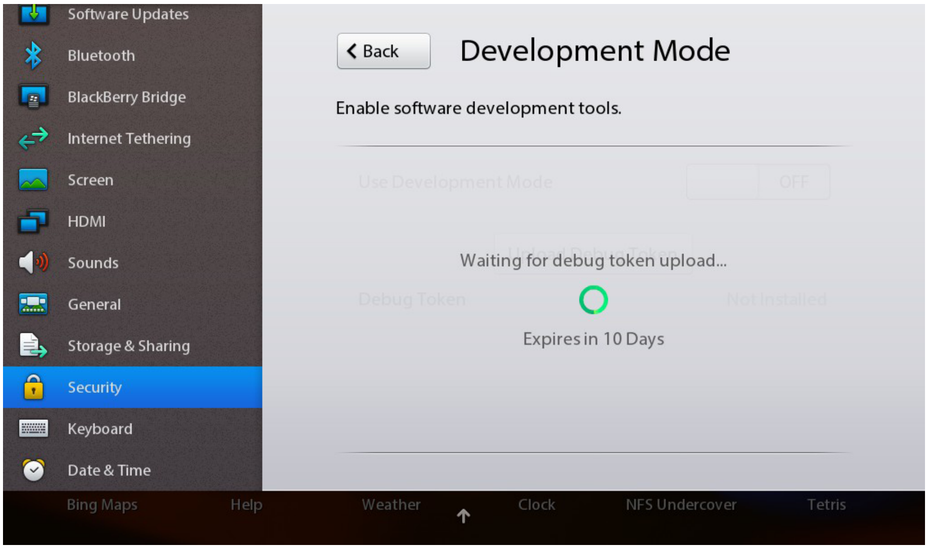
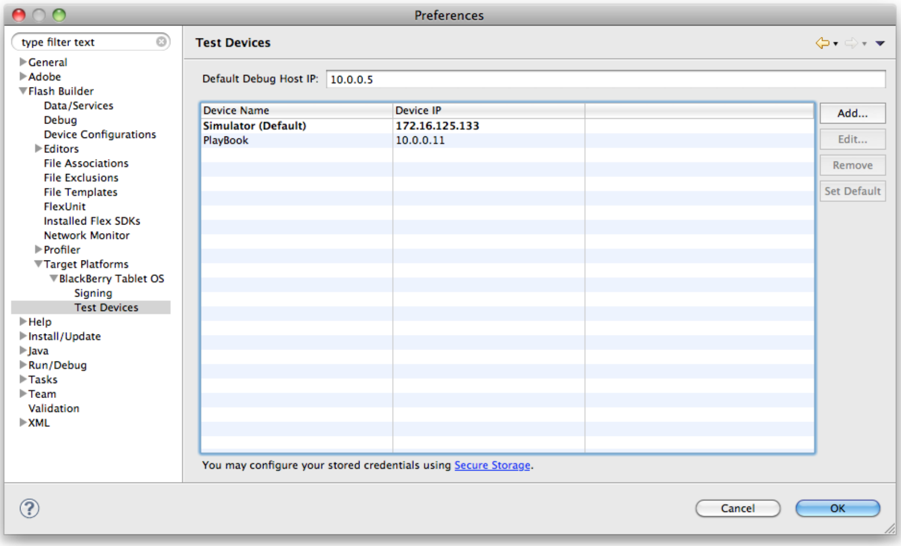


Figure 1-24. Waiting for Debug Token

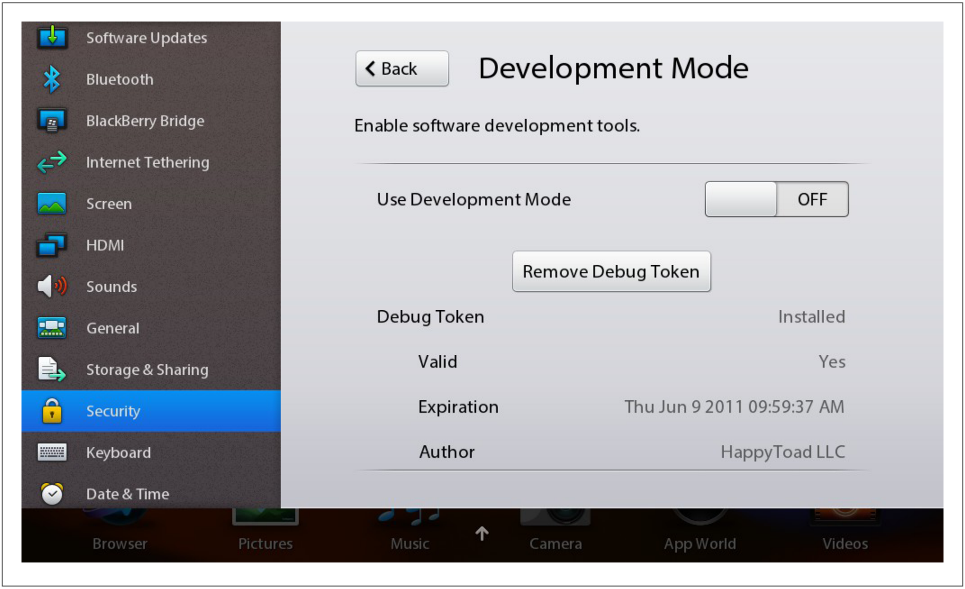


Default Debug Host IP: 10.0.0.5

Device Name	Device IP
Simulator (Default)	172.16.125.133
PlayBook	10.0.0.11

You may configure your stored credentials using Secure Storage.

Figure 1-25. Debug Token added



Development Mode

Enable software development tools.

Use Development Mode: OFF

Remove Debug Token

Debug Token: Installed

Valid: Yes

Expiration: Thu Jun 9 2011 09:59:37 AM

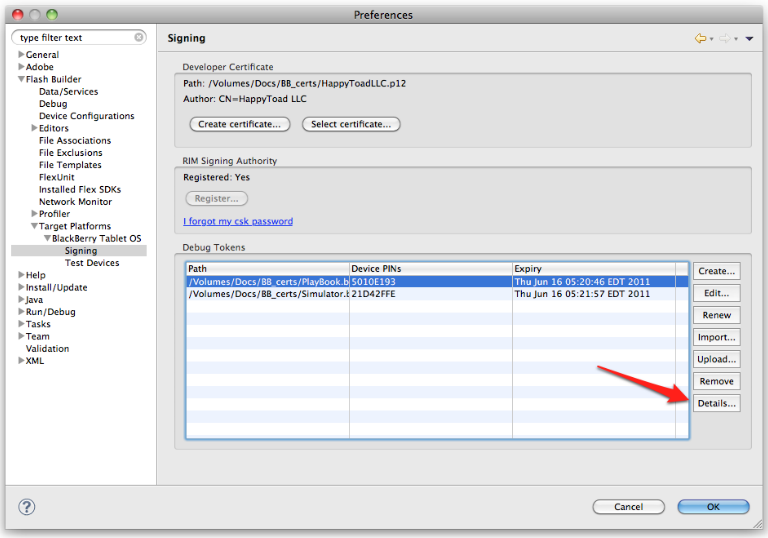
Author: HappyToad LLC

Figure 1-26. Debug Token installed on device

Reading and setting author information for debug

Before you can run your application, you need to add your author name and ID to the *blackberry-tablet.xml* file. To read these values, go to Preferences→BlackBerry Tablet OS→Signing. Highlight one of your Debug Tokens and click the Details button (see Figure 1-27). Figure 1-28 shows the author information. Copy the author name and author ID into the *blackberry-tablet.xml* file as shown in the following code.

```
<?xml version="1.0" encoding="UTF-8"?>
<qnx>
   <author>HappyToad LLC</author>
   <authorId>gYAAgBCK5G6OOhJ_Alo1WGVOfks</authorId>
   <buildId>349</buildId>
   <platformVersion>1.0.0.0</platformVersion>
</qnx>
```



Signing

Developer Certificate

Path: /Volumes/Docs/BB_certs/HappyToadLLC.p12

Author: CN=HappyToad LLC

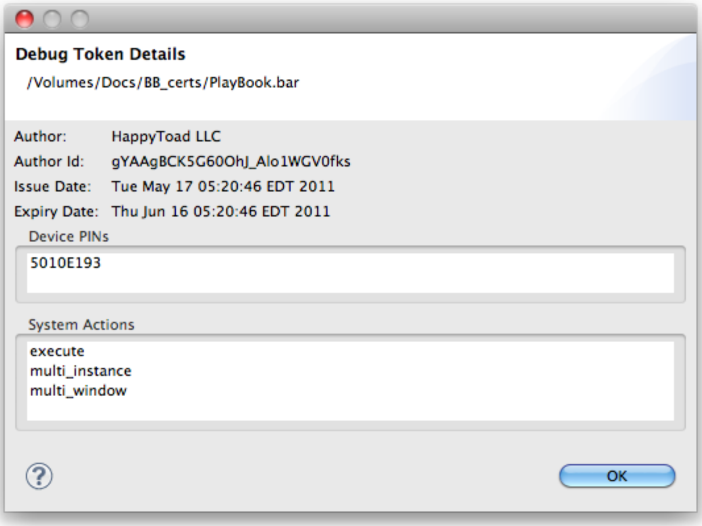
RIM Signing Authority

Registered: Yes

Debug Tokens

Path	Device PINs	Expiry
/Volumes/Docs/BB_certs/PlayBook.b	5010E193	Thu Jun 16 05:20:46 EDT 2011
/Volumes/Docs/BB_certs/Simulator.t	21D42FFE	Thu Jun 16 05:21:57 EDT 2011

Figure 1-27. Select debug token then click details



Debug Token Details

/Volumes/Docs/BB_certs/PlayBook.bar

Author:	HappyToad LLC
Author Id:	gYAAgBCK5G60OhJ_Alo1WGV0fks
Issue Date:	Tue May 17 05:20:46 EDT 2011
Expiry Date:	Thu Jun 16 05:20:46 EDT 2011

Device PINs

5010E193

System Actions

execute
multi_instance
multi_window

Figure 1-28. Copy debug token details

Now we can run the application. To do this, right-click on the *HelloWorld.mxml* file within the Package Explorer and select Run As→Mobile Application, as shown in Figure 1-29. Since this is the first time running this application, the Run Configurations window will open.

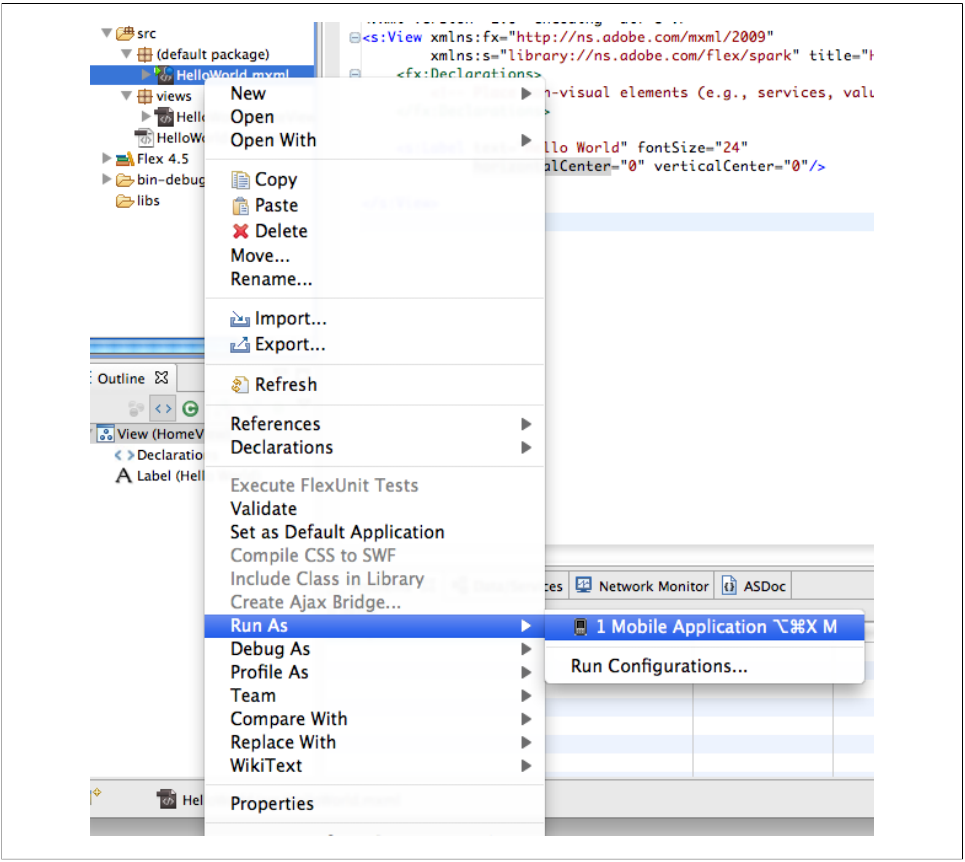
To run this application on the simulator, select “On device” and choose Simulator from the drop-down menu (see Figure 1-30).

To run this application on the device, select “On device” and choose PlayBook from the drop-down menu (see Figure 1-31).

Now click Apply and then click Run as you see the Hello World application launch.

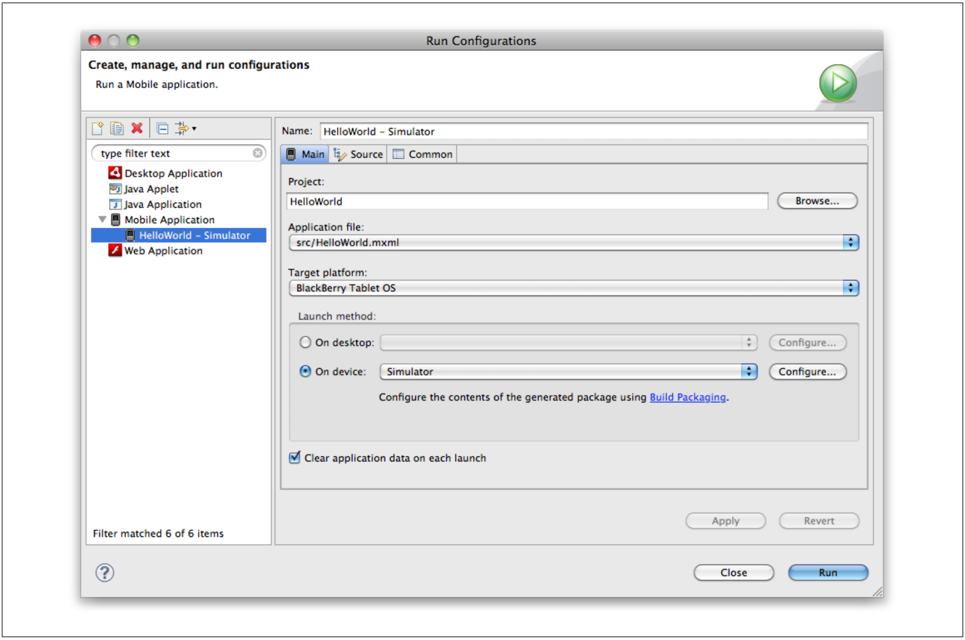
Figure 1-32 shows Hello World running on the simulator.

Figure 1-33 shows Hello World running on the device.



```
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="H
    <fx:Declarations>
                   n-visual elements (e.g., services, valu

                   llo World" fontSize="24"
                   alCenter="0" verticalCenter="0"/>
```

Figure 1-29. Run As Mobile Application


Run Configurations

Create, manage, and run configurations

Run a Mobile application.

Name: HelloWorld – Simulator

Project: HelloWorld

Application file: src/HelloWorld.mxml

Target platform: BlackBerry Tablet OS

Launch method:

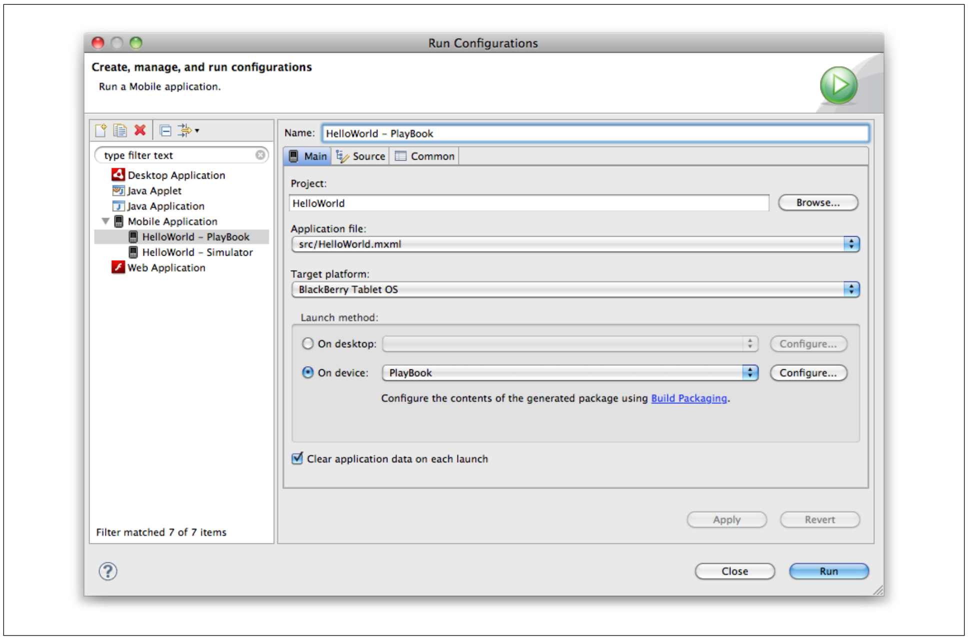
( ) On desktop:

(●) On device: Simulator

Configure the contents of the generated package using Build Packaging.

☑ Clear application data on each launch

Figure 1-30. Run on Simulator


Run Configurations

Create, manage, and run configurations

Run a Mobile application.

Name: HelloWorld – PlayBook

Project: HelloWorld

Application file: src/HelloWorld.mxml

Target platform: BlackBerry Tablet OS

Launch method:

( ) On desktop:

(●) On device: PlayBook

Configure the contents of the generated package using Build Packaging.

☑ Clear application data on each launch

Figure 1-31. Run on Device

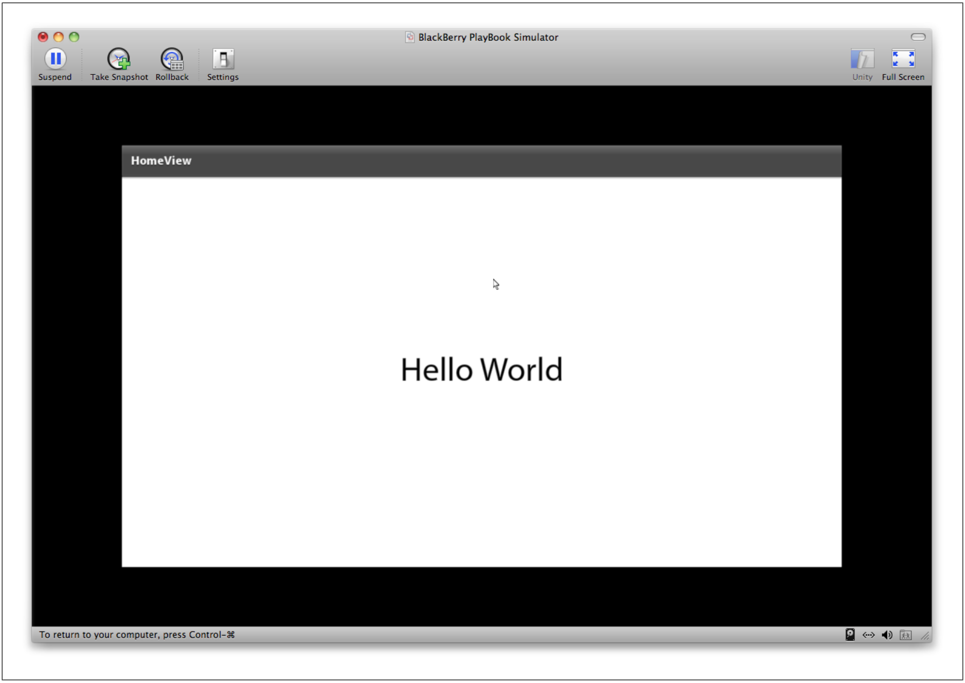

BlackBerry PlayBook Simulator
Suspend
Take Snapshot
Rollback
Settings
Unity
Full Screen
HomeView
Hello World
To return to your computer, press Control-⌘

Figure 1-32. Hello World running on simulator

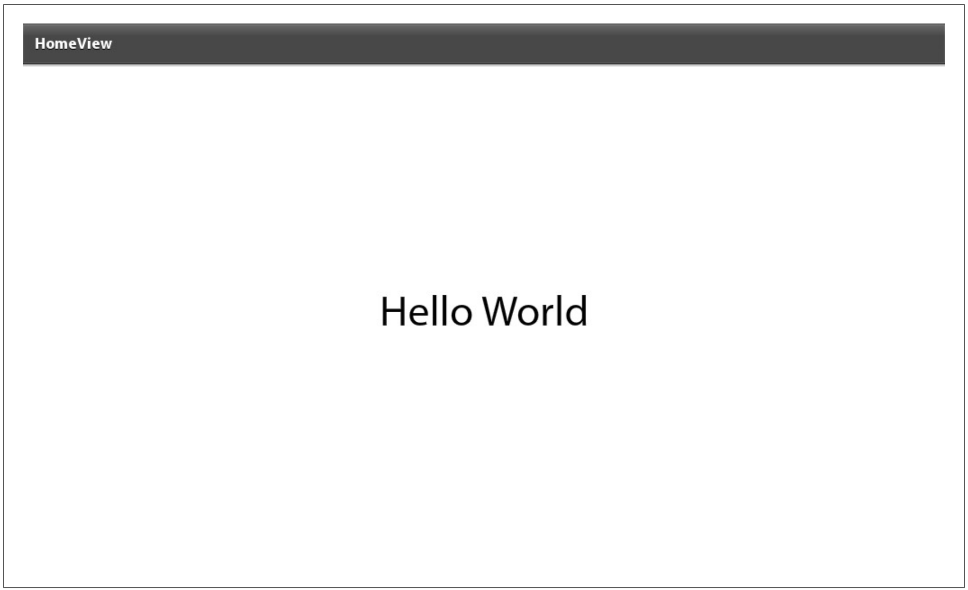


Figure 1-33. Hello World running on device

Congratulations, you have just created your first BlackBerry Tablet OS application with Adobe Flex 4.5.

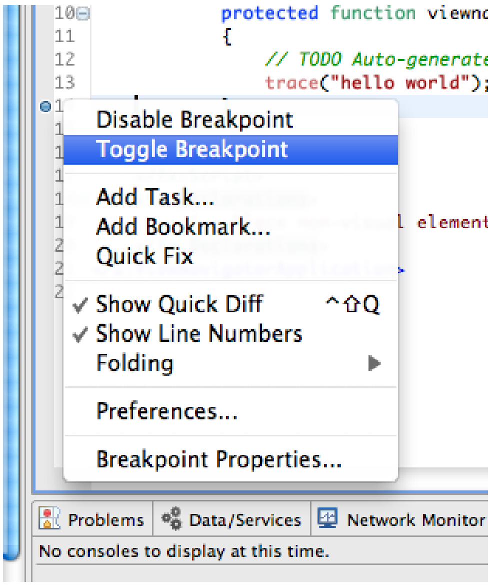
Debug a Flex Mobile Project

Now that you have created your Hello World application and run it via the Run Configurations window, you may wish to debug your application. Fortunately for you, the workflow for debugging a Flex Mobile application is the same as debugging any other Adobe Flex or Adobe AIR application.

Update the *HelloWorld.mxml* file to include a `creationComplete` handler as shown in the following code.

```
<?xml version="1.0" encoding="utf-8"?>
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.HelloWorldHomeView"
    creationComplete="viewnavigatorapplication1_creationCompleteHandler(event)">
    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;

            protected function viewnavigatorapplication1_creationCompleteHandler
            (event:FlexEvent):void
            {
                // TODO Auto-generated method stub
                trace("hello world");
            }

        ]]>
    </fx:Script>
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
</s:ViewNavigatorApplication>
```

We now need to toggle a breakpoint within the application on line 14 to demonstrate a debugging session. To do this, right-click on line 14 within Flash Builder and select Toggle Breakpoint from the context menu. Figure 1-34 shows this process. A small blue dot will appear in the gutter, showing that the break point is enabled.


```
protected function viewnc
{
    // TODO Auto-generate
    trace("hello world");
```

No consoles to display at this time.

Figure 1-34. Toggle a Breakpoint

We're now ready to debug this application. To do this, right-click on the *Hello-World.mxml* file within the Package Explorer, then select Debug As→Mobile Application, as shown in Figure 1-35. Since this is the first time debugging this application, the Debug Configurations window will open. To debug this using the Flash builder emulator, select "On device" as the Launch method and select a device from the drop-down menu, as shown in Figure 1-36.

When asked if you would like to switch to the Flash Builder debug perspective, select Yes (see Figure 1-37). Figure 1-38 shows the application paused on line 14 within Flash Builder's debug perspective. You can see the trace message within the console panel. To allow the application to complete, click the Resume button.

Congratulations, you have just completed your first Flash Builder debug session for a Flex Mobile application.

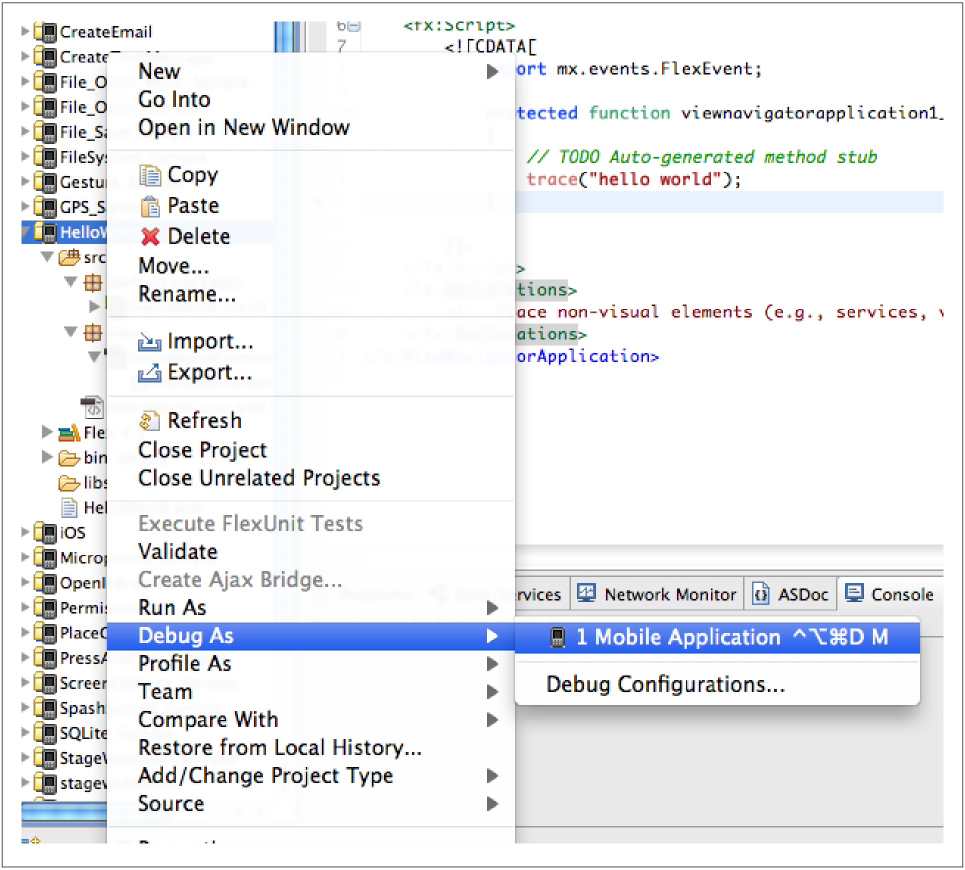
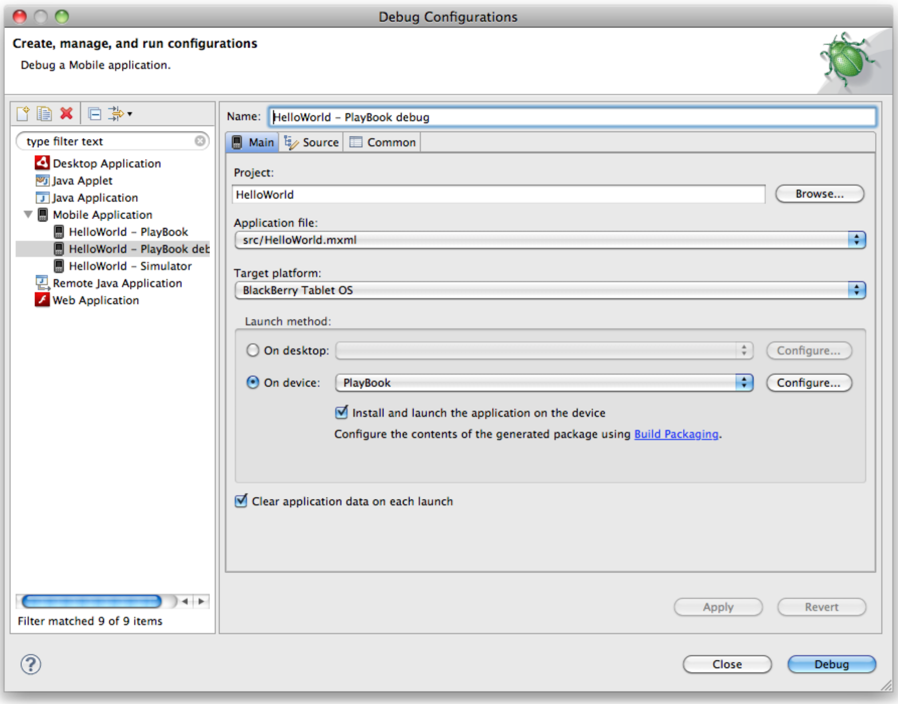


Figure 1-35. Debug As Mobile Application



Name: HelloWorld – PlayBook debug

Project: HelloWorld

Application file: src/HelloWorld.mxml

Target platform: BlackBerry Tablet OS

Launch method:

On desktop:

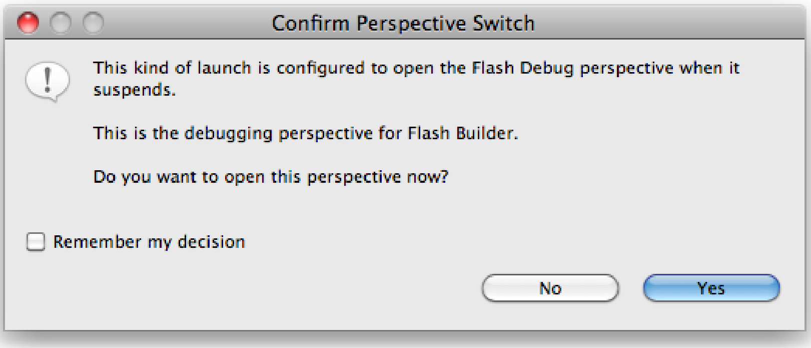
On device: PlayBook

☑ Install and launch the application on the device

Configure the contents of the generated package using Build Packaging.

☑ Clear application data on each launch

Figure 1-36. Debug Configurations Window



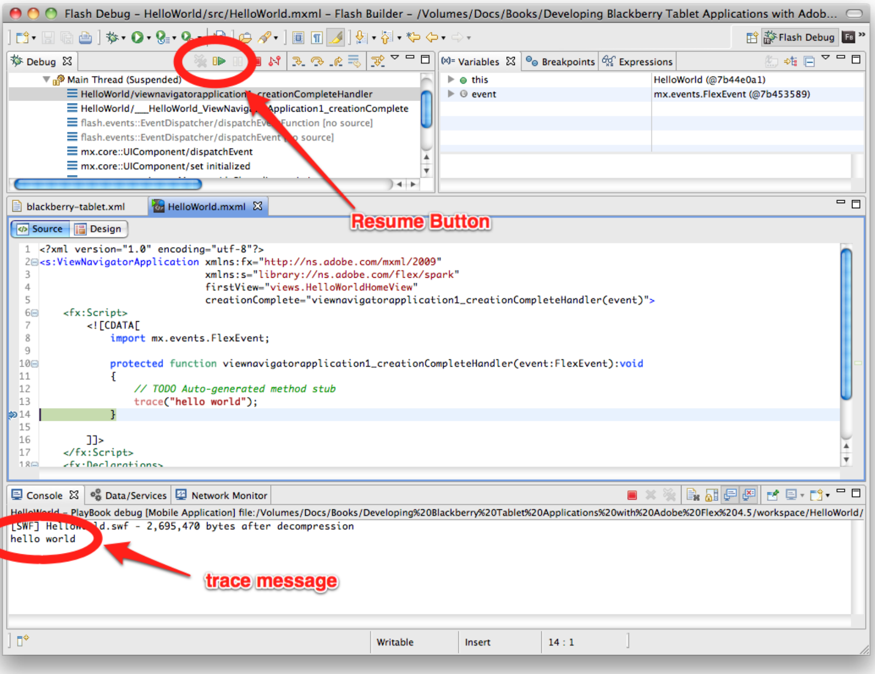
This kind of launch is configured to open the Flash Debug perspective when it suspends.

This is the debugging perspective for Flash Builder.

Do you want to open this perspective now?

☐ Remember my decision

Figure 1-37. Confirm switch to debug perspective



```
Main Thread (Suspended)
    HelloWorld/viewnavigatorapplication1_creationCompleteHandler
    HelloWorld/___HelloWorld_ViewNavigatorApplication1_creationComplete
    flash.events::EventDispatcher/dispatchEventFunction [no source]
    flash.events::EventDispatcher/dispatchEvent [no source]
    mx.core::UIComponent/dispatchEvent
    mx.core::UIComponent/set initialized
```

this	HelloWorld (@7b44e0a1)
event	mx.events.FlexEvent (@7b453589)

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
3                              xmlns:s="library://ns.adobe.com/flex/spark"
4                              firstView="views.HelloWorldHomeView"
5                              creationComplete="viewnavigatorapplication1_creationCompleteHandler(event)">
6      <fx:Script>
7          <![CDATA[
8              import mx.events.FlexEvent;
9
10             protected function viewnavigatorapplication1_creationCompleteHandler(event:FlexEvent):void
11             {
12                 // TODO Auto-generated method stub
13                 trace("hello world");
14             }
15
16         ]]>
17     </fx:Script>
18     <fx:Declarations>
```

```
HelloWorld – PlayBook debug [Mobile Application] file:/Volumes/Docs/Books/Developing%20Blackberry%20Tablet%20Applications%20with%20Adobe%20Flex%204.5/workspace/HelloWorld/
[SWF] HelloWorld.swf - 2,695,470 bytes after decompression
hello world
```

Figure 1-38. Hello World application paused on line 14

CHAPTER 2

Application Layouts

When creating a Flex Mobile project, you have three choices for your layout: Blank Application, View-Based Application, and Tabbed Application (see Figure 2-1). The selection you make when choosing a layout will determine which files Flash Builder 4.5 will auto-generate. The View-Based Application and Tabbed Application types come with built-in navigation frameworks. Let's explore each of these.

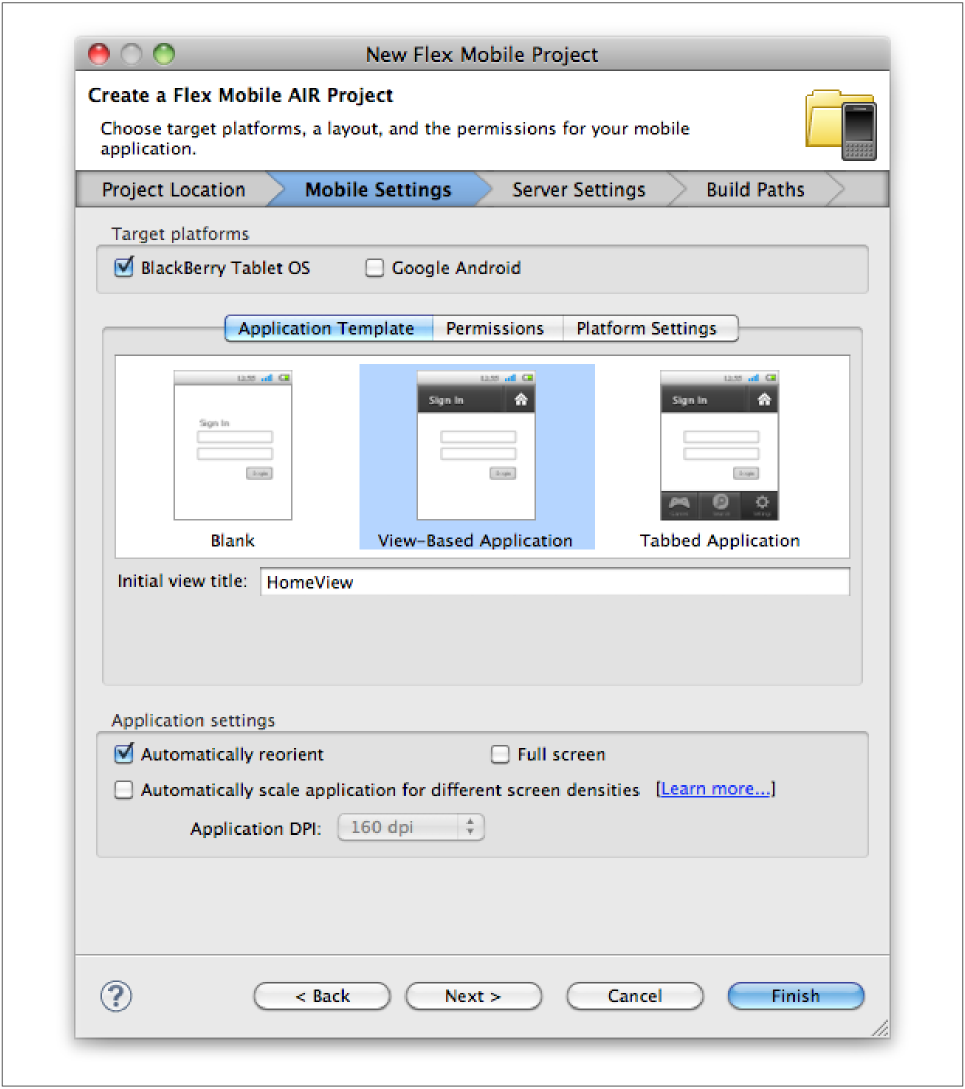
Blank Application

The Blank Application layout is best used when you're planning to build your own custom navigation. Choosing this option when creating a new Flex Mobile application within Flash Builder 4.5 will create a only the main application file, as shown in the following code.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark">
     <fx:Declarations>
          <!-- Place non-visual elements (e.g., services, value objects) here -->
     </fx:Declarations>
</s:Application>
```

In the following code, I have added a simple `Label`. You can see the results in Figure 2-2.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark">
     <fx:Declarations>
          <!-- Place non-visual elements (e.g., services, value objects) here -->
     </fx:Declarations>
     <s:Label text="Blank" fontSize="36"
             horizontalCenter="0" verticalCenter="0"/>
</s:Application>
```



New Flex Mobile Project

Create a Flex Mobile AIR Project

Choose target platforms, a layout, and the permissions for your mobile application.

Target platforms

[✓] BlackBerry Tablet OS [ ] Google Android

Initial view title: HomeView

Application settings

[✓] Automatically reorient [ ] Full screen

[ ] Automatically scale application for different screen densities [Learn more...]

Application DPI: 160 dpi

Figure 2-1. Layout options

Figure 2-2. Blank application layout on device

View-Based Application

The View-Based Application adds the concept of a navigator, which is a built-in navigation framework built specifically for use within mobile applications. The navigator will manage the screens within your application. Creating a new View-Based Application within Flash Builder 4.5 results in the generation of two files. These files are the main application file as well as the default view that will be shown within your application. Unlike the Blank Application, where the main application file was created with the `<s:Application>` as the parent, a View-Based Application uses the new `<s:ViewNavigatorApplication>` as its parent, as shown in the following code.

```
<?xml version="1.0" encoding="utf-8"?>
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark" firstView="views.ViewBasedHomeView">
     <fx:Declarations>
          <!-- Place non-visual elements (e.g., services, value objects) here -->
     </fx:Declarations>
</s:ViewNavigatorApplication>
```

The second file that is created is the default view, which is automatically placed in a package named Views. In this case, it is named ViewBasedHomeView and is automatically set as the `firstView` property of the ViewNavigatorApplication. The auto-generated code for this file is shown in the following code.

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
          xmlns:s="library://ns.adobe.com/flex/spark" title="HomeView">
     <fx:Declarations>
          <!-- Place non-visual elements (e.g., services, value objects) here -->
     </fx:Declarations>
</s:View>
```

Figure 2-3 shows the View-Based Application after adding a *Label* to the *ViewBasedHomeView*. As you can see, the navigation framework automatically provides a header and places the title of the current view in that header.

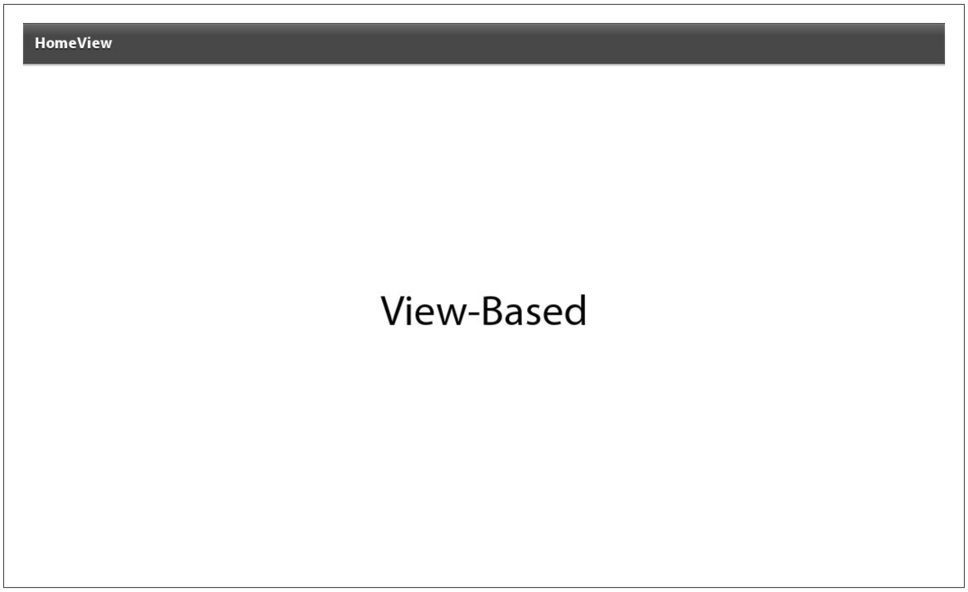


Figure 2-3. View-Based Application

Now let's explore the navigator a bit. I have created a second view for my application named SecondView. I updated the ViewBasedHomeView to have a `Button` and also added a `Button` to the `SecondView` as shown in the following code. As you can see, each view contains a `Button` with a similar `clickHandler`. The `clickHandler` simply calls the `pushView` function on the navigator and passes in the view you want the user to navigate to. The Home View will navigate to the Second View and the Second View will navigate to the Home View. Between each view, a transition is automatically played, and the title of the view is reflected in the navigation bar. This can be seen in Figure 2-4 and Figure 2-5.

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="HomeView">

    <fx:Script>
        <![CDATA[
            protected function button1_clickHandler(event:MouseEvent):void
            {
                navigator.pushView(views.SecondView);
            }
        ]]>
    </fx:Script>

    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:Button label="Go To Second View"
```

```
                 horizontalCenter="0" verticalCenter="0"
                 click="button1_clickHandler(event)"/>
</s:View>
```

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
          xmlns:s="library://ns.adobe.com/flex/spark" title="SecondView">

     <fx:Script>
          <![CDATA[
               protected function button1_clickHandler(event:MouseEvent):void
               {
                    navigator.pushView(views.ViewBasedHomeView);
               }
          ]]>
     </fx:Script>

     <fx:Declarations>
          <!-- Place non-visual elements (e.g., services, value objects) here -->
     </fx:Declarations>

     <s:Button label="Go To Home View"
                 horizontalCenter="0" verticalCenter="0"
                 click="button1_clickHandler(event)"/>
</s:View>
```

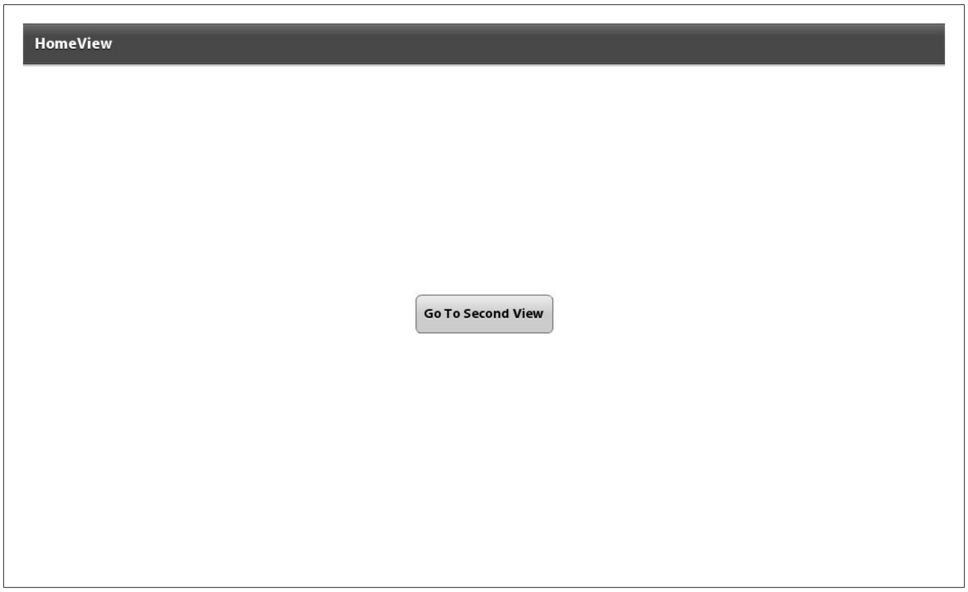


Figure 2-4. HomeView

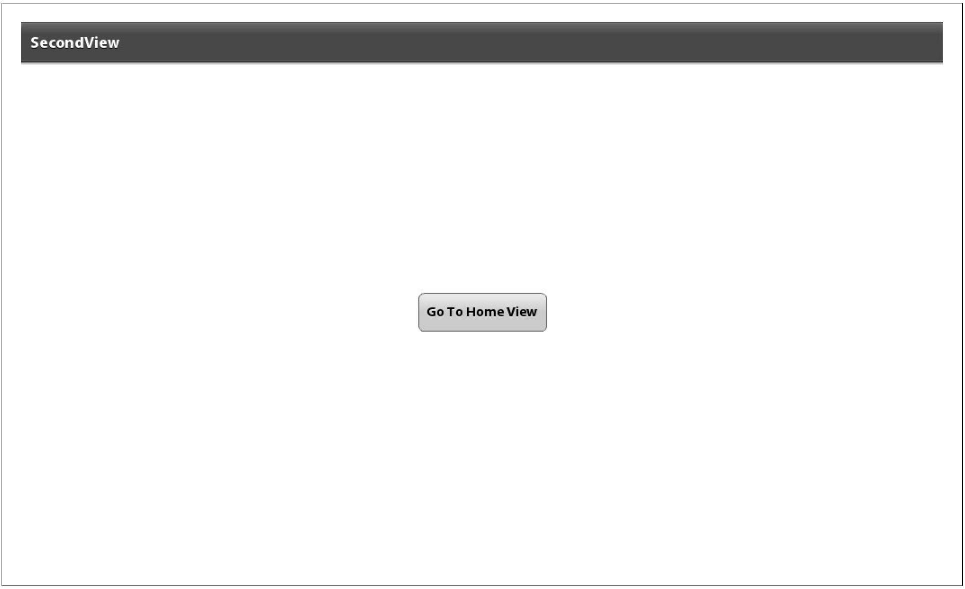


Figure 2-5. Second View

The navigator has additional methods for moving between views within your application. They are as follows:

`navigator.popAll()`
: Removes all the views from the navigator stack. This method changes the display to a blank screen.

`navigator.popToFirstView()`
: Removes all views except the bottom view from the navigation stack. The bottom view is the one that was first pushed onto the stack.

`navigator.popView()`
: Pops the current view off the navigation stack. The current view is represented by the top view on the stack. The previous view on the stack becomes the current view.

`navigator.pushView()`
: Pushes a new view onto the top of the navigation stack. The view pushed onto the stack becomes the current view.

Each of the methods described above allow for a transition to be passed in. By default, they will use a wipe transition. All pop actions will wipe from left to right, while a push action will wipe from right to left.

Another important item to note on the `navigator.pushView()` is the ability to pass an object into the method call. I have updated the following sample to demonstrate how to use this within your applications.

The ViewBasedHomeView shown next now includes a piece of String data—“Hello from Home View”—within the pushView() method. The SecondView has also been updated to include a new Label, which is bound to the data object. This data object will hold the value of the object passed in through the pushView() method. Figure 2-6 shows the how the SecondView is created with the Label showing our new message.

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
          xmlns:s="library://ns.adobe.com/flex/spark" title="HomeView">

     <fx:Script>
          <![CDATA[
               protected function button1_clickHandler(event:MouseEvent):void
               {
                    navigator.pushView(views.SecondView, "Hello from Home View");
               }
          ]]>
     </fx:Script>

     <fx:Declarations>
          <!-- Place non-visual elements (e.g., services, value objects) here -->
     </fx:Declarations>
     <s:Button label="Go To Second View"
                 horizontalCenter="0" verticalCenter="0"
                 click="button1_clickHandler(event)"/>
</s:View>

<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
          xmlns:s="library://ns.adobe.com/flex/spark" title="SecondView">

     <fx:Script>
          <![CDATA[
               protected function button1_clickHandler(event:MouseEvent):void
               {
                    navigator.pushView(views.ViewBasedHomeView);
               }
          ]]>
     </fx:Script>

     <fx:Declarations>
          <!-- Place non-visual elements (e.g., services, value objects) here -->
     </fx:Declarations>
     <s:Label text="{data}" horizontalCenter="0" top="30"/>
     <s:Button label="Go To Home View"
                 horizontalCenter="0" verticalCenter="0"
                 click="button1_clickHandler(event)"/>
</s:View>
```

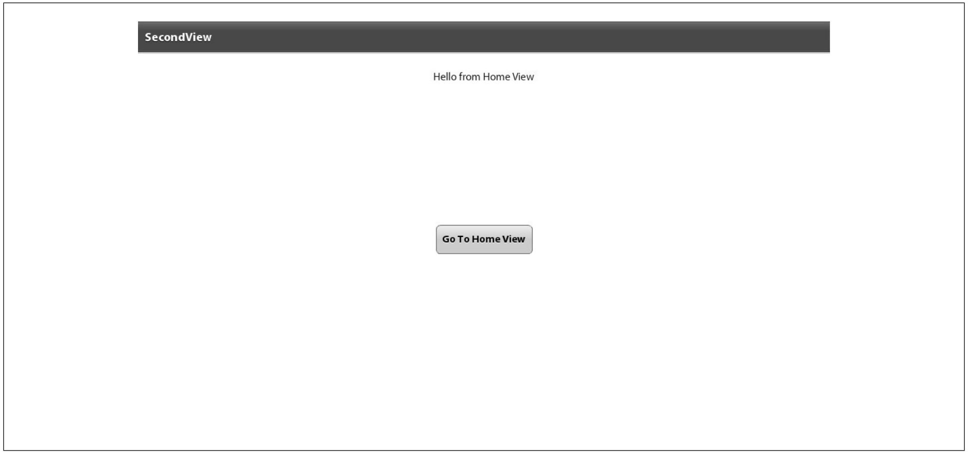


Figure 2-6. pushView() with data passed through

The navigation bar at the top of a View-Based Application allows you to set specific elements. These are the `navigationContent` and the `actionContent`. By setting these elements, your application can include a common navigation throughout. The code that follows shows an example of the View-Based Application's main file updated with these new elements. You will notice that the `navigationContent` and `actionContent` and `Spark` components are defined in MXML. Within each, I have included a `Button`. Each `Button` has a `clickHandler` that includes a call to one of the navigator methods. The `Button` that has the label "Home" has a `clickHandler` that includes a call to the `popToFirstView()` method, which will always send the user back to the view that is defined in the `firstView` property of the `ViewNavigationApplication`. The `Button` that has the label "Back" has a `clickHandler` that includes a call to the `popView()` method, which will always send the user back to previous view in the stack. Note that, when using `popView()`, you will need to make sure your application is aware of where it is in the stack, as a call to `popView()` when the user is already on the `firstView` will send the user to a blank screen. Figure 2-7 shows the application, which now includes the new navigation elements within the navigation bar.

Although this example utilizes a *Button* component to demonstrate view navigation, best practices when developing mobile applications dictate that your application should rely on the device's native back button navigation.

```
<?xml version="1.0" encoding="utf-8"?>
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                                   xmlns:s="library://ns.adobe.com/flex/spark"
firstView="views.ViewBasedHomeView">

     <fx:Script>
```

```
          <![CDATA[
               protected function homeButton_clickHandler(event:MouseEvent):void
               {
                    navigator.popToFirstView();
               }

               protected function backButton_clickHandler(event:MouseEvent):void
               {
                    navigator.popView();
               }

          ]]>
     </fx:Script>

     <fx:Declarations>
          <!-- Place non-visual elements (e.g., services, value objects) here -->
     </fx:Declarations>

     <s:navigationContent>

          <s:Button id="homeButton" click="homeButton_clickHandler(event)"
                      label="Home"/>

     </s:navigationContent>

     <s:actionContent>
          <s:Button id="backButton" click="backButton_clickHandler(event)"
                      label="Back"/>
     </s:actionContent>
</s:ViewNavigatorApplication>
```

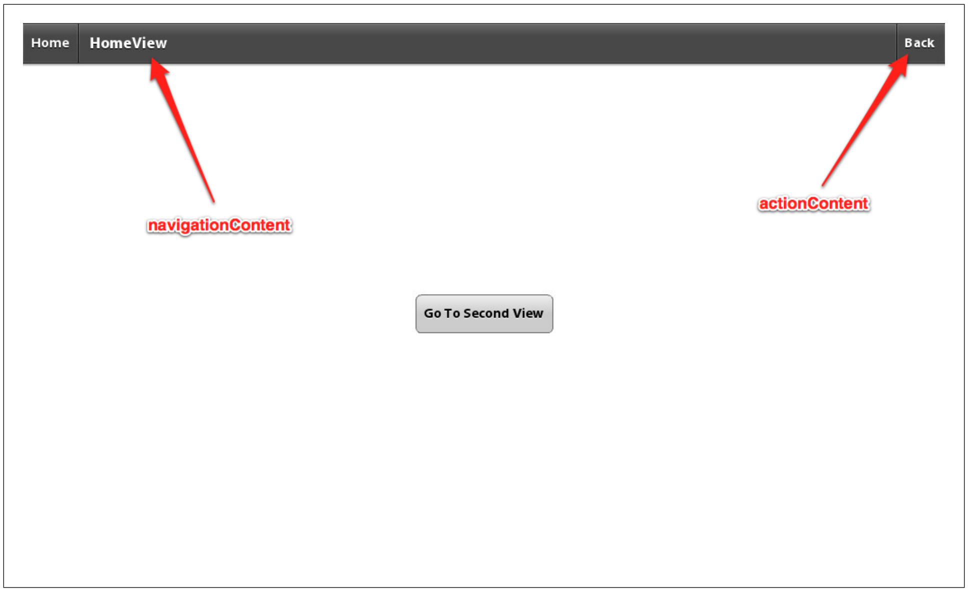


Figure 2-7. navigationContent and actionContent

View Life Cycle

The View class includes some new life cycle events specifically added for mobile applications. These events are important for application memory conservation.

- `FlexEvent.VIEW_ACTIVATE` is dispatched when the view has been activated.

  ```
  viewActivate="view1_viewActivateHandler(event)"
  ```

- `FlexEvent.VIEW_DEACTIVATE` is dispatched when the view has been deactivated.

  ```
  viewDeactivate="view1_viewDeactivateHandler(event)"
  ```

- `FlexEvent.REMOVING` is dispatched right before `FlexEvent.VIEW_DEACTIVATE` is dispatched when the view is about to be deactivated. Calling `preventDefault()` will cancel the screen change.

Although this life cycle is great for keeping the application's memory usage to a minimum, the default behavior to deactivate a view also destroys any data associated with that view. To preserve data so that it will be available if the user returns to that view, you can save the data to the `View.data` property.

If you would like to prevent a view from ever being deactivated, you can set the `destructionPolicy` attribute of the view to be never; it usually defaults to auto.

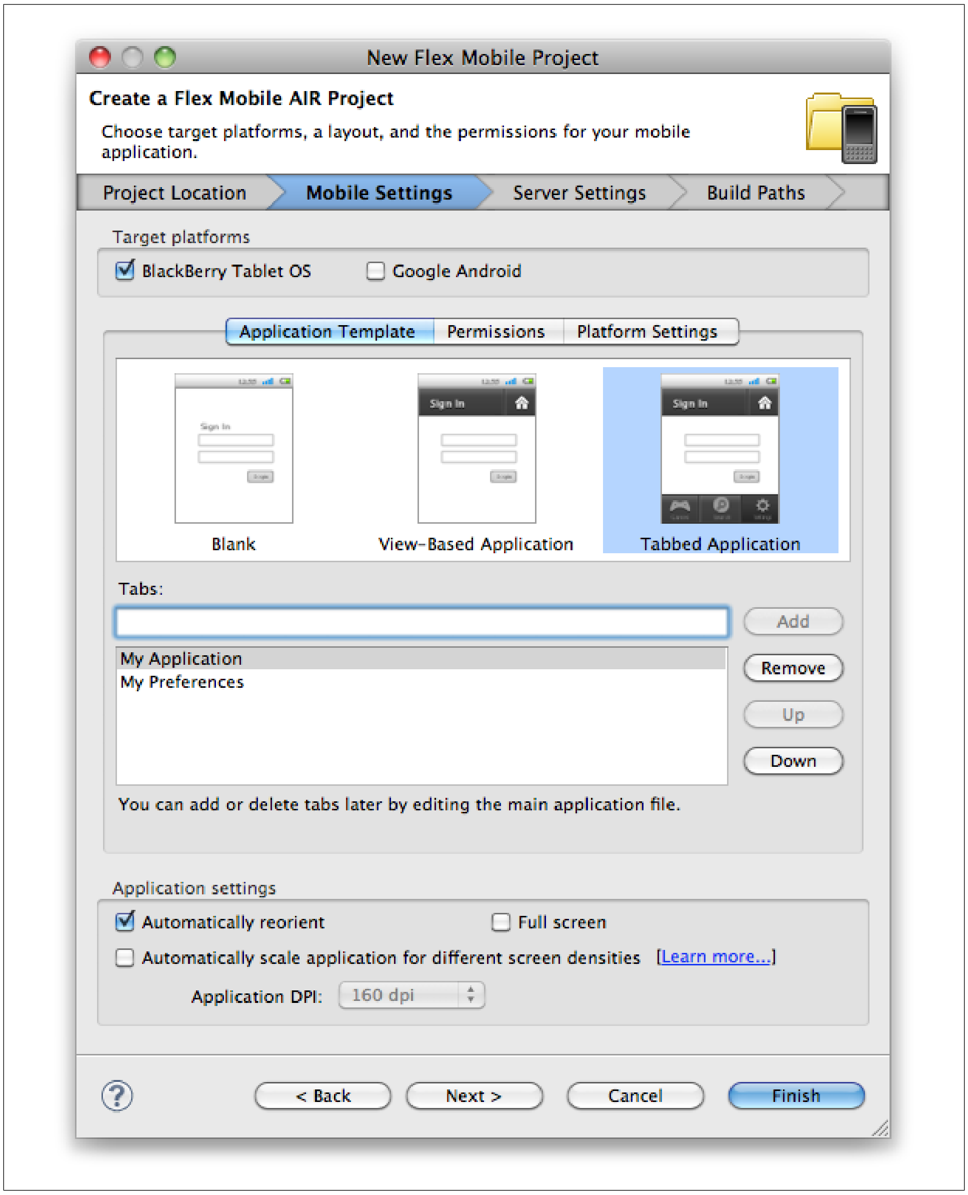
```
destructionPolicy="never"
```

Tabbed Application

The final option for application type is the Tabbed Application. Selecting Tabbed Application (see Figure 2-1) when creating a new Flex Mobile project will trigger Flash Builder to provide some additional functionality. As you can see in Figure 2-8, changing to Tabbed Application allows you to define your tabs within the new Flex Mobile project interface. In this example, I have added a My Application tab and a My Preferences tab. After clicking Finish, Flash Builder will create my new Tabbed Application, as well as the tabs I defined as views. The next code example shows the contents of my main application file named *Tabbed.mxml*. It is important to note that each of the views I defined as My Application and My Preferences are included as `ViewNavigator` objects. This means that they will have their own navigator objects and can include their own independent navigation, just as we had within the View-Based Applications we previously discussed. Figure 2-9 shows the running Tabbed application. Figure 2-10 shows the View-Based Application views we previously created within the My Application tab of the Tabbed application.

```
<?xml version="1.0" encoding="utf-8"?>
<s:TabbedViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                                  xmlns:s="library://ns.adobe.com/flex/spark">
     <s:ViewNavigator label="My Application" width="100%" height="100%"
firstView="views._MyApplicationView"/>
     <s:ViewNavigator label="My Preferences" width="100%" height="100%"
firstView="views._MyPreferencesView"/>
```

```
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
</s:TabbedViewNavigatorApplication>
```



Create a Flex Mobile AIR Project

Choose target platforms, a layout, and the permissions for your mobile application.

Target platforms

[✓] BlackBerry Tablet OS ☐ Google Android

Tabs:

My Application
My Preferences

You can add or delete tabs later by editing the main application file.

Application settings

[✓] Automatically reorient ☐ Full screen

☐ Automatically scale application for different screen densities

Application DPI: 160 dpi

Figure 2-8. Create a new Tabbed Application

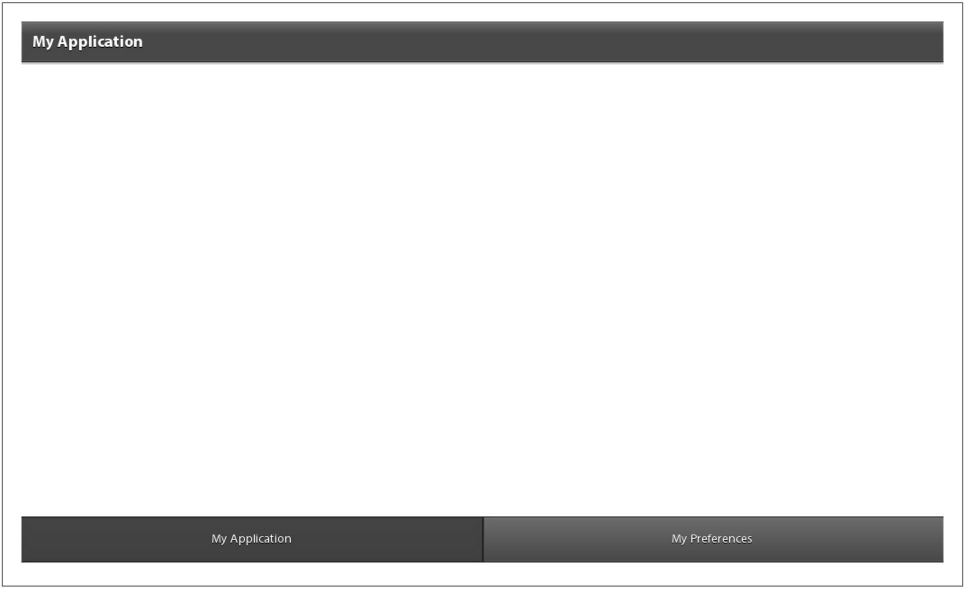


Figure 2-9. Tabbed Application

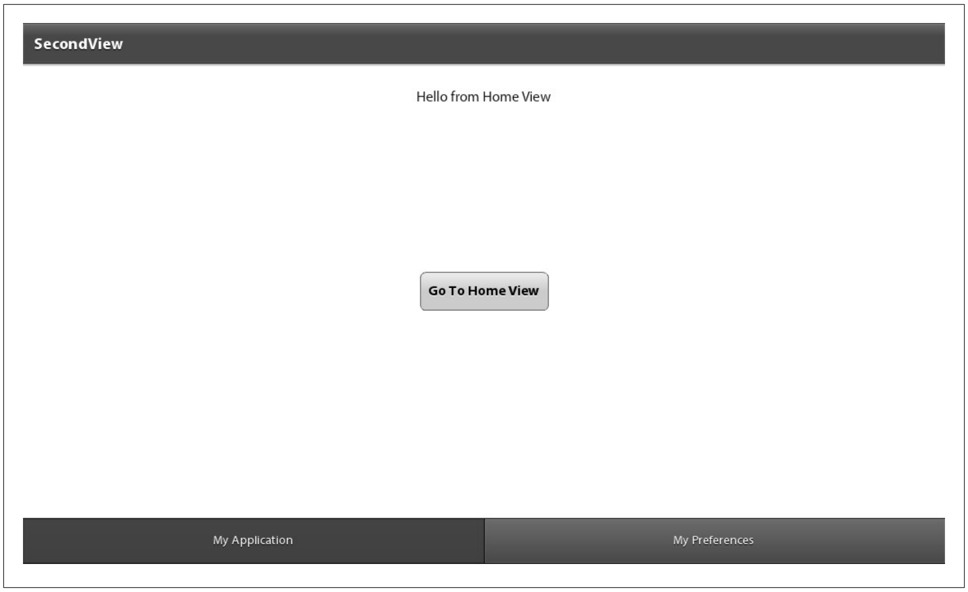


Figure 2-10. Tabbed Application with navigators

CHAPTER 3

Permissions and Configuration Settings

When creating a BlackBerry Tablet application, it is necessary to select the permissions that your application will require to operate. This chapter also covers several other configuration settings that are unique to BlackBerry Tablet applications.

Permissions

The AIR 2.6 release includes the permission options outlined in the following list; they can be selected within the new Flex Mobile project interface of Flash Builder 4.5 (see Figure 3-1). Figure 3-2 shows the warning the user will see when installing an application with permission requests.

`access_shared`
: Read and write files that are shared between all applications run by the current user.

`record_audio`
: Access the audio stream from the microphone.

`read_geolocation`
: Read the current location of the device.

`use_camera`
: Access the data coming from of cameras.

`access_internet`
: Use a WiFi, wired, or other connection to a destination that is not local.

`play_audio`
: Play an audio stream.

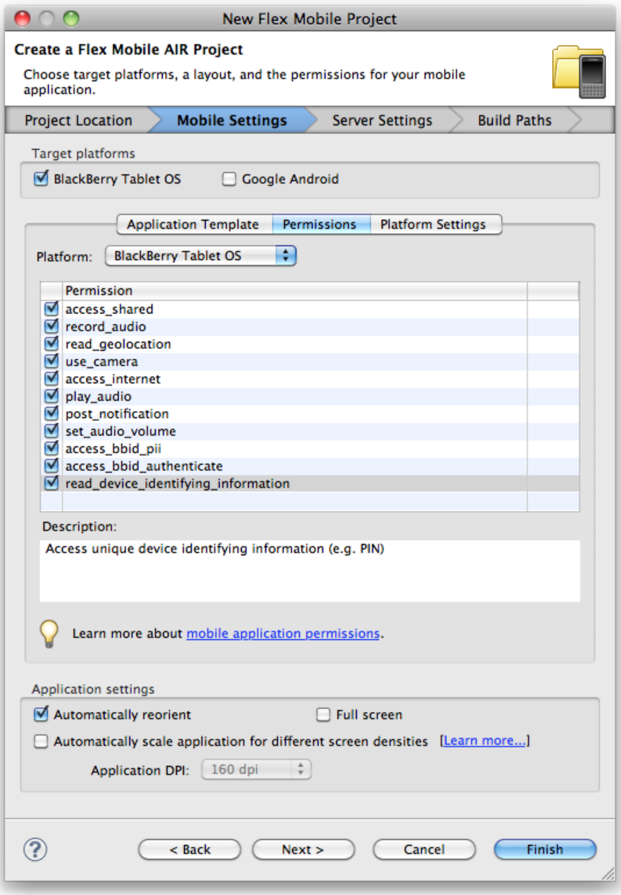
`post_notification`
: Post a notification to the Notifications area of the screen. Note: At the time of this writing, this API has not yet been documented by RIM.

`set_audio_volume`
: Change the volume of an audio stream being played. Note: At the time of this writing, this API has not yet been documented by RIM.

`access_bbid_pii`
: Access the BBID user's personally identifiable information. Note: At the time of this writing, this API has not yet been documented by RIM.

`access_bbid_authenticate`
: Authenticate the user to an external system. Note: At the time of this writing, this API has not yet been documented by RIM.

`read_device_identifying_information`
: Access unique device identifying information (e.g. PIN).



New Flex Mobile Project

Create a Flex Mobile AIR Project

Choose target platforms, a layout, and the permissions for your mobile application.

Project Location | **Mobile Settings** | Server Settings | Build Paths

Target platforms

☑ BlackBerry Tablet OS ☐ Google Android

Application Template | **Permissions** | Platform Settings

Platform: BlackBerry Tablet OS

	Permission
☑	access_shared
☑	record_audio
☑	read_geolocation
☑	use_camera
☑	access_internet
☑	play_audio
☑	post_notification
☑	set_audio_volume
☑	access_bbid_pii
☑	access_bbid_authenticate
☑	read_device_identifying_information

Description:

Access unique device identifying information (e.g. PIN)

Learn more about mobile application permissions.

Application settings

☑ Automatically reorient ☐ Full screen

☐ Automatically scale application for different screen densities [Learn more...]

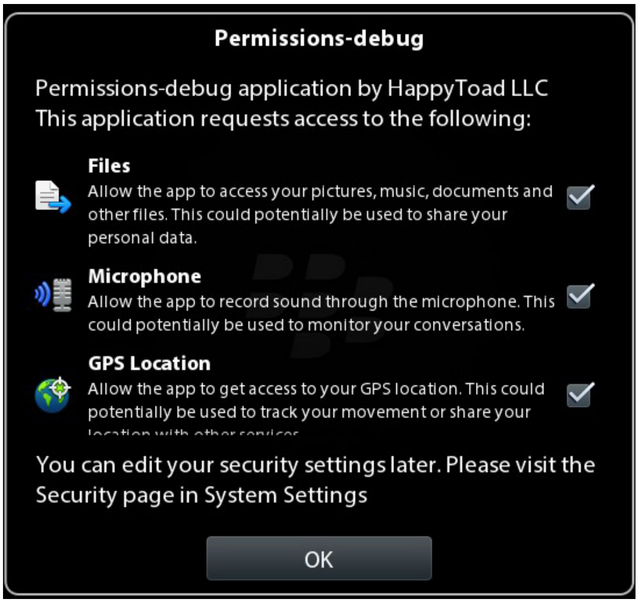
Application DPI: 160 dpi

Figure 3-1. Permission Selections

These permissions are also editable within the application's XML configuration file. Following is a sample of what these look like.

Manually editing the configurations within the application's XML configuration file is the only way to make permission changes once you have created the mobile project.

```
<?xml version="1.0" encoding="UTF-8"?>
<qnx>
   <icon>
      <image>blackberry-tablet-default-icon.png</image>
   </icon>

   <author>HappyToad LLC</author>
   <authorId>gYAAgBCK5G60OhJ_Alo1WGV0fks</authorId>
   <buildId>349</buildId>
   <platformVersion>1.0.0.0</platformVersion>
   <permission>access_shared</permission>
   <permission>record_audio</permission>
   <permission>read_geolocation</permission>
   <permission>use_camera</permission>
   <permission>access_internet</permission>
   <permission>play_audio</permission>
   <permission>post_notification</permission>
   <permission>set_audio_volume</permission>
   <permission>access_bbid_pii</permission>
   <permission>access_bbid_authenticate</permission>
   <permission>read_device_identifying_information</permission>
</qnx>
```


Permissions-debug

Permissions-debug application by HappyToad LLC
This application requests access to the following:

Files
Allow the app to access your pictures, music, documents and other files. This could potentially be used to share your personal data. ✓

Microphone
Allow the app to record sound through the microphone. This could potentially be used to monitor your conversations. ✓

GPS Location
Allow the app to get access to your GPS location. This could potentially be used to track your movement or share your ✓

You can edit your security settings later. Please visit the Security page in System Settings

Figure 3-2. Installer permission warnings

Other Configuration Settings

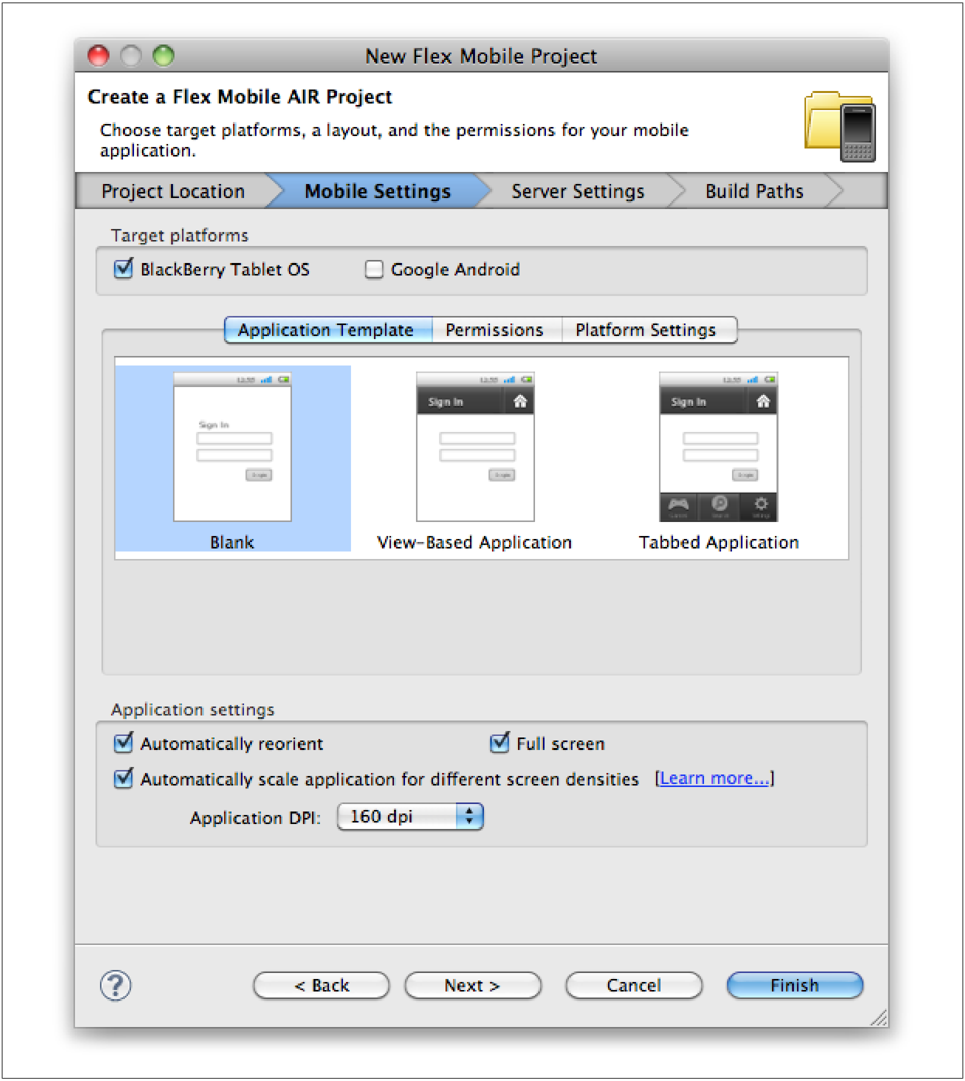
When creating a new Flex Mobile application, there are a few additional settings you can choose to configure. These include Automatically reorient, Full screen, and Automatically scale application for different screen densities. Figure 3-3 shows these options.

Automatically reorient

This option is set to true automatically unless you uncheck the box during your project creation. Setting this to true will allow the device to use its accelerometer to automatically switch between portrait and landscape.

This property can be edited at any time within the application's configuration file. This setting can also be changed programmatically while the application is running. See Chapter 5 for more information on this.

```
<autoOrients>false</autoOrients>
```


New Flex Mobile Project

Create a Flex Mobile AIR Project

Choose target platforms, a layout, and the permissions for your mobile application.

Target platforms

[✓] BlackBerry Tablet OS [ ] Google Android

Application settings

[✓] Automatically reorient [✓] Full screen

[✓] Automatically scale application for different screen densities [Learn more...]

Application DPI: 160 dpi

Figure 3-3. Additional Configuration Settings

Full Screen

Checking this box during your project creation will force your application to launch in full-screen mode. By default, this is set to false.

This property can be edited at any time within the application's configuration file. This setting can also be changed programmatically while the application is running. See Chapter 5 for more information on this.

```
<fullScreen>false</fullScreen>
```

Automatically scale application for different screen densities

Checking this box will allow your application to automatically scale for different screen densities. It will also allow you to set the default applicationDPI, which will be written to the main application file. The options for this value are 160, 240, and 320.

```
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
             xmlns:s="library://ns.adobe.com/flex/spark"
             applicationDPI="320">
```

Aspect Ratio

You have the option to force an application to run only in portrait or landscape mode. Uncommenting the `<aspectRatio>` node within the application's XML configuration file and setting its value to either landscape or portrait can accomplish this. This setting can also be changed programmatically while the application is running. See Chapter 5 for more information on this.

```
<aspectRatio>landscape</aspectRatio>
```

CHAPTER 4

Exploring the APIs

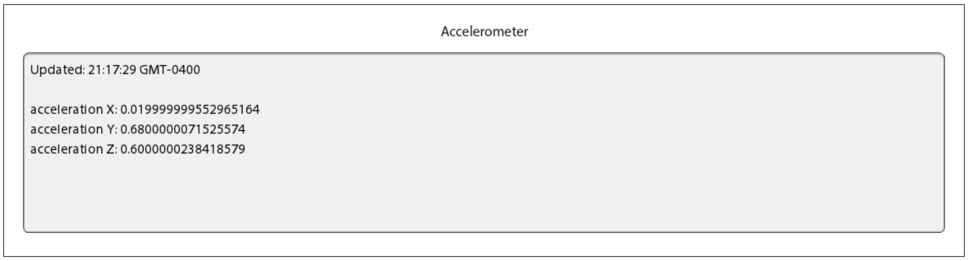
Now that you know how to create a new application with your choice of layout options and how to request application permissions, it's time to explore the ways in which your application can interact with the BlackBerry Tablet operating system. The AIR 2.6 release includes access to many BlackBerry Tablet OS features. These include the accelerometer, GPS unit, camera, camera roll, file system, and multi-touch screen.

Accelerometer

The accelerometer is a device that measures the speed or g-forces created when a device accelerates across multiple planes. The faster the device is moved through space, the higher the readings will be across the x, y, and z axes.

Let's review the following code. First, you will notice that there is a private variable named `accelerometer` declared of type `flash.sensors.Accelerometer`. Within the `applicationComplete` event of the application, an event handler function is called; it first checks to see if the device has an accelerometer by reading the static property of the `Accelerometer` class. If this property returns as true, a new instance of `Accelerometer` is created and an event listener of type `AccelerometerEvent.UPDATE` is added to handle updates. Upon update, the accelerometer information is read from the event and written to a `TextArea` within the `handleUpdate` function. The results are shown in Figure 4-1.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               applicationComplete="application1_applicationCompleteHandler(event)">
    <fx:Script>
        <![CDATA[
            import flash.sensors.Accelerometer;

            import mx.events.FlexEvent;

            private var accelerometer:Accelerometer;

            protected function application1_applicationCompleteHandler
(event:FlexEvent):void {
                if(Accelerometer.isSupported==true){
                    accelerometer = new Accelerometer();
                    accelerometer.addEventListener
                    (AccelerometerEvent.UPDATE,handleUpdate);
                } else {
                    status.text = "Accelerometer not supported";
                }

            }

            private function handleUpdate(event:AccelerometerEvent):void {
                info.text = "Updated: " + new Date().toTimeString() + "\n\n"
                + "acceleration X: " + event.accelerationX + "\n"
                + "acceleration Y: " + event.accelerationY + "\n"
                + "acceleration Z: " + event.accelerationZ;
            }

        ]]>
    </fx:Script>

    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
    <s:Label id="status" text="Shake your phone a bit" top="10" width="100%"
textAlign="center"/>
    <s:TextArea id="info" width="100%" height="200" top="40" editable="false"/>
</s:Application>
```



```
Updated: 21:17:29 GMT-0400

acceleration X: 0.019999999552965164
acceleration Y: 0.6800000071525574
acceleration Z: 0.6000000238418579
```

Figure 4-1. Accelerometer Information

GPS

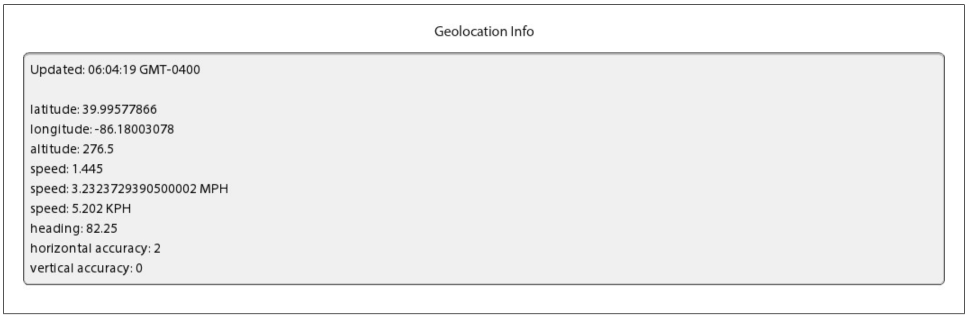
GPS stands for Global Positioning System. GPS is a space-based satellite navigation system that sends reliable location information to your handheld device.

If your application requires the use of the device's GPS capabilities, you will need to select the `read_geolocation` permission when creating your project. See Chapter 3 for help with permissions.

Let's review the code that follows. First, you'll notice that there is a private variable named `geoLocation` declared of type `flash.sensors.GeoLocation`. Within `application`

Complete of the application, an event handler function is called; it first checks to see if the device has an available GPS unit by reading the static property of the GeoLocation class. If this property returns as true, a new instance of GeoLocation is created and the data refresh interval is set to 500 milliseconds (.5 seconds) within the setRequested UpdateInterval method, and an event listener of type GeoLocationEvent.UPDATE is added to handle updates. Upon update, the GPS information is read from the event and written to a TextArea within the handleUpdate function. Note that there is also some math being done to convert the speed property into miles per hour and kilometers per hour. The results can be seen within Figure 4-2.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               applicationComplete="application1_applicationCompleteHandler(event)">
    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;

            import flash.sensors.Geolocation;

            private var geoLocation:Geolocation;

            protected function application1_applicationCompleteHandler
            (event:FlexEvent):void {
                if(Geolocation.isSupported==true){
                    geoLocation = new Geolocation();
                    geoLocation.setRequestedUpdateInterval(500);
                    geoLocation.addEventListener(GeolocationEvent.UPDATE,
                    handleLocationRequest);
                } else {
                    status.text = "Geolocation feature not supported";
                }
            }

            private function handleLocationRequest(event:GeolocationEvent):void {
                var mph:Number = event.speed*2.23693629;
                var kph:Number = event.speed*3.6;
                info.text = "Updated: " + new Date().toTimeString() + "\n\n"
                    + "latitude: " + event.latitude.toString() + "\n"
                    + "longitude: " + event.longitude.toString() + "\n"
                    + "altitude: " + event.altitude.toString()  + "\n"
                    + "speed: " + event.speed.toString()  + "\n"
                    + "speed: " + mph.toString()  + " MPH \n"
                    + "speed: " + kph.toString()  + " KPH \n"
                    + "heading: " + event.heading.toString()  + "\n"
                    + "horizontal accuracy: "
                    + event.horizontalAccuracy.toString()  + "\n"
                    + "vertical accuracy: "
                    + event.verticalAccuracy.toString();
            }

        ]]>
    </fx:Script>
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:Label id="status" text="Geolocation Info" top="10" width="100%"
textAlign="center"/>
    <s:TextArea id="info" width="100%" top="40" editable="false"/>
</s:Application>
```


Geolocation Info

```
Updated: 06:04:19 GMT-0400

latitude: 39.99577866
longitude: -86.18003078
altitude: 276.5
speed: 1.445
speed: 3.2323729390500002 MPH
speed: 5.202 KPH
heading: 82.25
horizontal accuracy: 2
vertical accuracy: 0
```

Figure 4-2. GPS Information

Camera UI

A camera is available on all BlackBerry Tablet devices.

If your application requires the use of the device's camera, you will need to select the `use_camera` permission to access the camera and the `access_shared` permission to read the new image when you're creating your project. See Chapter 3 for help with permissions. The Camera UI tools will allow your application to use the native Camera interface within the BlackBerry Tablet device.

Let's review the following code. First, you'll notice that there is a private variable named `camera` declared of type `flash.media.CameraUI`. Within `applicationComplete` of the application, an event handler function is called; it first checks to see if the device has an available Camera by reading the static property of the `CameraUI` class. If this property returns as true, a new instance of `CameraUI` is created and event listeners of type `MediaEvent.COMPLETE` and `ErrorEvent.COMPLETE` are added to handle a successfully captured image, as well as any errors that may occur.

A `Button` with an event listener on the click event is used to allow the application user to launch the `CameraUI`. When the user clicks the TAKE A PICTURE button, the `captureImage` method is called, which then opens the camera by calling the `launch` method and passing in the `MediaType.IMAGE` static property. At this point, the user is redirected from your application to the native camera. Once the user takes a picture and clicks OK, he is directed back to your application, the `MediaEvent.COMPLETE` event is triggered, and the `onComplete` method is called. Within the `onComplete` method, the `event.data` property is cast to a `flash.Media.MediaPromise` object. The `mediaPromise`

`.file.url` property is then used to populate `Label` and `Image` components that display the path to the image and the actual image to the user.

Utilizing `CameraUI` within your application is different than the raw camera access provided by Adobe AIR on the desktop. Raw camera access is also available within AIR on BlackBerry Tablet OS and works the same as the desktop version.

Figure 4-3 shows the application, Figure 4-4 shows the native camera user interface, and Figure 4-5 shows the application after a picture is taken and the user clicks OK to return to the application.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               applicationComplete="application1_applicationCompleteHandler(event)">
    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;

            private var camera:CameraUI;

            protected function application1_applicationCompleteHandler
            (event:FlexEvent):void {
                if (CameraUI.isSupported){
                    camera = new CameraUI();
                    camera.addEventListener(MediaEvent.COMPLETE, onComplete);
                    camera.addEventListener(ErrorEvent.ERROR, onError);
                    status.text="CameraUI supported";
                } else {
                    status.text="CameraUI NOT asuported";
                }
            }

            private function captureImage(event:MouseEvent):void {
                camera.launch(MediaType.IMAGE);
            }

            private function onError(event:ErrorEvent):void {
                trace("error has occurred");
            }

            private function onComplete(event:MediaEvent):void {
                var mediaPromise:MediaPromise = event.data;
                status.text = mediaPromise.file.url;
                image.source = mediaPromise.file.url;
            }

        ]]>
    </fx:Script>
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:Label id="status" text="Click Take a Picture button" top="10" width="100%"
textAlign="center"/>

    <s:Button width="300" height="60" label="TAKE A PICTURE"
click="captureImage(event)"
              horizontalCenter="0" enabled="{CameraUI.isSupported}"
              top="80"/>

    <s:Image id="image" width="230" height="350" horizontalCenter="0" top="170"/>
</s:Application>
```

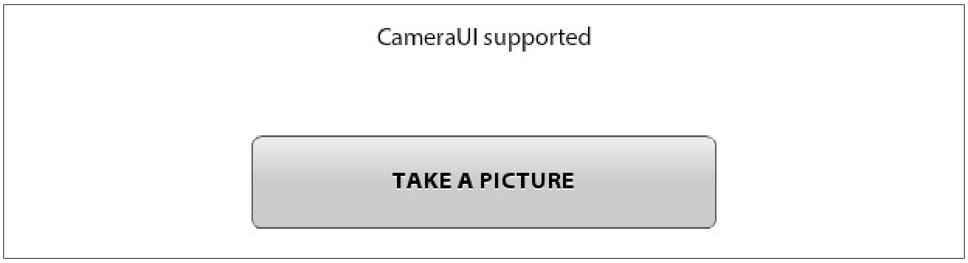


Figure 4-3. Camera UI Application

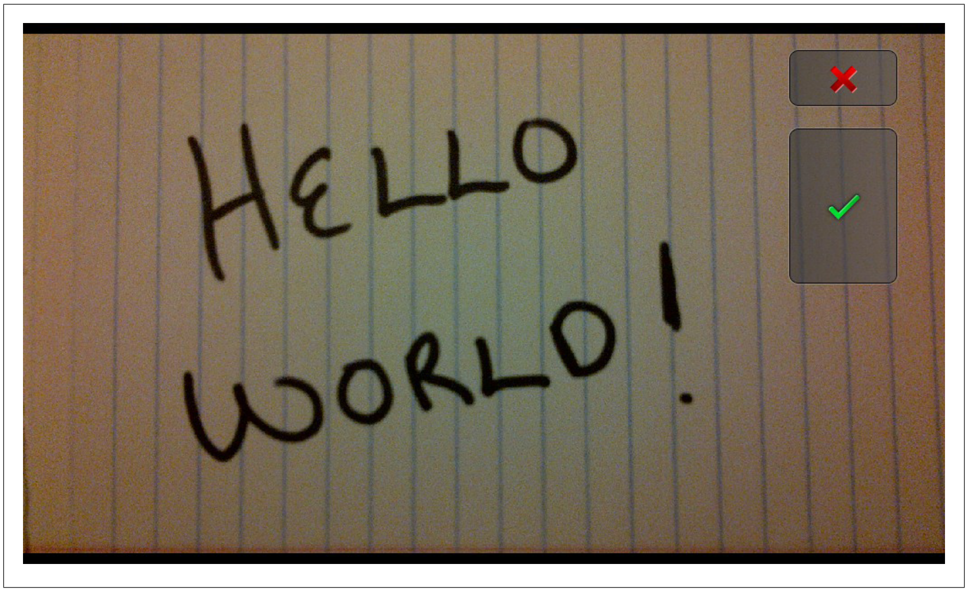


Figure 4-4. Native Camera UI

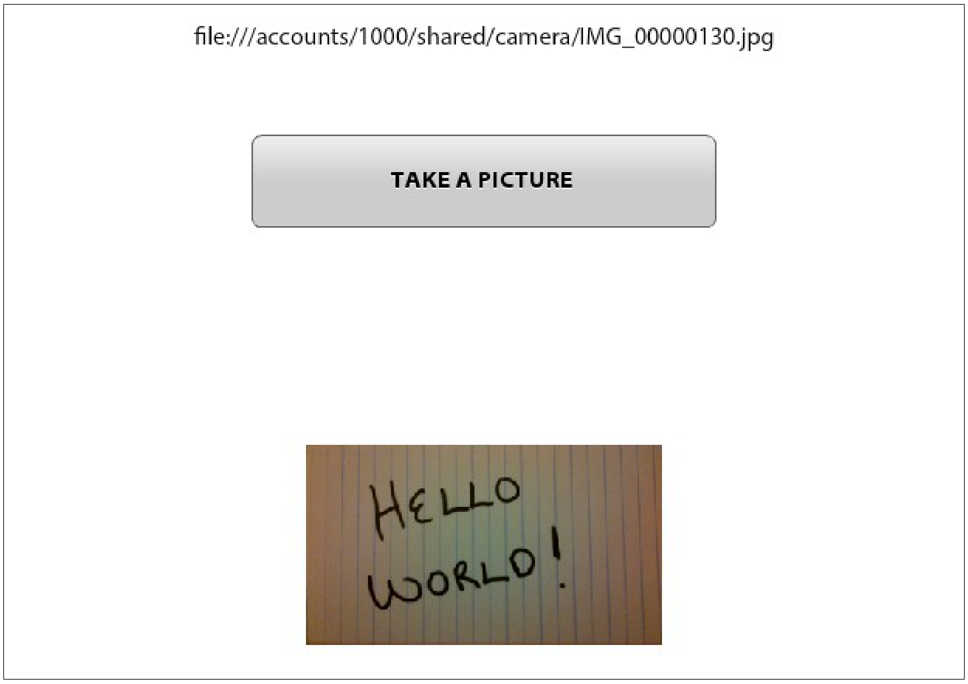

file:///accounts/1000/shared/camera/IMG_00000130.jpg

Figure 4-5. Application after taking picture

Camera Roll

The Camera Roll is the camera's gallery of images.

If your application requires the use of the device's camera roll, you will need to select the `access_shared` permission when you're creating your project. See Chapter 3 for help with permissions.

Let's review the code that follows. First, you will notice there is a private variable named `cameraRoll` declared of type `flash.media.CameraRoll`. Within `applicationComplete` of the application, an event handler function is called; it first checks to see if the device supports access to the image gallery by reading the static property of the `CameraRoll` class. If this property returns as true, a new instance of `CameraRoll` is created, and event listeners of type `MediaEvent.COMPLETE` and `ErrorEvent.COMPLETE` are added to handle a successfully captured image, as well as any errors that may occur.

A button with an event listener on the click event is used to allow the user to browse the image gallery. When the user clicks the BROWSE GALLERY button, the `browseGallery` method is called and then opens the device's image gallery. At this point, the user is redirected from your application to the native gallery application. Once the user selects an image from the gallery, she is directed back to your application, the `Media`

Event.COMPLETE event is triggered, and the mediaSelected method is called. Within the mediaSelected method, the event.data property is cast to a flash.Media.MediaPromise object. The mediaPromise.file.url property is then used to populate Label and Image components that display the path to the image and the actual image to the user. Figure 4-6 shows the application and Figure 4-7 shows the application after a picture is selected from the gallery and the user returns to the application.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               applicationComplete="application1_applicationCompleteHandler(event)">
    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;

            private var cameraRoll:CameraRoll;

            protected function application1_applicationCompleteHandler
            (event:FlexEvent):void {
                if(CameraRoll.supportsBrowseForImage){
                    cameraRoll = new CameraRoll();
                    cameraRoll.addEventListener(MediaEvent.SELECT, mediaSelected);
                    cameraRoll.addEventListener(ErrorEvent.ERROR, onError);
                } else{
                    status.text="CameraRoll NOT suported";
                }
            }

            private function browseGallery(event:MouseEvent):void {
                cameraRoll.browseForImage();
            }

            private function onError(event:ErrorEvent):void {
                trace("error has occurred");
            }

            private function mediaSelected(event:MediaEvent):void{
                var mediaPromise:MediaPromise = event.data;
                status.text = mediaPromise.file.url;
                image.source = mediaPromise.file.url;
            }
        ]]>
    </fx:Script>
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:Label id="status" text="Click Browse Gallery to select image" top="10"
width="100%" textAlign="center"/>

    <s:Button width="300" height="60" label="BROWSE GALLERY"
click="browseGallery(event)"
              enabled="{CameraRoll.supportsBrowseForImage}"
              top="80" horizontalCenter="0"/>
    <s:Image id="image" width="230" height="350" top="170" horizontalCenter="0"/>
</s:Application>
```

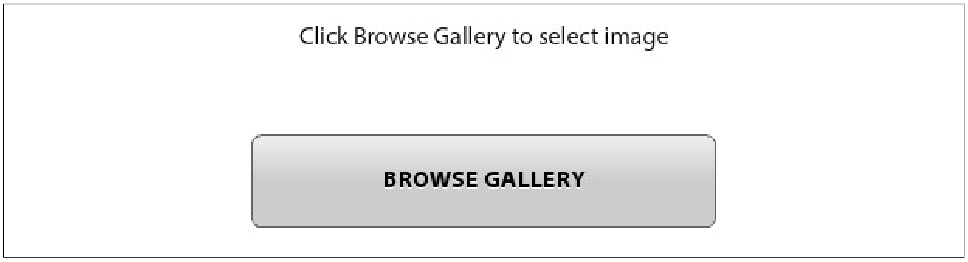

Click Browse Gallery to select image

Figure 4-6. Browse Gallery Application

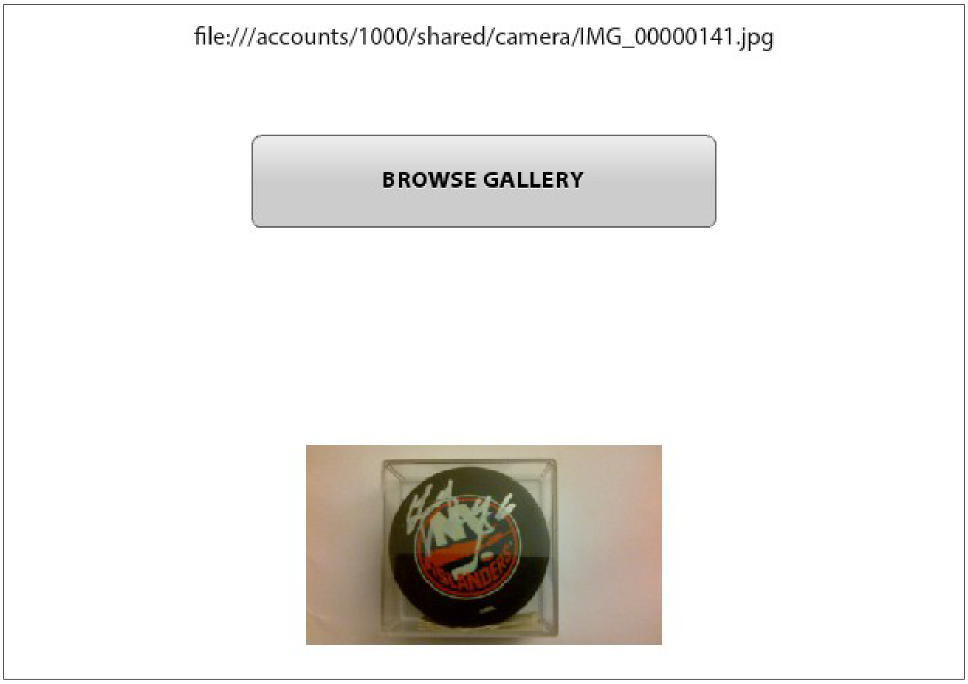

file:///accounts/1000/shared/camera/IMG_00000141.jpg

Figure 4-7. Browse Gallery Application with Picture Selected

Microphone

If your application requires the use of the device's microphone, you'll need to select the `record_audio` and `play_audio` permissions when creating your project. See Chapter 3 for help with permissions.

Let's review the following code. First, you'll notice that there is a private variable named `microphone` declared of type `flash.media.Microphone`. Within `applicationComplete` of the application, an event handler function is called; it first checks to see if the device supports access to the microphone by reading the `static` property of the `Microphone` class. If this property returns as true, an instance of the `Microphone` is retrieved and set to the microphone variable, the `rate` is set to 44, and the `setUseEchoSuppression` method is used to set the echo suppression to true. Variables of type `ByteArray` and `Sound` are also declared within this application. Instances of these variables will be created during use of this application.

There are three `button` components within the application to trigger the record, stop, and playback functionalities.

Clicking the Record button will call the `record_clickHandler` function, which will create a new instance of the recording variable of type `ByteArray`. An event listener of type `SampleDataEvent.SAMPLE_DATA` is added to the microphone, which will call the `micDataHandler` method when it receives data. Within the `micDataHandler` method, the data is written to the recording `ByteArray`.

Clicking the Stop button will stop the recording by removing the `SampleDataEvent.SAMPLE_DATA` event listener.

Clicking the Play button will call the `play_clickHandler` method, which first sets the position of the recording `ByteArray` to `0` so it is ready for playback. It then creates a new instance of the `Sound` class and sets it to the sound variable. It also adds an event listener of type `SampleDataEvent.SAMPLE_DATA` that will call the `playSound` method when it receives data. Finally, the play method is called on the sound variable to start the playback.

The `playSound` method loops through the recording `ByteArray` in memory and writes those bytes back to the data property of the `SampleDataEvent`, which then plays through the device's speaker.

To extend this sample, you need to use some open source classes to convert the recording `ByteArray` to an mp3 or wav file so that it can be saved to disk. Figure 4-8 shows the application.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               applicationComplete="application1_applicationCompleteHandler(event)">
    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;

            private var microphone:Microphone;
            private var recording:ByteArray;
            private var sound:Sound;

            protected function application1_applicationCompleteHandler
            (event:FlexEvent):void
            {
                if(Microphone.isSupported){
                    microphone = Microphone.getMicrophone();
                    microphone.rate = 44;
                    microphone.setUseEchoSuppression(true);
                } else {
                    status.text="Microphone NOT suported";
                }
            }

            private function micDataHandler(event:SampleDataEvent):void{
                recording.writeBytes(event.data);
            }

            protected function record_clickHandler(event:MouseEvent):void
            {
                recording = new ByteArray();
                microphone.addEventListener(SampleDataEvent.SAMPLE_DATA,
                micDataHandler);
            }

            protected function stop_clickHandler(event:MouseEvent):void
            {
                microphone.removeEventListener(SampleDataEvent.SAMPLE_DATA,
                micDataHandler);
            }

            protected function play_clickHandler(event:MouseEvent):void
            {
                recording.position = 0;
                sound = new Sound();
                sound.addEventListener(SampleDataEvent.SAMPLE_DATA, playSound);
                sound.play();
            }

            private function playSound(event:SampleDataEvent):void
            {
                for (var i:int = 0; i < 8192 && recording.bytesAvailable > 0; i++){
                    var sample:Number = recording.readFloat();
                    event.data.writeFloat(sample);
                    event.data.writeFloat(sample);
                }
            }

        ]]>
    </fx:Script>
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:Label id="status"
text="Click Record to grab some audio, then Stop and Play it back"
             top="10" width="100%" textAlign="center"/>
    <s:HGroup top="80" horizontalCenter="0">
```

```
        <s:Button id="record" label="Record" click="record_clickHandler(event)" />
        <s:Button id="stop" label="Stop" click="stop_clickHandler(event)" />
        <s:Button id="play" label="Play" click="play_clickHandler(event)" />
    </s:HGroup>
</s:Application>
```

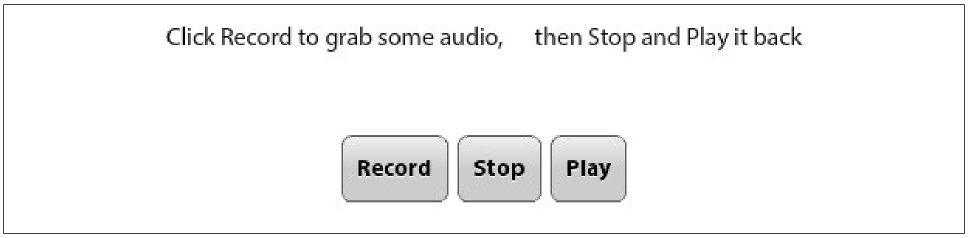

Click Record to grab some audio,   then Stop and Play it back
Record
Stop
Play

Figure 4-8. Microphone application

Multi-Touch

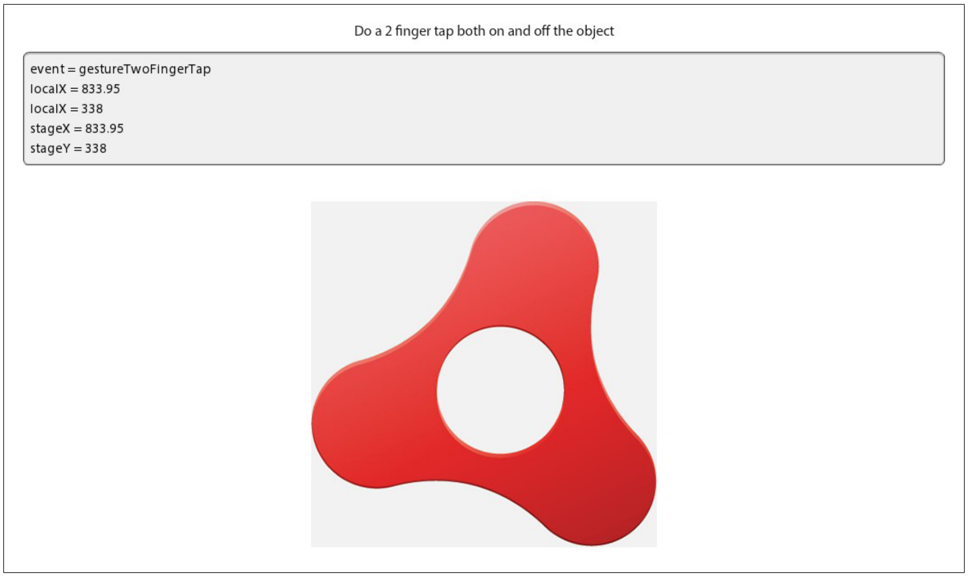
One of the navigation methods unique to mobile devices is the ability to interact with an application via gestures on the device's touch screen. Multi-touch is defined as the ability to simultaneously register three or more touch points on the device. Within Adobe AIR 2.6, there are two event classes used to listen for multi-touch events.

GestureEvent

The `GestureEvent` class is used to listen for a two-finger tap on the device. `GESTURE_TWO_FINGER_TAP` is the event used to listen for this action. This event will return the registration points for the x and y coordinates when a two-finger tap occurs for both stage positioning as well as object positioning.

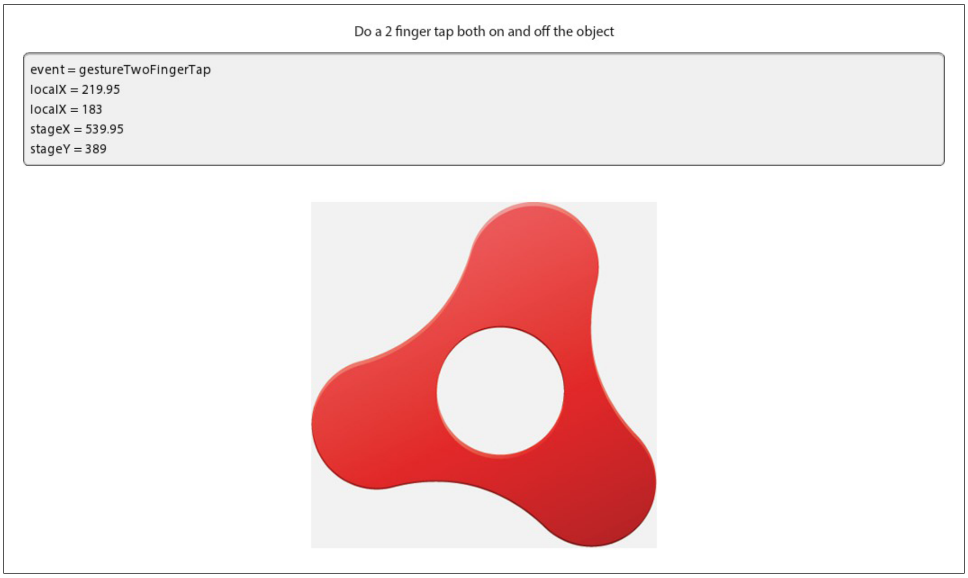
Let's review the code that follows. Within `applicationComplete` of the application, an event handler function is called; it first sets the `Multitouch.inputMode` to `MultitouchInputMode.GESTURE`. Next, it checks to see if the device supports multi-touch by reading the static property of the `Multitouch` class. If this property returns as true, an event listener is added to the stage to listen for `GestureEvent.GESTURE_TWO_FINGER_TAP` events. When this event occurs, the `onGestureTwoFingerTap` method is called. The `onGestureTwoFingerTap` method will capture the *localX* and *localY* coordinates, as well as the *stageX* and *stageY* coordinates. If you two-finger tap on an empty portion of the stage, these values will be identical. If you two-finger tap on an object on the stage, the *localX* and *localY* coordinates will be the values within the object, and the *stageX* and *stageY* will be relative to the stage itself. See Figure 4-9 for an example of a two-finger tap on the stage and Figure 4-10 for a two-finger tap on the Android image.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               applicationComplete="application1_applicationCompleteHandler(event)">
    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;

            protected function application1_applicationCompleteHandler
            (event:FlexEvent):void {
                Multitouch.inputMode = MultitouchInputMode.GESTURE;
                if(Multitouch.supportsGestureEvents){
                    stage.addEventListener(GestureEvent.GESTURE_TWO_FINGER_TAP,
                    onGestureTwoFingerTap);
                } else {
                    status.text="gestures not supported";
                }

            }
            private function onGestureTwoFingerTap(event:GestureEvent):void {
                info.text = "event = " + event.type + "\n" +
                    "localX = " + event.localX + "\n" +
                    "localX = " + event.localY + "\n" +
                    "stageX = " + event.stageX + "\n" +
                    "stageY = " + event.stageY;
            }

        ]]>
    </fx:Script>
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
    <s:Label id="status" text="Do a 2 finger tap both on and off the object"
             top="10" width="100%" textAlign="center"/>
    <s:TextArea id="info" width="100%" top="40" editable="false"/>
    <s:Image width="384" height="384" bottom="10" horizontalCenter="0"
             source="@Embed('android_icon.png')"/>
</s:Application>
```


Do a 2 finger tap both on and off the object

```
event = gestureTwoFingerTap
localX = 833.95
localX = 338
stageX = 833.95
stageY = 338
```

Figure 4-9. Two-finger tap on stage


Do a 2 finger tap both on and off the object

```
event = gestureTwoFingerTap
localX = 219.95
localX = 183
stageX = 539.95
stageY = 389
```

Figure 4-10. Two-finger tap on image object

TransformGesture

There are multiple transform gesture events available within AIR 2.6. Each will capture a unique multi-touch event. The next example demonstrates how to listen for `GESTURE_PAN`, `GESTURE_ROTATE`, `GESTURE_SWIPE`, and `GESTURE_ZOOM` events.

Let's review the following code. Within `applicationComplete` of the application, an event handler function is called; it first sets the `Multitouch.inputMode` to `MultitouchInputMode.GESTURE`. Next, it checks to see if the device supports multi-touch by reading the static property of the `Multitouch` class. If this property returns as true, event listeners are added to the stage to listen for the `TransformGestureEvent.GESTURE_PAN`, `TransformGestureEvent.GESTURE_ROTATE`, `TransformGestureEvent.GESTURE_SWIPE`, and `TransformGestureEvent.GESTURE_ZOOM` events.

When a user grabs the object with two fingers and drags the object, the `TransformGestureEvent.GESTURE_PAN` event is triggered and the `onGesturePan` method is called. Within the `onGesturePan` method, the *offsetX* and *offsetY* values of the event are written to the text property of the `TextArea` component. Adding the *offsetX* and *offsetY* values returned from the event to the object's `x` and `y` properties will move the object across the stage. The results are shown in Figure 4-11.

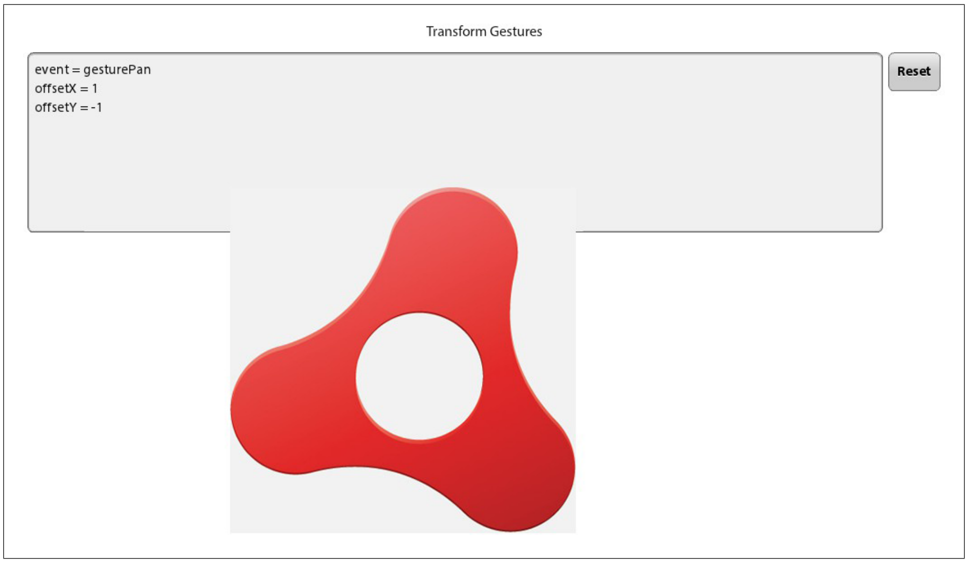
When a user grabs the object with two fingers and rotates the object, the `TransformGestureEvent.GESTURE_ROTATE` event is triggered and the `onGestureRotate` method is called. Within the `onGestureRotate` method, the rotation value of this event is written to the text property of the `TextArea` component. To allow the object to rotate around its center, the object's `transformAround` method is called, and the event's rotation value is added to the object's `rotationZ` value. The results are shown in Figure 4-12.

When a user swipes across the object with one finger in any direction, the `TransformGestureEvent.GESTURE_SWIPE` event is triggered and the `onGestureSwipe` method is called. Within the `onGestureSwipe` method, the value of the event's *offsetX* and *offsetY* is evaluated to determine which direction the user swiped across the object. This direction is then written to the text property of the `TextArea` component. The results are pictured in Figure 4-13.

When a user performs a "pinch and zoom" on the object with two fingers, the `TransformGestureEvent.GESTURE_ZOOM` event is triggered and the `onGestureZoom` method is called. Within the `onGestureZoom` method, the value of the event's *scaleX* and *scaleY* is written to the text property of the `TextArea` component. The *scaleX* value is then used as a multiplier on the object's *scaleX* and *scaleY* properties to increase or decrease the size of the object as the user pinches or expands two fingers on the object. The results are shown in Figure 4-14.

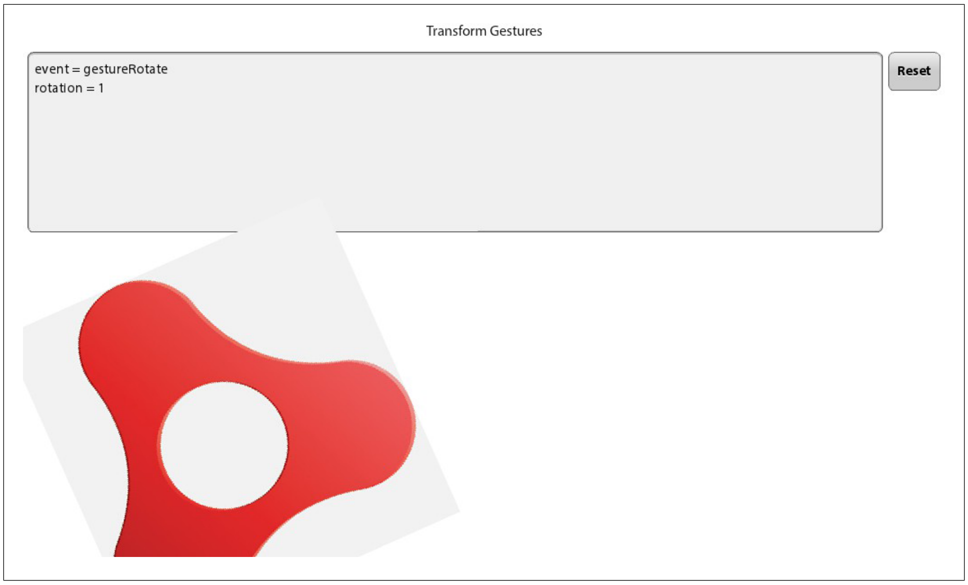
```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               applicationComplete="application1_applicationCompleteHandler(event)">
    <fx:Script>
```

```
        <![CDATA[
            import mx.events.FlexEvent;

            protected function application1_applicationCompleteHandler
            (event:FlexEvent):void {
                Multitouch.inputMode = MultitouchInputMode.GESTURE;
                if(Multitouch.supportsGestureEvents){
                    image.addEventListener(TransformGestureEvent.GESTURE_PAN,
onGesturePan);
                    image.addEventListener(TransformGestureEvent.GESTURE_ROTATE,
onGestureRotate);
                    image.addEventListener(TransformGestureEvent.GESTURE_SWIPE,
onGestureSwipe);
                    image.addEventListener(TransformGestureEvent.GESTURE_ZOOM,
onGestureZoom);
                } else {
                    status.text="gestures not supported";
                }
            }

            private function onGesturePan(event:TransformGestureEvent):void{
                info.text = "event = " + event.type + "\n" +
                "offsetX = " + event.offsetX + "\n" +
                "offsetY = " + event.offsetY;
                image.x += event.offsetX;
                image.y += event.offsetY;
            }

            private function onGestureRotate( event : TransformGestureEvent ) : void {
                info.text = "event = " + event.type + "\n" +
                "rotation = " + event.rotation;
                image.transformAround(new Vector3D(image.width/2,image.height/2, 0),
                                      null,
                                      new Vector3D(0,0,image.rotationZ + event.rotation));
            }

            private function onGestureSwipe( event : TransformGestureEvent ) : void {
                var direction:String = "";
                if(event.offsetX == 1) direction = "right";
                if(event.offsetX == -1) direction = "left";
                if(event.offsetY == 1) direction = "down";
                if(event.offsetY == -1) direction = "up";
                info.text = "event = " + event.type + "\n" +
                "direction = " + direction;
            }

            private function onGestureZoom( event : TransformGestureEvent ) : void {
                info.text = "event = " + event.type + "\n" +
                "scaleX = " + event.scaleX + "\n" +
                "scaleY = " + event.scaleY;
                image.scaleX = image.scaleY *= event.scaleX;
            }

            protected function button1_clickHandler(event:MouseEvent):void
            {
                image.rotation = 0;
                image.scaleX = 1;
                image.scaleY = 1;
                image.x = 40;
                image.y = 260;
                info.text = "";
            }

        ]]>
    </fx:Script>
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
    <s:Label id="status" text="Transform Gestures" top="10" width="100%"
textAlign="center"/>
    <s:HGroup width="100%" top="40" left="5" right="5">
        <s:TextArea id="info" editable="false" width="100%" height="200"/>
        <s:Button label="Reset" click="button1_clickHandler(event)"/>
    </s:HGroup>
    <s:Image id="image" x="40" y="260" width="400" height="400"
             source="@Embed('android_icon.png')"/>
</s:Application>
```



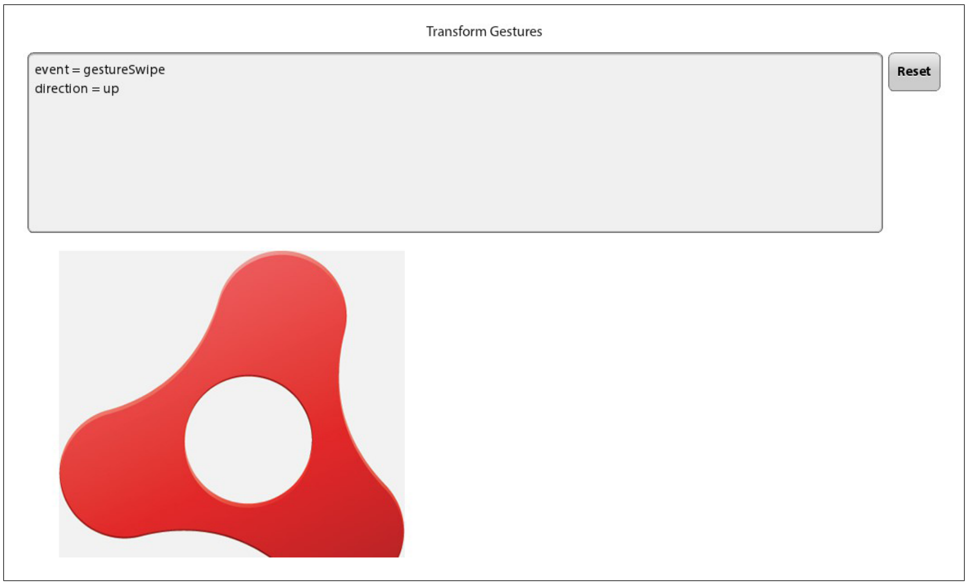
```
event = gesturePan
offsetX = 1
offsetY = -1
```

Figure 4-11. GESTURE_PAN event



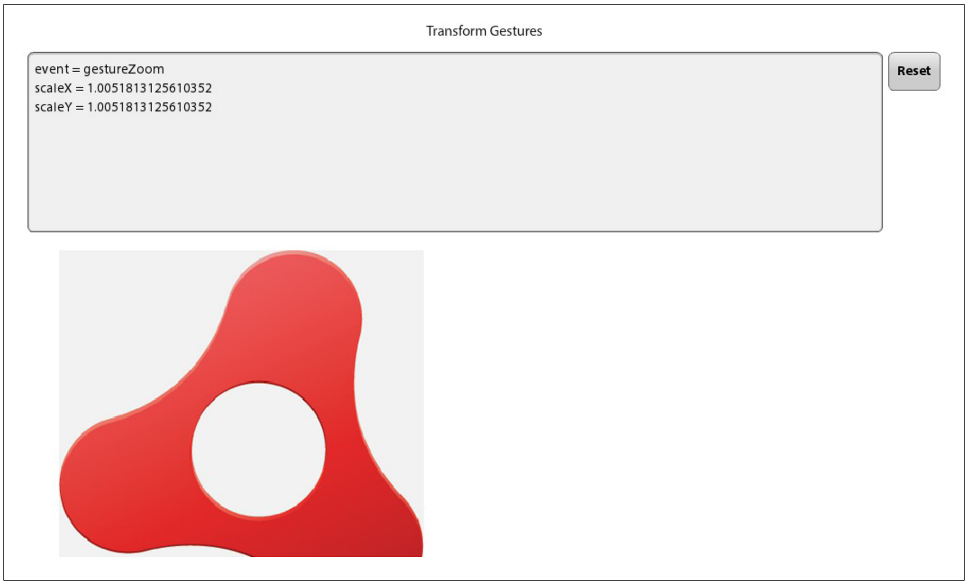
```
event = gestureRotate
rotation = 1
```

Figure 4-12. GESTURE_ROTATE event



```
event = gestureSwipe
direction = up
```

Figure 4-13. GESTURE_SWIPE event



```
event = gestureZoom
scaleX = 1.0051813125610352
scaleY = 1.0051813125610352
```

Figure 4-14. GESTURE_ZOOM event

Busy Indicator

A new component has been added to provide feedback to users within your mobile application. There is no cursor to show busy status like in desktop development, so the `BusyIndicator` component was added specifically for this reason. Using this component is simple.

Let's review the code that follows. There is a `CheckBox` with the label "Show Busy Indicator" that, when checked, calls the `checkbox1_clickHandler` method. There is a `BusyIndicator` component with an ID of indicator and the visible property set to false. Within the `checkbox1_clickHandler` method, the indicator's visible property is set to the value of the `CheckBox`. This simply shows or hides the `BusyIndicator`. Within the `BusyIndicator`, you can set the `height`, `width`, and `symbolColor` to suit the needs and style of your application. The results are shown in Figure 4-15.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark">

    <fx:Script>
        <![CDATA[
            protected function checkbox1_clickHandler(event:MouseEvent):void
            {
                indicator.visible = event.target.selected;
            }
        ]]>
    </fx:Script>

    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:CheckBox label="Show Busy Indicator"
                horizontalCenter="0"
                click="checkbox1_clickHandler(event)" top="10"/>
    <s:BusyIndicator id="indicator" height="300" width="300"
                     verticalCenter="0"
                     horizontalCenter="0"
                     visible="false"
                     symbolColor="black"/>

</s:Application>
```

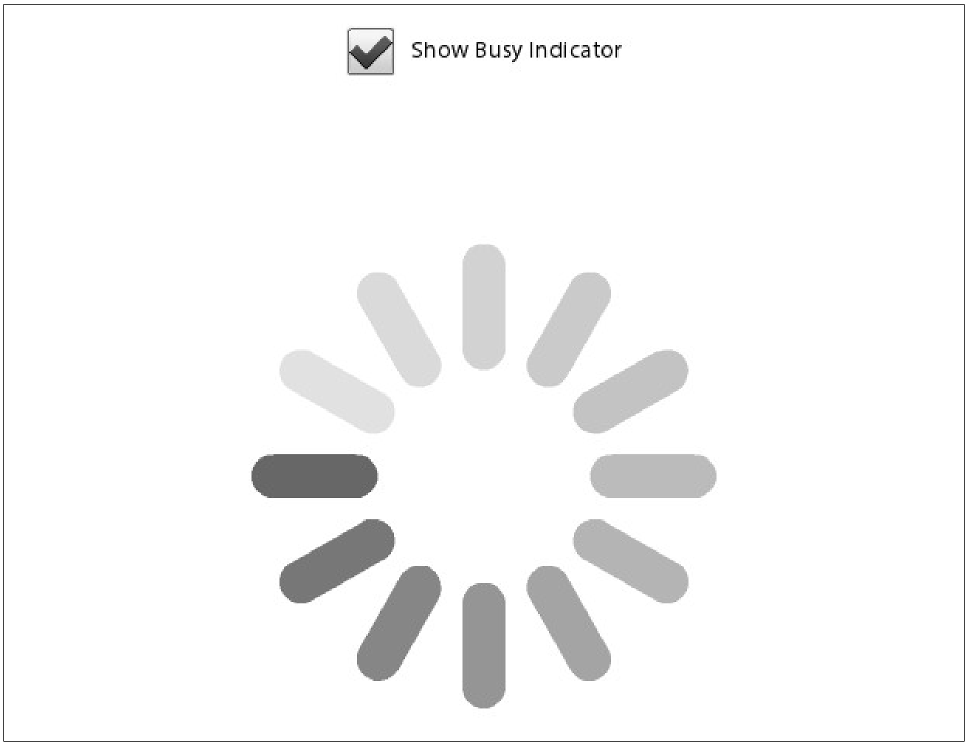


Figure 4-15. Busy Indicator component

CHAPTER 5

Working with the File System

AIR on BlackBerry Tablet OS provides access to the file system to read, write, and update files of all types. This functionality can be very useful, not only for reading existing files, but also for storing files, media, data, etc. This chapter will demonstrate how to read and write text files, browse the file system for media files, and create and write to an SQLite database.

This chapter has sample applications that require access to the file system—you will need to select the WRITE_EXTERNAL_STORAGE permission when creating these projects. See Chapter 3 for help with permissions.

File System Access

Just as in the desktop version of Adobe AIR, AIR on BlackBerry Tablet OS provides you access to the file system. The usage is exactly the same.

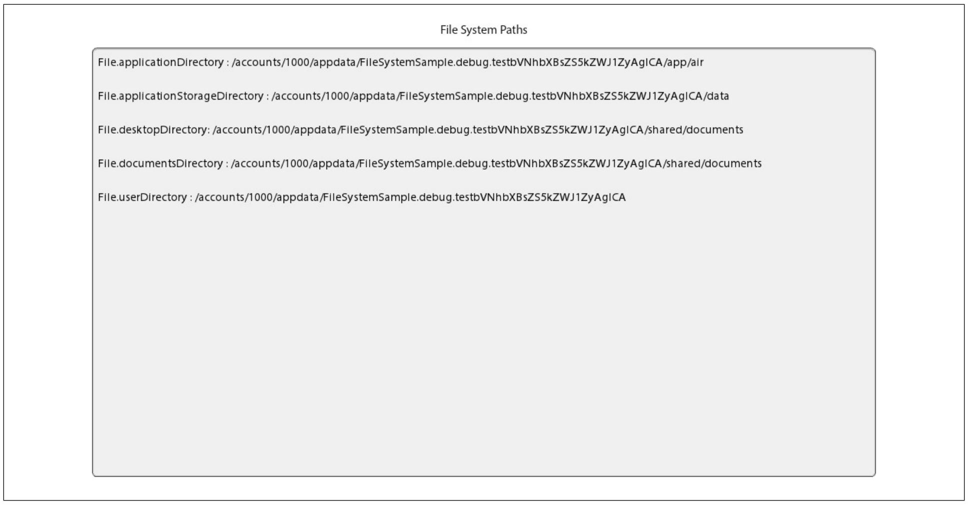
Folder Aliases

To access the file system, you can navigate using several folder static alias properties of the File class. Since you are attempting to read these shared files, you will need to select the access_shared permission when creating your project.

Let's take a look at the following code. Within applicationComplete of the application, the application1_applicationCompleteHandler method is called and the static File properties are read and written to a String variable. This String variable is written to the text property of a TextArea component. Figure 5-1 shows the results. You will notice that many of the aliases return the same value, which is a path to the device's external storage card.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               applicationComplete="application1_applicationCompleteHandler(event)">
    <fx:Script>
```

```
        <![CDATA[
            import mx.events.FlexEvent;

            protected function application1_applicationCompleteHandler
            (event:FlexEvent):void
            {
                var s:String = "";
                s += "File.applicationDirectory : " +
                File.applicationDirectory.nativePath + "\n\n";
                s += "File.applicationStorageDirectory : " +
                File.applicationStorageDirectory.nativePath + "\n\n";
                s += "File.desktopDirectory: " +
                File.desktopDirectory.nativePath + "\n\n";
                s += "File.documentsDirectory : " +
                File.documentsDirectory.nativePath + "\n\n";
                s += "File.userDirectory : " +
                File.userDirectory.nativePath + "\n\n";
                info.text = s;
            }

        ]]>
    </fx:Script>
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:Label text="File System Paths" top="10" width="100%" textAlign="center"/>

    <s:TextArea id="info" width="100%" height="100%" top="40" editable="false"/>

</s:Application>
```



```
File.applicationDirectory : /accounts/1000/appdata/FileSystemSample.debug.testbVNhbXBsZS5kZWJ1ZyAgICA/app/air

File.applicationStorageDirectory : /accounts/1000/appdata/FileSystemSample.debug.testbVNhbXBsZS5kZWJ1ZyAgICA/data

File.desktopDirectory: /accounts/1000/appdata/FileSystemSample.debug.testbVNhbXBsZS5kZWJ1ZyAgICA/shared/documents

File.documentsDirectory : /accounts/1000/appdata/FileSystemSample.debug.testbVNhbXBsZS5kZWJ1ZyAgICA/shared/documents

File.userDirectory : /accounts/1000/appdata/FileSystemSample.debug.testbVNhbXBsZS5kZWJ1ZyAgICA
```

Figure 5-1. File System Paths

Read and Write to the File System

Adobe AIR gives you the ability to read and write files to the file system. The following example will create a new file and then read it back.

Let's review the following code. There are two `TextArea` and two `Button` components that make up this sample. The first `TextArea` with the ID of "contents" will hold the contents of what is to be written to the file; and the second, with the ID of "results," will output the file contents when read back. The application is shown in Figure 5-2.

Clicking on the `Button` component with the label of "Save" will call the `button1_clickHandler` method. Within the `button1_clickHandler` method, an instance of `File` is created with the name "file," the path is resolved to the `userDirectory`, and "samples/test.txt" is passed in to the `resolvePath` method. An instance of `FileStream` with the name "stream" is created to write the data to the file. The `open` method is called on the stream object and the file and `FileMode.WRITE` is passed in, which will open the file with write permissions. Next, the `writeUTFBytes` method is called and the `contents.text` is passed in. Finally, the stream is closed.

Clicking on the `Button` with the label of "Load" will call the `button2_clickHandler` method. Within the `button2_clickHandler` method, an instance of `File` is created with the name "file," the path is resolved to the `userDirectory`, and "samples/test.txt" is passed in to the `resolvePath` method. An instance of `FileStream` with the name "stream" is created to read the data from the file. The `open` method is called on the stream object and the file and `FileMode.READ` is passed in, which will open the file with write permissions. Next, the `readUTFBytes` method is called, the `stream.bytesAvailable` is passed in, and the results are set to the `results.text` property of the second `TextArea`. Finally, the stream is closed. Figure 5-3 shows the contents of the file and the path to the newly created file within the `TextArea` with the ID of `results`.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark">

    <fx:Script>
        <![CDATA[

            protected function button1_clickHandler(event:MouseEvent):void
            {
                var file:File = File.applicationStorageDirectory.
resolvePath("samples/test.txt");
                var stream:FileStream = new FileStream()
                stream.open(file, FileMode.WRITE);
                stream.writeUTFBytes(contents.text);
                stream.close();
            }

            protected function button2_clickHandler(event:MouseEvent):void
            {
                var file:File = File. applicationStorageDirectory.
resolvePath("samples/test.txt");
                var stream:FileStream = new FileStream()
                stream.open(file, FileMode.READ);
                results.text = stream.readUTFBytes(stream.bytesAvailable);
                stream.close();
            }

        ]]>
    </fx:Script>

    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:TextArea id="contents" left="10" right="10" top="10" height="100"/>
    <s:Button right="10" top="120" label="Save" click="button1_clickHandler(event)"/>
    <s:Label id="path" left="10" top="160"/>
    <s:Button left="10" top="200" label="Load" click="button2_clickHandler(event)"/>
    <s:TextArea id="results" left="10" right="10" top="280" height="100"
editable="false"/>
</s:Application>
```

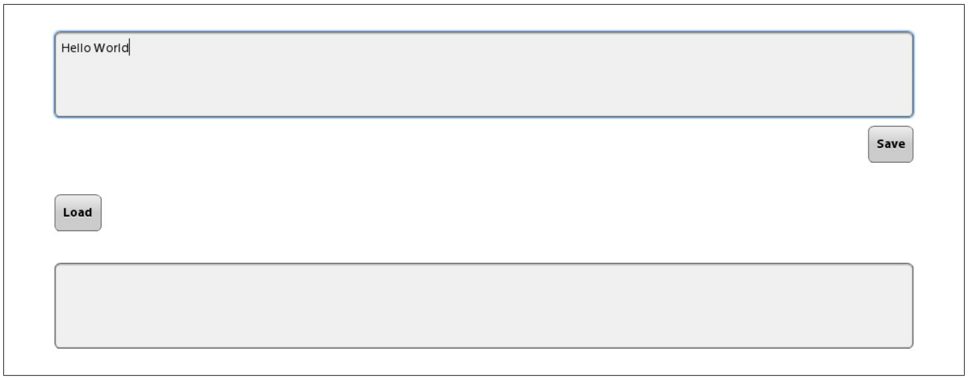
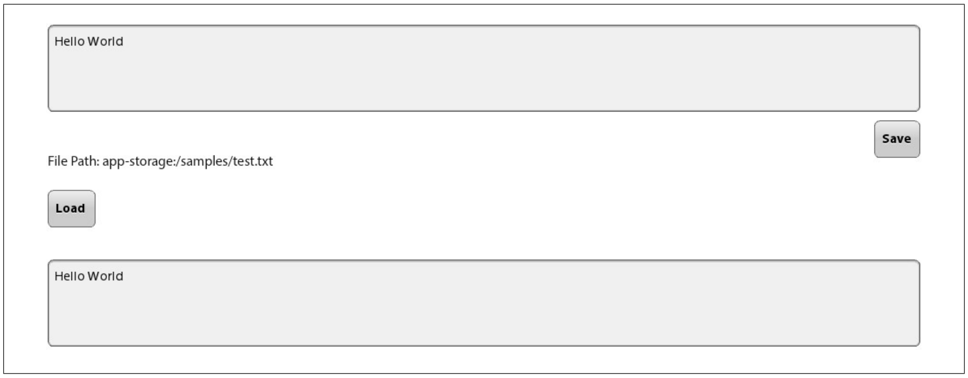


Figure 5-2. File Save Application


Hello World

File Path: app-storage:/samples/test.txt

Hello World

Figure 5-3. File contents loaded into results TextArea and file path displayed

File Browse for Single File

The browse for file functionality of the File class works a bit differently in BlackBerry Tablet OS as compared to the desktop version. Within BlackBerry Tablet OS, the browseForOpen method will open up a specific native file selector that will allow you to open a file of type Audio, Image, Documents, or Video. Since you are attempting to read these shared files, you will need to select the access_shared permission when creating your project. See Chapter 3 for help with permissions.

Let's review the next code sample. The button with the "Browse" label will call the button1_clickHandler when clicked. Within this function, an instance of File is created with the variable name "file." An event listener listening for the Event.SELECT event is added to the File object, then the browseForOpen method is called. The application is shown in Figure 5-4. When browseForOpen is called, the BlackBerry Tablet OS file selector is launched (see Figure 5-5). After selecting a file within the BlackBerry Tablet OS file selector, the event is fired and the onFileSelect method is called. The event .currentTarget is cast to a File object and its nativePath, extension, and url properties are used to display the nativePath and the image in the example shown in Figure 5-6.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark">

    <fx:Script>
        <![CDATA[

            protected function button1_clickHandler(event:MouseEvent):void
            {
                var file:File = new File();
                file.addEventListener(Event.SELECT, onFileSelect);
                file.browseForOpen("Open");
            }

            private function onFileSelect(event:Event):void {
                var file:File = File(event.currentTarget);
                filepath.text = file.nativePath;
                if(file.extension == "jpg"){
                    image.source = file.url;
                }
            }
        ]]>
    </fx:Script>

    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:Button horizontalCenter="0" top="10" label="Browse"
click="button1_clickHandler(event)"/>
    <s:Label id="filepath" left="10" right="10" top="100"/>
    <s:Image id="image" width="230" height="350" top="150" horizontalCenter="0"/>
</s:Application>
```

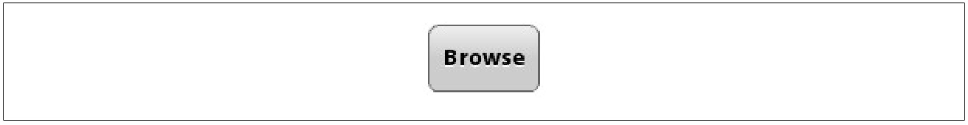


Figure 5-4. Browse for file application

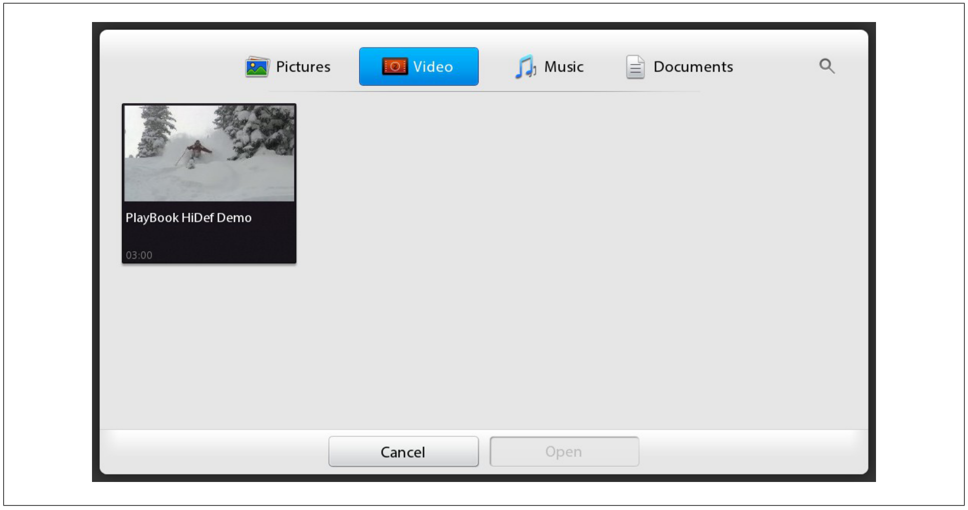
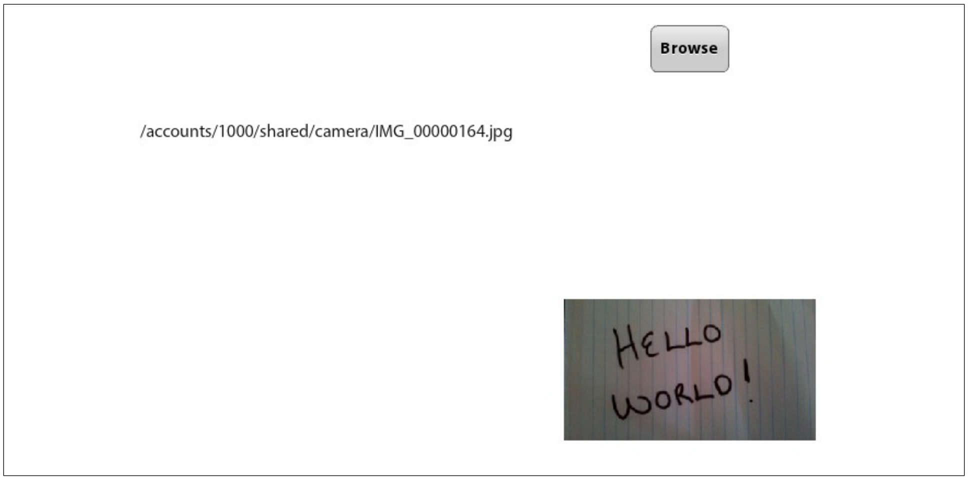


Figure 5-5. File selector



/accounts/1000/shared/camera/IMG_00000164.jpg

Figure 5-6. Browse for file with an Image selected

File Browse for Multiple Files

The browse for file functionality of the `File` class works a bit differently in BlackBerry Tablet OS compared to the desktop version. Within BlackBerry Tablet OS, the `browseForOpenMultiple` method will open up a specific native file selector that will allow you to open multiple files of type Audio, Image, Documents, or Video. Since you are attempting to read these shared files, you will need to select the `access_shared` permission when creating your project. See Chapter 3 for help with permissions.

Let's review the following code. The button with the "Browse" label will call the `button1_clickHandler` when clicked. Within this function, an instance of `File` is created with the variable name "file." An event listener listening for the `Event.SELECT_MULTIPLE` event is added to the `File` object, then the `browseForOpen` method is called. When `browseForOpen` is called, the BlackBerry Tablet OS file selector is launched (see Figure 5-7). After selecting the files within the BlackBerry Tablet OS file selector, the event is fired and the `onMultipleFileSelect` method is called. Within this method, the array of files included in the event is looped over, and if the file type is an image, it is added as a new element. The results are shown in Figure 5-8.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark">

    <fx:Script>
        <![CDATA[
            import spark.components.Image;

            protected function button1_clickHandler(event:MouseEvent):void
            {
                var file:File = new File();
                file.addEventListener(FileListEvent.SELECT_MULTIPLE,
onMultipleFileSelect);
                file.browseForOpenMultiple("Open");
            }

            private function onMultipleFileSelect(event:FileListEvent):void {
                holder.removeAllElements();
                for (var i:int=0; i<event.files.length; i++){
                    var f:File = event.files[i] as File;
                    if(f.extension == "jpg"){
                        var image:Image = new Image();
                        image.source = f.url;
                        image.scaleX = .1;
                        image.scaleY = .1;
                        holder.addElement(image);
                    }
                }
            }
        ]]>
    </fx:Script>

    <fx:Declarations>
```

```
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:Button horizontalCenter="0" top="10" label="Browse"
click="button1_clickHandler(event)"/>
    <s:Label id="filepath" left="10" right="10" top="100"/>
    <s:Scroller top="150" horizontalCenter="0" bottom="0">
        <s:VGroup id="holder"/>
    </s:Scroller>

</s:Application>
```

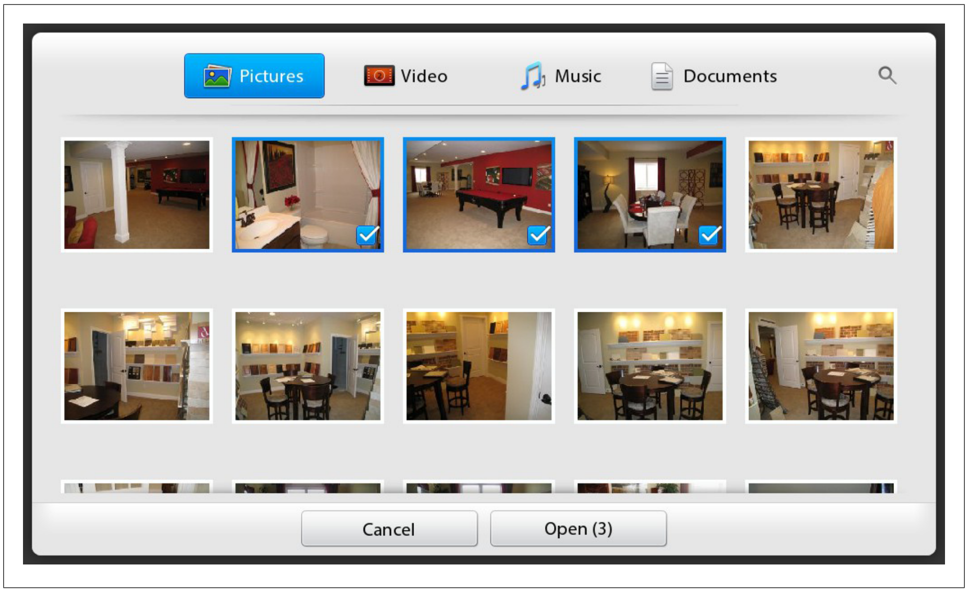


Figure 5-7. Browse for multiple files

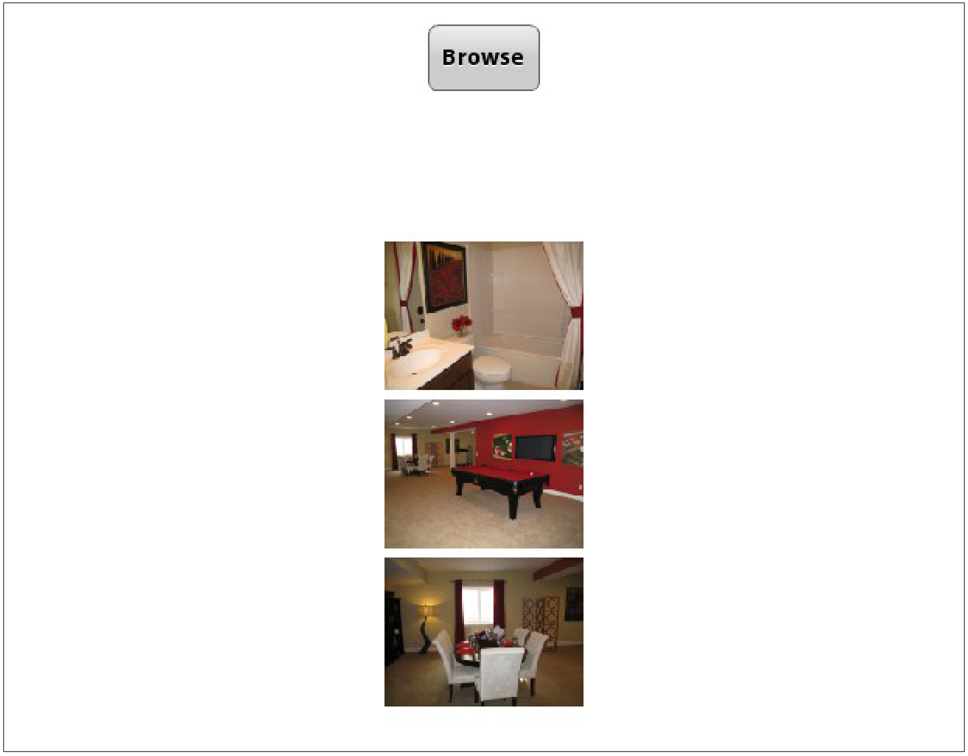


Figure 5-8. Multiple images selected

SQLite Databases

Just as within Adobe AIR on the desktop, you can utilize an SQLite database for storing data on a mobile device. The next example will create a database, use a simple form to save data to that database, and retrieve and display the stored data.

Let's review the following code. At the top, you will see the database file defined as a file called *users.db* within the `userDirectory`. Next, the `SQLConnection` is defined. Finally, several `SQLStatements` are declared and SQL strings defined, which will be used for working with the database.

Within the `applicationComplete` event handler, the `SQLConnection` is initiated, two event listeners are added to listen for `SQLEvent.OPEN` and `SQLErrorEvent.ERROR`, and finally, the `openAsync` method is called and the *db* file is passed in.

After the database is opened, the `openHandler` function is called. Within this function, the `SQLEvent.OPEN` event listener is removed. Next, the `createTableStmt` is created, configured, and executed. This statement will create a new table called Users if it doesn't yet exist. If this statement is successful, then the `createResult` method is called. Within the createResult method, the SQLEvent.RESULT event is removed and the selectUsers method is called.

Within the selectUsers method, the selectStmt is created, configured, and executed. This statement will return all rows within the Users table. This data is then stored within the selectStmt. If this statement is successful, then the selectResult method is called. Within the selectResult method, the data is read from the selectStmt by using the getResults method and is cast to an ArrayCollection and set to the dataProvider of a DataGroup, where it is shown on screen by formatting within an itemRenderer named UserRenderer.

All the processes just described occur as chained events when the application loads up. So if there is any data in the database from previous usage, it will automatically display when the application is loaded. This is shown in Figure 5-9.

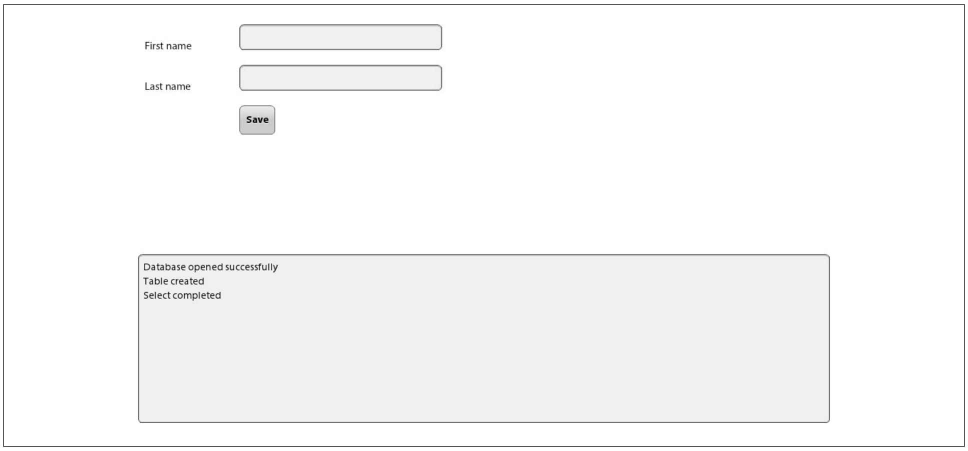
The only remaining functionality is the ability to add a new user. There are two text fields with the IDs of firstName and lastName and a Button that, when clicked, will call the button1_clickHandler function. Within the button1_clickHandler function, the insertStmt is created, configured, and executed. Notice that within the insertStmt configuration, the parameters firstName and lastName that were defined in the insertSQL are set to the text properties of the firstName and lastName TextInput components. If this statement is successful, then the insertResult method is called. Within the insertResult method, the selectUsers method is called, and the DataGroup is updated showing the newly added data. This is shown in Figure 5-10.

Here is the code for the Main Application:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               applicationComplete="application1_applicationCompleteHandler(event)">
    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            import mx.events.FlexEvent;

            private var db:File = File. applicationStorageDirectory.
resolvePath("users.db");
            private var conn:SQLConnection;

            private var createTableStmt:SQLStatement;
            private var createTableSQL:String = "CREATE TABLE IF NOT EXISTS User (" +
                                     "userId INTEGER PRIMARY KEY AUTOINCREMENT," +
                                     "firstName TEXT," + "lastName TEXT)";

            private var selectStmt:SQLStatement;
            private var selectSQL:String = "SELECT * FROM User";

            private var insertStmt:SQLStatement;
            private var insertSQL:String = "INSERT INTO User (firstName, lastName)" +
                                           "VALUES (:firstName, :lastName)";
            protected function application1_applicationCompleteHandler
(event:FlexEvent):void
            {
                conn = new SQLConnection();
                conn.addEventListener(SQLEvent.OPEN, openHandler);
                conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);
                conn.openAsync(db);
            }

            private function openHandler(event:SQLEvent):void {
                log.text += "Database opened successfully";
                conn.removeEventListener(SQLEvent.OPEN, openHandler);
                createTableStmt = new SQLStatement();
                createTableStmt.sqlConnection = conn;
                createTableStmt.text = createTableSQL;
                createTableStmt.addEventListener(SQLEvent.RESULT, createResult);
                createTableStmt.addEventListener(SQLErrorEvent.ERROR, errorHandler);
                createTableStmt.execute();
            }

            private function createResult(event:SQLEvent):void {
                log.text += "\nTable created";
                conn.removeEventListener(SQLEvent.RESULT, createResult);
                selectUsers();
            }

            private function errorHandler(event:SQLErrorEvent):void {
                log.text += "\nError message: " +  event.error.message;
                log.text += "\nDetails: " +  event.error.details;
            }

            private function selectUsers():void{
                selectStmt = new SQLStatement();
                selectStmt.sqlConnection = conn;
                selectStmt.text = selectSQL;
                selectStmt.addEventListener(SQLEvent.RESULT, selectResult);
                selectStmt.addEventListener(SQLErrorEvent.ERROR, errorHandler);
                selectStmt.execute();
            }

            private function selectResult(event:SQLEvent):void {
                log.text += "\nSelect completed";
                var result:SQLResult = selectStmt.getResult();
                users.dataProvider = new ArrayCollection(result.data);
            }

            protected function button1_clickHandler(event:MouseEvent):void
            {
                insertStmt = new SQLStatement();
                insertStmt.sqlConnection = conn;
                insertStmt.text = insertSQL;
                insertStmt.parameters[":firstName"] = firstName.text;
                insertStmt.parameters[":lastName"] = lastName.text;
                insertStmt.addEventListener(SQLEvent.RESULT, insertResult);
                insertStmt.addEventListener(SQLErrorEvent.ERROR, errorHandler);
                insertStmt.execute();
            }

            private function insertResult(event:SQLEvent):void {
                log.text += "\nInsert completed";
                selectUsers();
            }

        ]]>
    </fx:Script>
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:Label text="First name" top="35" left="10"/>
    <s:TextInput id="firstName" left="150" top="10" width="300"/>

    <s:Label text="Last name" top="95" left="10"/>
    <s:TextInput id="lastName" left="150" top="70" width="300"/>

    <s:Button label="Save" click="button1_clickHandler(event)" top="130" left="150"/>

    <s:Scroller height="200" width="100%" left="10" right="10" top="200">
        <s:DataGroup id="users" height="100%" width="95%"
                     itemRenderer="UserRenderer">
            <s:layout>
                <s:VerticalLayout/>
            </s:layout>
        </s:DataGroup>
    </s:Scroller>

    <s:TextArea id="log" width="100%" bottom="0" height="250"/>

</s:Application>
```

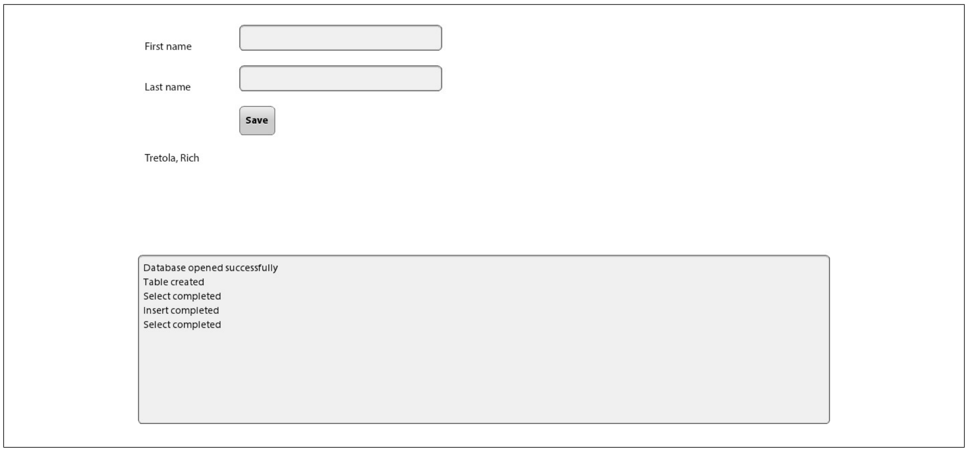
The code for the UserRenderer

```
<?xml version="1.0" encoding="utf-8"?>
<s:ItemRenderer xmlns:fx="http://ns.adobe.com/mxml/2009"
                    xmlns:s="library://ns.adobe.com/flex/spark">
    <s:Label text="{data.lastName}, {data.firstName}"/>
</s:ItemRenderer>
```



```
Database opened successfully
Table created
Select completed
```

Figure 5-9. SQLite sample application



Tretola, Rich

```
Database opened successfully
Table created
Select completed
Insert completed
Select completed
```

Figure 5-10. After adding a record

CHAPTER 6

OS Interactions

Open in Browser

From within your application, you can open a link using the device's native browser, just like with a traditional, browser-based Flex application. The URLRequest class accomplishes this. The URLRequest class accomplishes this. Let's take a look at the following code. Simply creating a new URLRequest and passing it into the navigateToURL method will invoke the user's browser to handle the request. Figure 6-1 shows the sample application running.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark">

    <fx:Script>
        <![CDATA[
            protected function sendIt_clickHandler(event:MouseEvent):void
            {
                var s:String = "";
                s+= address.text;
                navigateToURL(new URLRequest(s));

            }
        ]]>
    </fx:Script>

    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:Label text="URL" top="40" left="50"/>
    <s:TextInput id="address" top="30" left="160" text="http://www.happytoad.com"
width="250"/>
    <s:Button id="sendIt" label="Open" click="sendIt_clickHandler(event)" top="110"
left="160"/>
</s:Application>
```

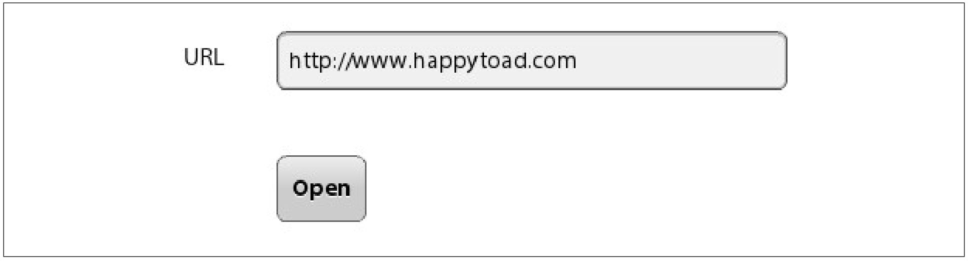


Figure 6-1. Open a link in a browser

Splash Screen

Adobe has made it very easy to add a splash screen to your application. A splash screen is an image that loads first and displays while the application is loading. There are also several options for the splash-screen display. Let's take a look at the following code, which shows the `splashScreenImage` property set to a .png image. Figure 6-2 shows a splash screen with the default settings.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               splashScreenImage="@Embed('happytoad.png')">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
</s:Application>
```

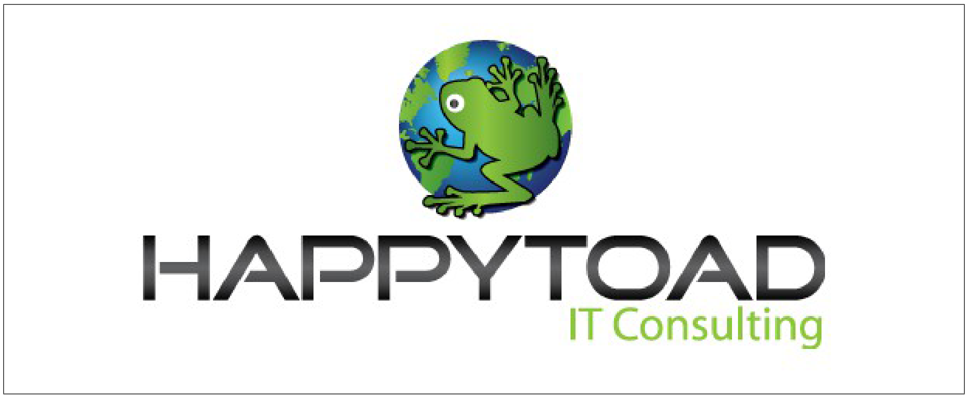


Figure 6-2. Splash screen with splashScreenScaleMode set at none

You can also set some options for the splash screen. Setting the `splashScreenMinimumDisplayTime` and `splashScreenScaleMode` properties on the `Application`, `ViewNavigatorApplication`, or `TabbedViewNavigatorApplication` tag sets these options. The next example sets the display time to three seconds and the scale mode to "stretch."

The available options for the `splashScreenScaleMode` property are letterbox, none, stretch, and zoom. Figure 6-3 and the following code show a splash screen with the `splashScreenScaleMode` set to stretch.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               splashScreenImage="@Embed('happytoad.png')"
               splashScreenMinimumDisplayTime="3000"
               splashScreenScaleMode="stretch">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
</s:Application>
```

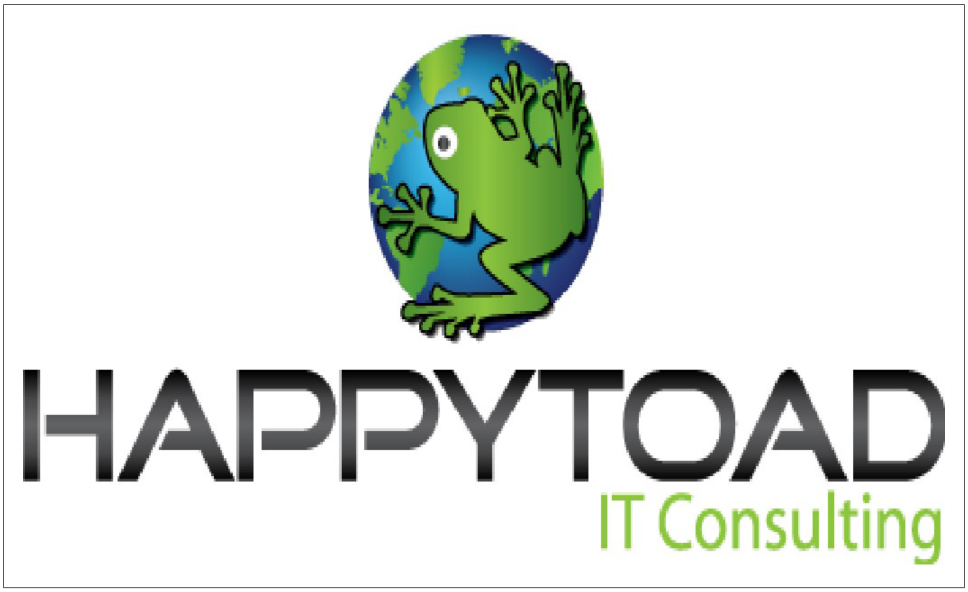


Figure 6-3. Splash screen with splashScreenScaleMode set at stretch

StageWebView

The `StageWebView` allows for web (HTML and Flash, on supported devices) and video content to be loaded into a Flex application. `StageWebView` will utilize the native browser to load HTML into your application.

Let's review the following code. First, you will notice that there is a private variable named `stageWebView` declared of type `flash.media.StageWebView`. Within `applicationComplete` of the application, an event handler function is called, which first checks to see if the device supports `StageWebView` by reading the static property of the `StageWebView` class. If this property returns as true, a new instance of `StageWebView` is created and a new Rectangle is created, sized to fill the remaining screen and set to the viewport property of the stageWebView.

There is a TextInput component with the ID of urlAddress—which holds the address that will be shown in the StageWebView—and a Button with the label "GO."

Clicking on the GO button will call the button1_clickHander method. Within the button1_clickHander method, the loadURL method is called with the urlAddress.text property passed in. This triggers the StageWebView to load the URL.

The results are shown in Figure 6-4.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               applicationComplete="application1_applicationCompleteHandler(event)">
    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;

            private var stageWebView:StageWebView;
            private var rect:Rectangle;

            protected function application1_applicationCompleteHandler
(event:FlexEvent):void
            {
                if(StageWebView.isSupported==true){
                    stageWebView = new StagewebView();
                    stageWebView.viewPort = new Rectangle(5,80,stage.width-10,
stage.height-90);
                    stageWebView.stage = this.stage;
                } else {
                    urlAddress.text = "StageWebView not supported";
                }
            }

            protected function button1_clickHandler(event:MouseEvent):void
            {
                stageWebView.loadURL(urlAddress.text);
            }

        ]]>
    </fx:Script>
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:TextInput id="urlAddress" left="5" right="80" top="15"
text="http://www.google.com"/>
    <s:Button right="5" top="5" label="GO" click="button1_clickHandler(event)"/>

</s:Application>
```

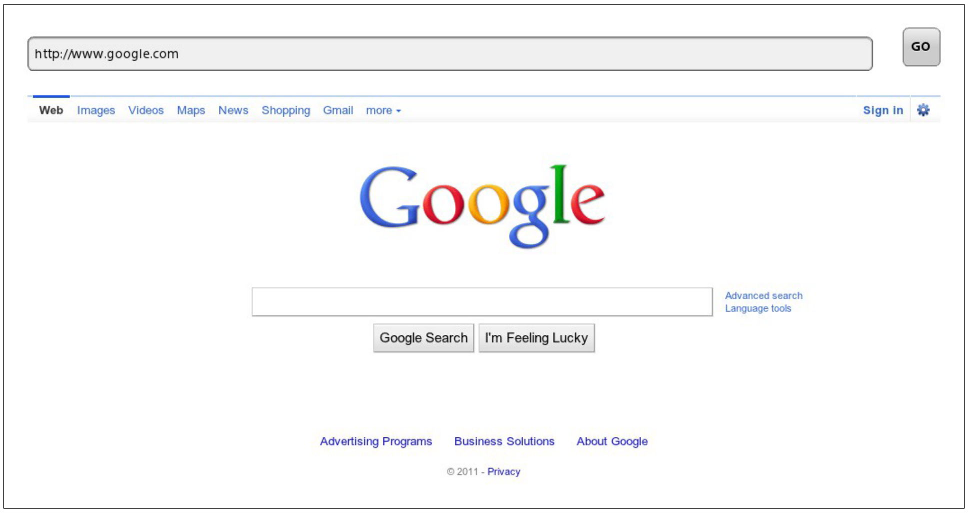


Figure 6-4. StageWebView with Google.com loaded

Screen Options

There are several options available to programmatically control several areas of the screen layout. These options determine the layout of the application, whether to show the action bar in View-Based or Tabbed application, and whether to show the application in full-screen mode. This sample application is shown in Figure 6-5.

Layout

The options for your application layout are portrait, where the application shows vertically in the device; or landscape, where the application shows horizontally in the device. Setting the aspect ratio by calling the `setAspectRatio` method on the `stage` can change the application's layout. The `StageAspectRatio` class contains two static values that should be used to set the aspect ratio.

The following code includes a `RadioGroup` with the ID of orientation. There are two `RadioButton` components in this group with values of portrait and landscape. When clicking on one of these radio buttons, the `radiobutton1_clickHandler` method is called. Within this method, the `orientation.selectedValue` is tested. If `orientation.selectedValue` is equal to portrait, the `stage.setAspectRatio` method is called and `StageAspectRatio.PORTRAIT` is passed in. If `orientation.selectedValue` is equal to landscape, the `stage.setAspectRatio` method is called and `StageAspectRatio.LANDSCAPE` is passed in. The results are pictured in Figure 6-6.

ActionBar

The ActionBar is the built-in navigation that comes along with the View-Based or Tabbed application layouts. This bar consumes significant screen real estate. Therefore, the option to hide and show it programmatically is available to you as the developer.

The following code includes a CheckBox with the label "Show ActionBar". This CheckBox is set to "selected" by default, as that is the normal state of the ActionBar. When clicking on this CheckBox to check or uncheck the value, the checkbox2_clickHandler is called. The actionBarVisible property of this view is set to the value of the CheckBox. The results are pictured in Figure 6-7, which shows an application that is in full screen with the ActionBar hidden.

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="HomeView">

    <fx:Script>
        <![CDATA[
            protected function checkbox2_clickHandler(event:MouseEvent):void
            {
                this.actionBarVisible = event.target.selected;
            }

            protected function radiobutton1_clickHandler(event:MouseEvent):void
            {
                if(orientation.selectedValue == "portrait"){
                    stage.setAspectRatio(StageAspectRatio.PORTRAIT);
                } else if(orientation.selectedValue == "landscape"){
                    stage.setAspectRatio(StageAspectRatio.LANDSCAPE);
                }
            }

        ]]>
    </fx:Script>

    <fx:Declarations>
        <s:RadioButtonGroup id="orientation"/>
    </fx:Declarations>

    <s:VGroup top="20" left="10">
        <s:CheckBox click="checkbox2_clickHandler(event)" label="Show ActionBar"
                  selected="true"/>
        <s:RadioButton groupName="orientation" value="portrait" label="Portrait"
                       click="radiobutton1_clickHandler(event)" selected="true"/>
        <s:RadioButton groupName="orientation" value="landscape" label="Landscape"
                       click="radiobutton1_clickHandler(event)"/>
    </s:VGroup>


</s:View>
```

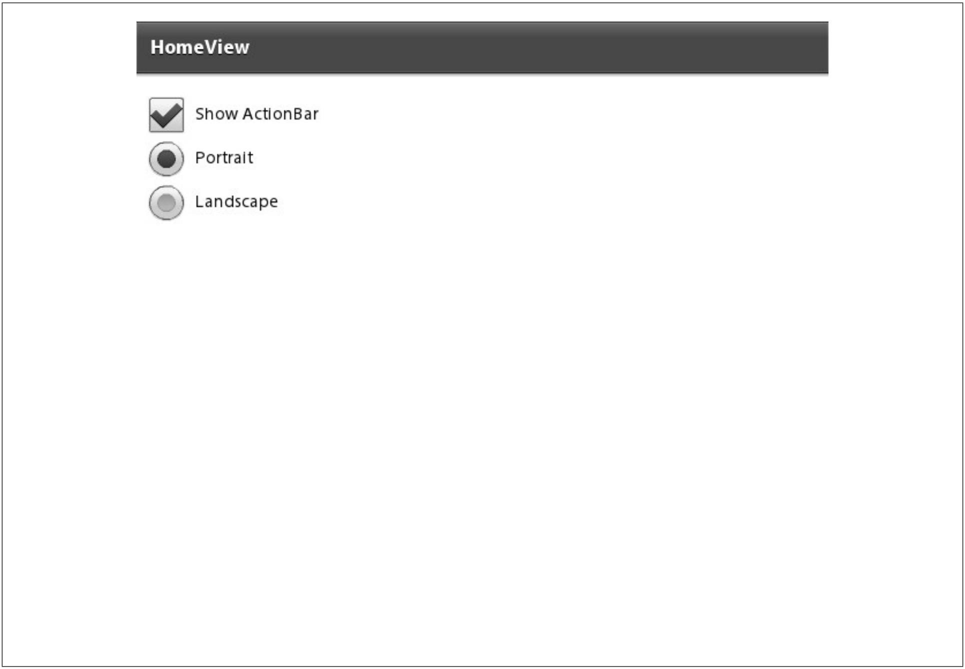


Figure 6-5. Screen Options Application

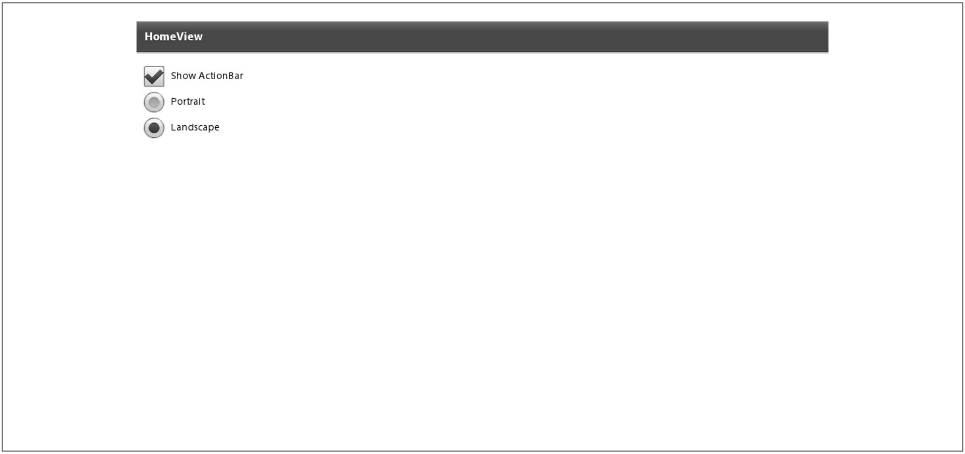


Figure 6-6. Landscape Mode

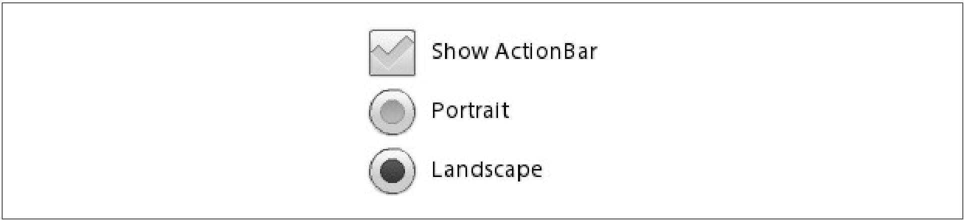
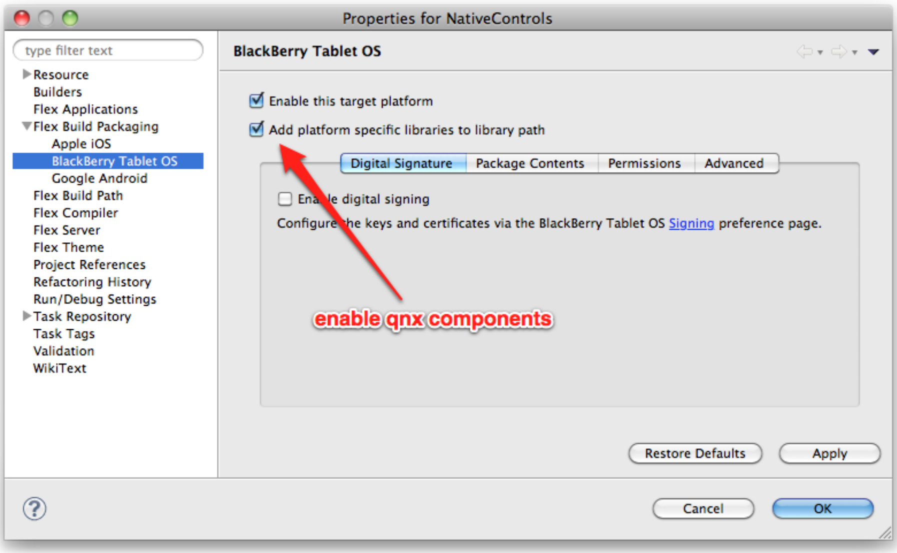


Figure 6-7. ActionBar Hidden

Native QNX Components

RIM has been nice enough to include an SWC library file within their SDK so that we can use their native components within our Flex applications. They have also provided the classes in this SWC to get some additional information about the environment. The next example demonstrates some of what is available. For the full documentation on these classes, visit *http://www.blackberry.com/developers/docs/airapi/1.0.0/*.

To enable these components within your application, simply check the box that says "Add platform specific libraries to library path" within the Preferences→Flex Build Packaging→BlackBerry Tablet OS. See Figure 6-8.



☑ Enable this target platform
☑ Add platform specific libraries to library path
☐ Enable digital signing
Configure the keys and certificates via the BlackBerry Tablet OS Signing preference page.

Figure 6-8. Enable qnx libraries

Let's examine the following code. Because the `qnx` library components do not inherit from Flex base UI components, you will have to declare them either in the declarations block and within ActionScript and then to a `UIComponent` wrapper, or create the instance of them with ActionScript and add them with ActionScript to a `UIComponent` wrapper.

Within the `fx:Declarations` block, I have defined several `qnx` components:

- `qnx.ui.buttons.LabelButton`
- `qnx.ui.buttons.IconButton`
- `qnx.ui.buttons.BackButton`
- `qnx.ui.buttons.CheckBox`
- `qnx.ui.progress.ActivityIndicator`
- `qnx.ui.picker.Picker`
- `qnx.ui.buttons.ToggleSwitch`
- `qnx.ui.progress.PercentageBar`

There is an event listener defined for both the `LabelButton` and the `IconButton`.

Within my `fx:Script` block, I have defined an Array of days, as well as some methods to handle events. On application complete, the `application1_applicationComplete Handler` method is called. Within this method, the `LabelButton` is added as a child of the `ui1 UIComponent` holder; the icon is set for the `IconButton` and it is added to `ui2`; the `BackButton` is added to `ui3`; the `CheckBox` is added to `ui4`; the `ActvityIndicator` is added to `ui5`; the `dataProvider` for the Picker is set and it is added to `ui6`; the `ToggleSwitch` is added to `ui7`; and the `PercentageBar` is added to `ui8`. The results are shown in Figure 6-9.

Clicking on the `LabelButton` calls the `showAlertDialog` method. Within this method, a `qnx.dialog.AlertDialog` is created, the title and message are defined, two buttons are added, the `dialogSize` is set to the static property of `DialogSize.SIZE_MEDIUM`, an event listener is added, and finally, the show method is called. The results are shown in Figure 6-10.

Clicking on the `IconButton` calls the `showInfoDialog` method. Within this method, a `qnx.dialog.AlertDialog` is created. This time, the message is set by reading some information from some of the non-`ui` classes that are included within the `qnx` package. The battery level is read from the `qnx.system.Device.device.batteryLevel` property; the battery state is read from the `qnx.system.Device.device.batteryState` property; the display height is read from the `qnx.display.Display.display.getDisplayHeight(0)` method; the display width is read from the `qnx.display.Display.display.getDisplay Width(0)` method; the operating system is read from the `qnx.system .Device.device.os` property; and the `qnx.utils.TimeFormatter.formatSeconds` method is used to format a time string. The results are shown in Figure 6-11.

Using these platform-specific components can certainly have benefits, but it can also render your application unusable on other device operating systems (e.g., Android, iOS), so be careful when using platform-specific classes.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               applicationComplete="application1_applicationCompleteHandler(event)"
               xmlns:buttons="qnx.ui.buttons.*"
               xmlns:mx="library://ns.adobe.com/flex/mx"
               xmlns:display="qnx.ui.display.*"
               xmlns:progress="qnx.ui.progress.*"
               xmlns:picker="qnx.ui.picker.*">

    <fx:Declarations>
        <buttons:LabelButton label="Show Alert" id="labelButton"
click="showAlertDialog(event)"/>
        <buttons:IconButton id="iconButton" click="showInfoDialog(event)"/>
        <buttons:BackButton label="Back" id="backButton"/>
        <buttons:CheckBox label="Yes" id="checkBox" width="100"/>
        <progress:ActivityIndicator id="activityIndicator"/>
        <picker:Picker id="dayPicker"/>
        <buttons:ToggleSwitch id="toggleSwitch" defaultLabel="Yes" selectedLabel="No"/>
        <progress:PercentageBar label="Saving" progress=".5" id="percentageBar"
width="200"/>
    </fx:Declarations>

    <fx:Script>
        <![CDATA[

            import mx.events.FlexEvent;
            import qnx.dialog.AlertDialog;
            import qnx.dialog.DialogSize;
            import qnx.display.Display;
            import qnx.display.IowWindow;
            import qnx.system.Device;
            import qnx.ui.data.DataProvider;
            import qnx.utils.TimeFormatter;

            private var arr:Array = [{label: "Monday", data:0},
                                {label: "Tuesday", data:1},
                                {label: "Wednesday", data:2},
                                {label: "Thursday", data:3},
                                {label: "Friday", data:4},
                                {label: "Saturday", data:5},
                                {label: "Sunday", data:6}];

            protected function application1_applicationCompleteHandler
(event:FlexEvent):void
            {
                ui1.addChild(labelButton);
                iconButton.setIcon("infoIcon.png");
                ui2.addChild(iconButton);
                ui3.addChild(backButton);
                ui4.addChild(checkBox);
                ui5.addChild(activityIndicator);
                dayPicker.dataProvider = new DataProvider([new DataProvider(arr)]);
                ui6.addChild(dayPicker);
                ui7.addChild(toggleSwitch);
                ui8.addChild(percentageBar);
            }

            private function showAlertDialog(event:Event):void
            {
                var alert:AlertDialog = new AlertDialog();
                alert.title = "Question";
                alert.message = "Do you love writing Flex 4.5" +
                              "code for BlackBerry Tablet OS?";
                alert.addButton("Hell Yeah!");
                alert.addButton("No way");
                alert.dialogSize= DialogSize.SIZE_MEDIUM;
                alert.addEventListener(Event.SELECT, alertButtonClicked);
                alert.show(IowWindow.getAirWindow().group);
            }

            private function showInfoDialog(event:Event):void
            {
                var alert:AlertDialog = new AlertDialog();
                alert.title = "Details";
                var info:String = "Battery Level: " + Device.device.batteryLevel + "\n" +
                    "Battery State: " + Device.device.batteryState + "\n" +
                    "Display Height: " + Display.display.getDisplayHeight(0) + "\n" +
                    "Display Width: " + Display.display.getDisplayWidth(0) + "\n" +
                    "Operating System: " + Device.device.os     + "\n" +
                    "Time Formatter: " + TimeFormatter.formatSeconds(22222, true );
                alert.message = info;
                alert.addButton("Close");
                alert.dialogSize= DialogSize.SIZE_MEDIUM;
                alert.show(IowWindow.getAirWindow().group);
            }

            private function alertButtonClicked(event:Event):void
            {
                trace("Button Clicked Index: " + event.target.selectedIndex);
            }

        ]]>
    </fx:Script>


    <mx:UIComponent id="ui1" y="10"/>
    <mx:UIComponent id="ui2" y="60"/>
    <mx:UIComponent id="ui3" y="110"/>
    <mx:UIComponent id="ui4" y="160"/>
    <mx:UIComponent id="ui5" y="210"/>
    <mx:UIComponent id="ui6" y="310"/>
    <mx:UIComponent id="ui7" y="410"/>
    <mx:UIComponent id="ui8" y="460"/>

</s:Application>
```

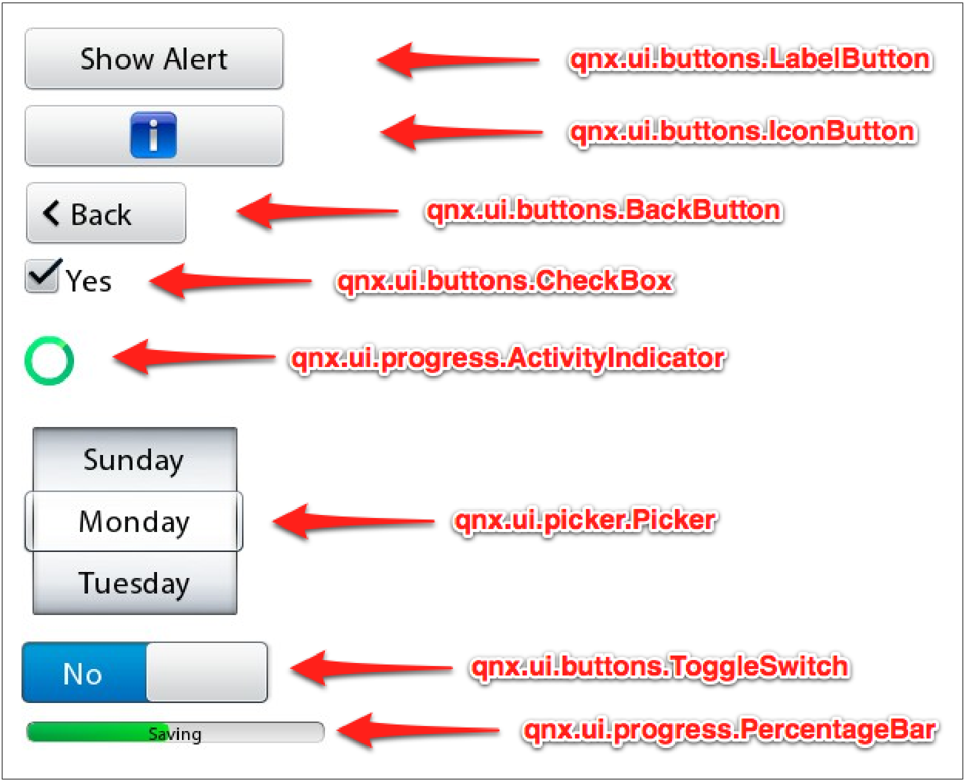
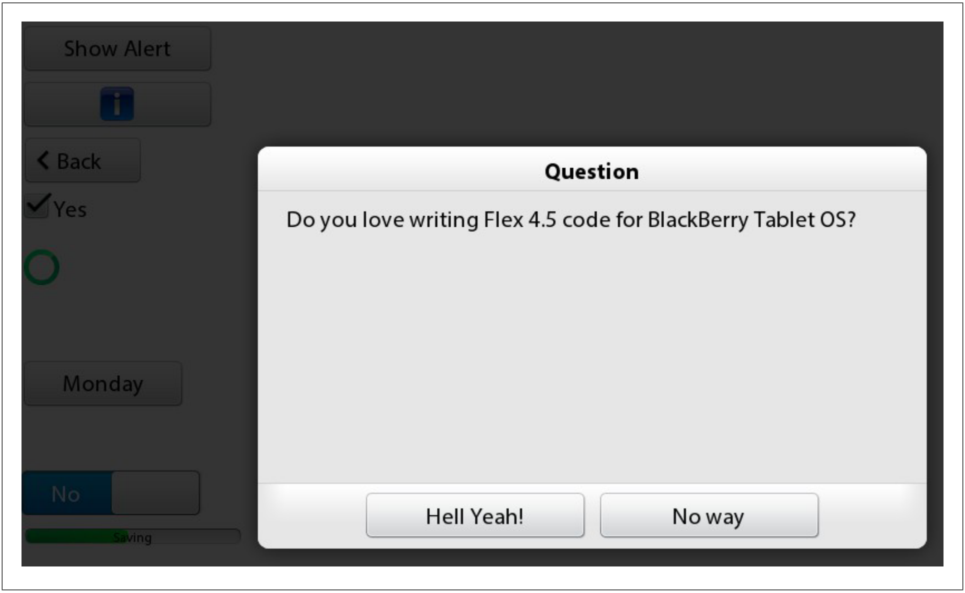

Show Alert
qnx.ui.buttons.LabelButton
qnx.ui.buttons.IconButton
Back
qnx.ui.buttons.BackButton
Yes
qnx.ui.buttons.CheckBox
qnx.ui.progress.ActivityIndicator
Sunday
Monday
Tuesday
qnx.ui.picker.Picker
No
qnx.ui.buttons.ToggleSwitch
Saving
qnx.ui.progress.PercentageBar

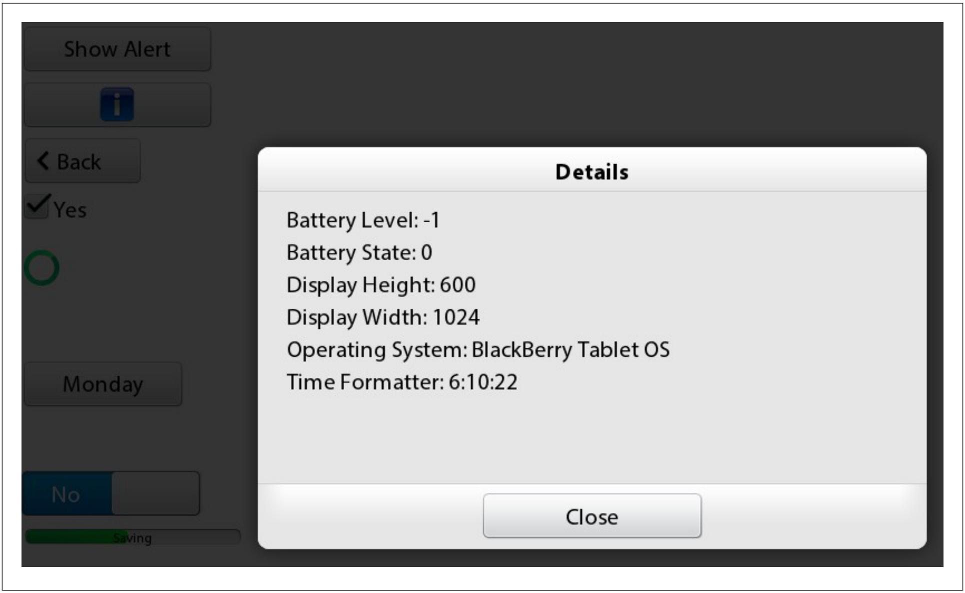
Figure 6-9. qnx components



Question

Do you love writing Flex 4.5 code for BlackBerry Tablet OS?

Figure 6-10. qnx.dialog.AlertDialog component



Details

Battery Level: -1
Battery State: 0
Display Height: 600
Display Width: 1024
Operating System: BlackBerry Tablet OS
Time Formatter: 6:10:22

Figure 6-11. various OS data displayed in AlertDialog

CHAPTER 7

Publish to BlackBerry Installer

Now that you have created your new application, it's time to publish it to a BlackBerry installer file, which is an archive file with the extension .bar. Flash Builder provides all the tooling to accomplish this task.

To demonstrate how to compile an application to a BlackBerry installer, let's walk through the process.

First, click on File→Export within Flash Builders main menu. See Figure 7-1.

Next, select Flash Builder→Release Build. See Figure 7-2.

Within the Export Release Build window, select the Project and Application you would like to compile. See Figure 7-3.

You must check the box that says, "Enable digital signing." See Figure 7-4.

If you followed the steps in Chapter 1, you should already have a certificate compiled, and you should be registered with the RIM Signing Authority. You can see this by clicking on the signing link. Figure 7-5 shows an example of signing information. If you don't have a certificate and are not registered with the RIM Signing Authority, go back to Chapter 1 and complete this setup.

To compile the BlackBerry installer file (*.bar*), click Finish (see Figure 7-6).

Congratulations, you have just compiled your first BlackBerry application. To publish your new application to BlackBerry App World, visit *https://appworld.blackberry.com/isvportal*, create a new user, and follow the directions to upload your application for approval.

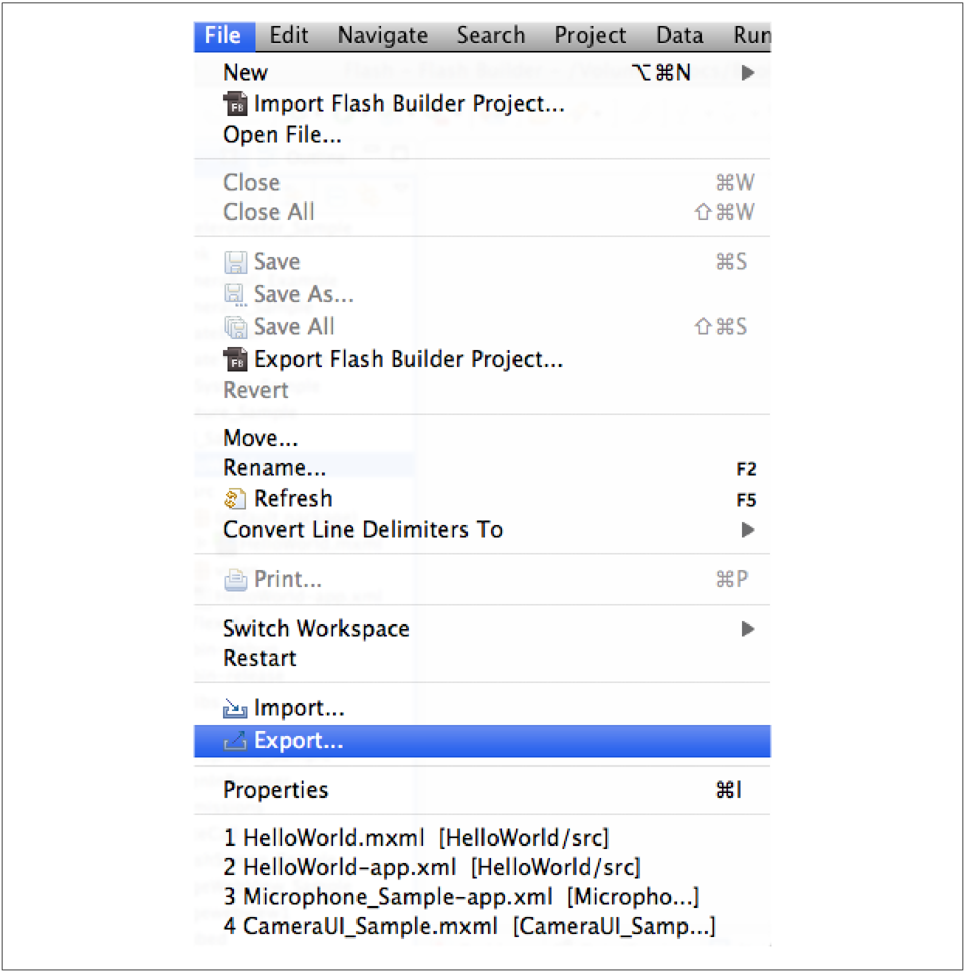
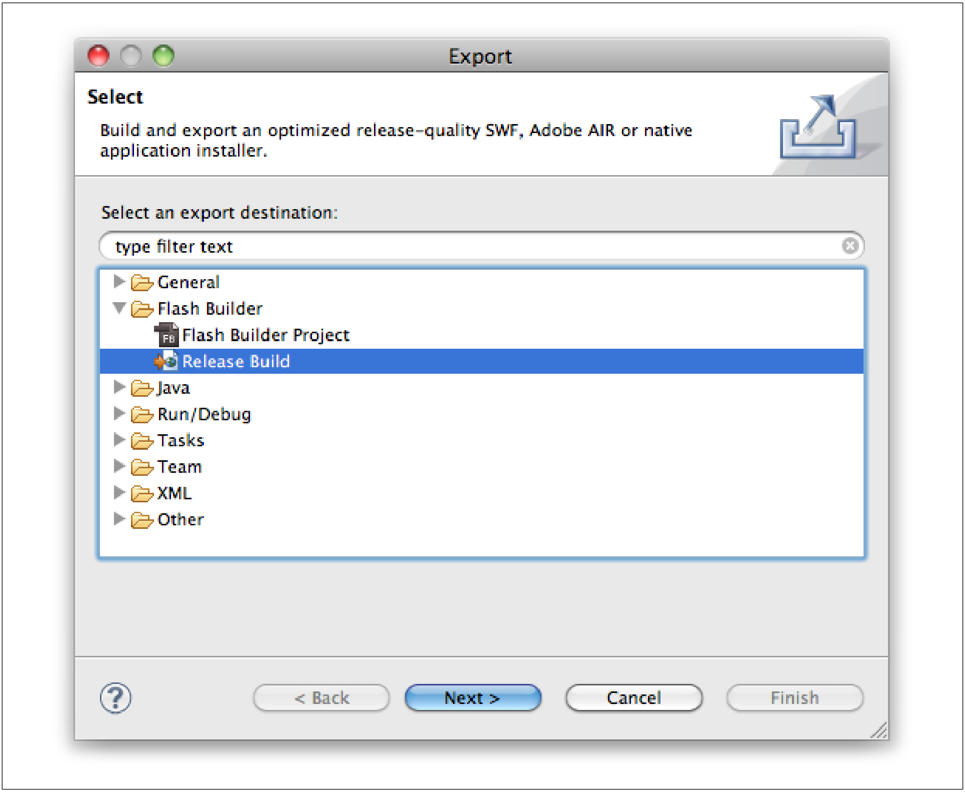

File Edit Navigate Search Project Data Run
New ⌥⌘N
Import Flash Builder Project...
Open File...
Close ⌘W
Close All ⇧⌘W
Save ⌘S
Save As...
Save All ⇧⌘S
Export Flash Builder Project...
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... ⌘P
Switch Workspace
Restart
Import...
Export...
Properties ⌘I
1 HelloWorld.mxml [HelloWorld/src]
2 HelloWorld-app.xml [HelloWorld/src]
3 Microphone_Sample-app.xml [Micropho...]
4 CameraUI_Sample.mxml [CameraUI_Samp...]

Figure 7-1. File→Export


Export

Select

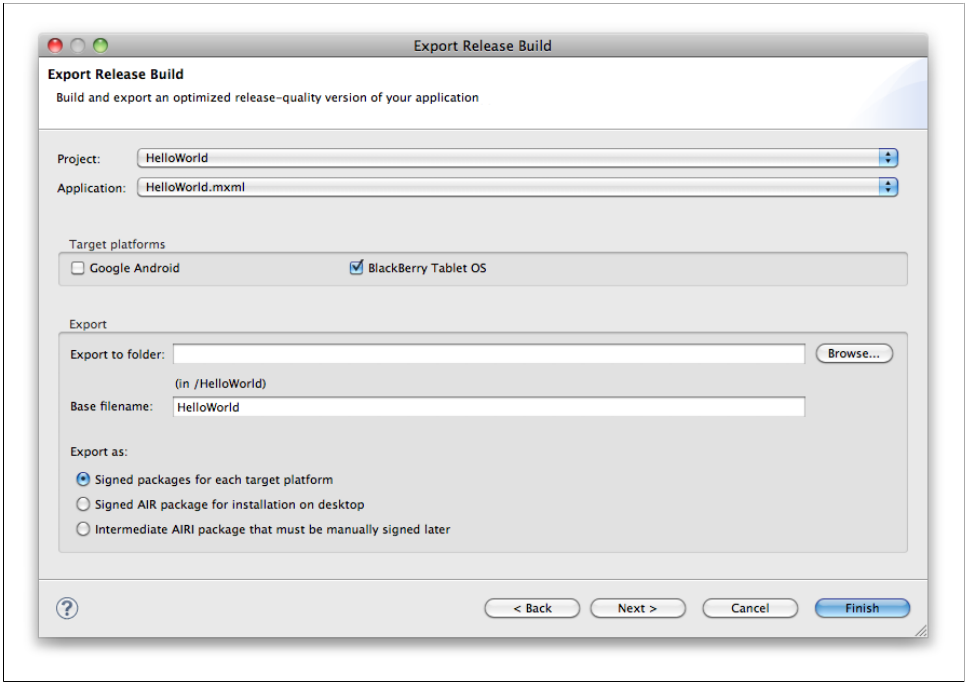
Build and export an optimized release-quality SWF, Adobe AIR or native application installer.

Select an export destination:

type filter text

- General
- Flash Builder
  - Flash Builder Project
  - Release Build
- Java
- Run/Debug
- Tasks
- Team
- XML
- Other

Figure 7-2. Flash Builder→Release Build


Export Release Build

Export Release Build

Build and export an optimized release-quality version of your application

Project: HelloWorld

Application: HelloWorld.mxml

Target platforms

☐ Google Android [✓ BlackBerry Tablet OS]

Export

Export to folder:

(in /HelloWorld)

Base filename: HelloWorld

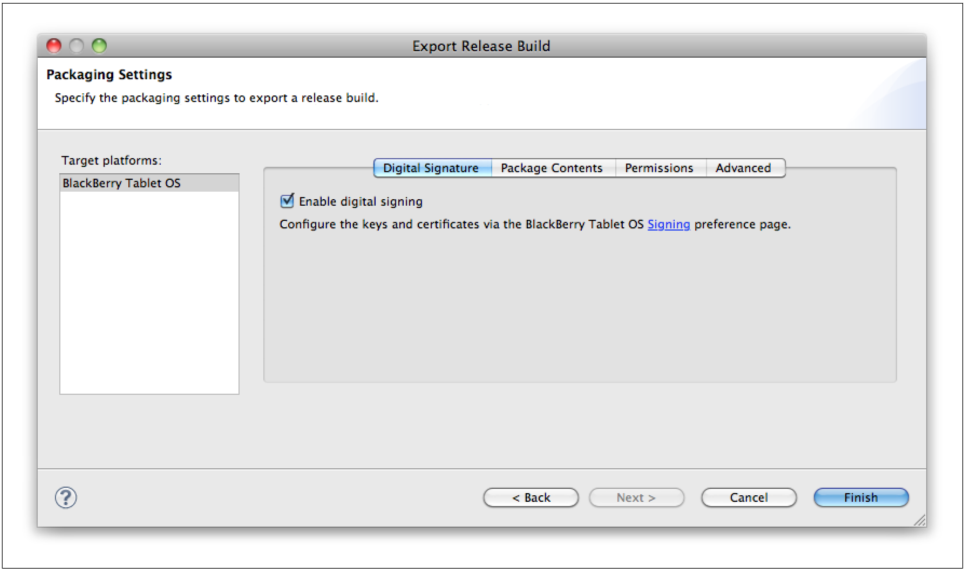
Export as:

[✓ Signed packages for each target platform]

Signed AIR package for installation on desktop

Intermediate AIRI package that must be manually signed later

Figure 7-3. Export Release Build


Export Release Build

Packaging Settings

Specify the packaging settings to export a release build.

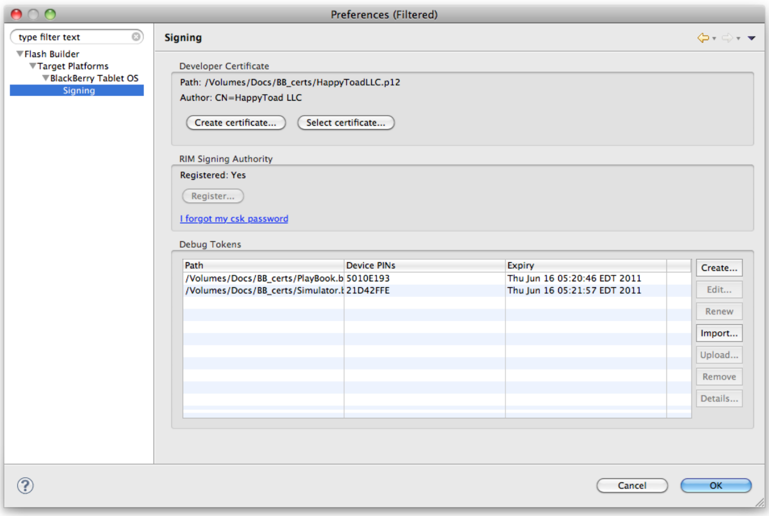
Target platforms:

BlackBerry Tablet OS

[✓ Enable digital signing]

Configure the keys and certificates via the BlackBerry Tablet OS Signing preference page.

Figure 7-4. Complete the export



Signing

Developer Certificate

Path: /Volumes/Docs/BB_certs/HappyToadLLC.p12

Author: CN=HappyToad LLC

RIM Signing Authority

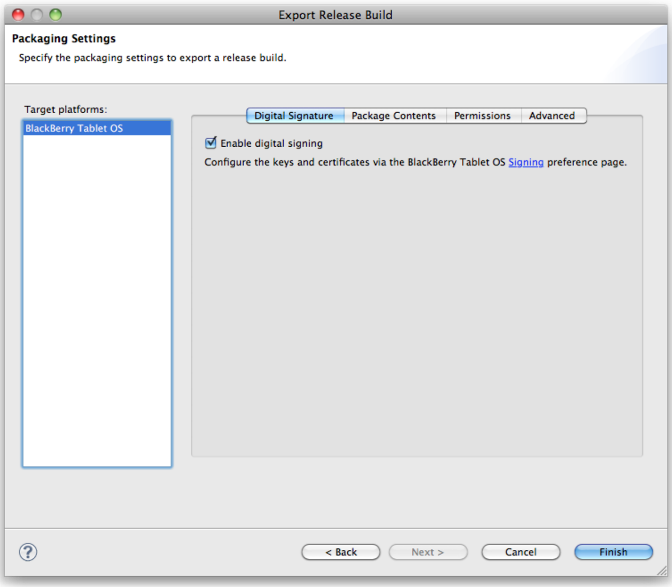
Registered: Yes

I forgot my csk password

Debug Tokens

Path	Device PINs	Expiry
/Volumes/Docs/BB_certs/PlayBook.b	5010E193	Thu Jun 16 05:20:46 EDT 2011
/Volumes/Docs/BB_certs/Simulator.b	21D42FFE	Thu Jun 16 05:21:57 EDT 2011

Figure 7-5. Signing information setup in Chapter 1



Packaging Settings

Specify the packaging settings to export a release build.

Target platforms:

BlackBerry Tablet OS

☑ Enable digital signing

Configure the keys and certificates via the BlackBerry Tablet OS Signing preference page.

Figure 7-6. Complete the export

About the Author

Rich Tretola currently holds the position of Applications Development Manager at Herff Jones, Inc.

He is an award-winning Flex developer and was the lead author of *Professional Flex 2* (Wrox) and sole author of *Beginning Adobe AIR: Building Applications for the Adobe Integrated Runtime* (Wrox). He is also a contributing author on *Adobe AIR 1.5 Cookbook* (O'Reilly) and *Flex 4 Cookbook* (O'Reilly).

Rich has been building Internet applications for over 10 years and has worked with Flex since the original Royale beta version of Flex in 2003. Other than Flex, Rich builds applications using ColdFusion, Flash, and Java. Rich is highly regarded within the Flex community as an expert in RIA and is also a five-time Adobe Community Professional. He runs a popular Flex and AIR blog at EverythingFlex.com, was the community manager of InsideRIA.com for over 3 years, and has also been a speaker at over 10 Adobe MAX sessions.

Recently, Rich has reengaged the RIA development community by funding the RIARockStars.com community and has been a principle partner in a new social polling service at twittapolls.com.

For a nontechnical escape, Rich is also co-owner of a chocolate bar–manufacturing company on Maui called WowWee Maui.

Colophon

The animal on the cover of *Developing BlackBerry Applications with Flex 4.5* is a parrot.

The cover image is from Cuvier's Animals, Dover. The cover font is Adobe ITC Garamond. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSansMonoCondensed.