

# Flash Builder and Flex Workflows for Multiscreen App Development

---

Daniel Koestler – Adobe applications developer

<http://blogs.adobe.com/koestler> or <http://goo.gl/35EF>

@antiChipotle

October, 2010

*Adobe MAX 2010 Device Lab*

## Preface

This document will walk you through the steps required to create, debug, package, and install a Flex/Android mobile project in the preview release of Flash Builder. It's intended to help you use a personal laptop with Flash Builder at the Adobe MAX 2010 Device Lab; we should have a number resource DVDs and some devices available, including the Motorola Droid X and Droid 2s. If you have any questions regarding the following content, feel free to ask the experts who are tending the lab.

This guide won't help you learn about the new APIs available when creating Flex or AIR projects for Android devices. These resources should be available to you at the device lab, or accessible online.

## Introduction

Adobe Flash Builder "Burrito" has improvements and modifications that are designed to make it easier to deploy applications to a number of platforms. The version that's now available at the MAX Device Lab makes it easier to build, debug, and package a Flex "Hero" application for all Android devices that can run Adobe AIR. As an attendee of this lab, you should have a number of Android tablets and phones available to use...all of which can run games and apps that you develop with Flex, AIR, and Flash Builder.

## Contents

Flash Builder and Flex Workflows for Multiscreen App Development.....	1
Preface .....	1
Introduction .....	1
Creating a New Mobile Project.....	4
Importing an Existing Mobile Project.....	6
Adding Code.....	6
Building .....	8
Running and Debugging on the Desktop and on a Device.....	8
Preparing to Run or Debug .....	8
Running and Debugging on the Desktop .....	10
Running .....	10
Debugging .....	11
Running and Debugging on the Device.....	11
Running .....	11
Debugging .....	12
Your First Flex “Hero” App: “Hello, World!” .....	13
Packaging for Release .....	16
Installing the Android SDK .....	17
Appendix: Additional Hints .....	19
Installing the USB driver on Windows .....	19
Conditional Compilation .....	19
Step One: Configure the Flex Compiler.....	20
Step Two: Create Conditional Blocks of Code.....	20
Managing SDKs.....	21
Step One: Installing the Flex and AIR SDKs to the Proper Directory.....	21
Step Two: Configuring Flash Builder to Use the New SDKs .....	23
Additional Resources .....	24
Tutorials/Guides.....	24
Sample App/Code .....	24
Data Centric Design.....	25
Packaging for Release .....	25

Android SDK .....	25
SDKs.....	25
Building .....	26

## Creating a New Mobile Project

1. In the menu, click File -> New -> Flex Mobile Project
2. Specify the Project name (e.g. "test").
3. Keep the default location, or specify your own. You'll want to make note of this directory, as you'll frequently want to manage assets or backups using it.
4. Leave the SDK as default (currently "Flex Hero"). Refer to the section on managing SDKs in the appendix for more information.
5. Click "Next."
6. Choose Google Android for the target platform, Mobile Application as the Application Template, and decide whether you want the application to automatically switch between landscape and portrait orientations (Figure 1).
7. Click "Next."

If you're connecting to a remote server, you now have the option of selecting an ASP.NET, ColdFusion, J2EE, or PHP backend. These settings can be configured later as well, and are part of Flash Builder's "Data Centric Design" features. If you select one of these server types, you'll be shown some additional configuration options. See the *Resources* section for more information.

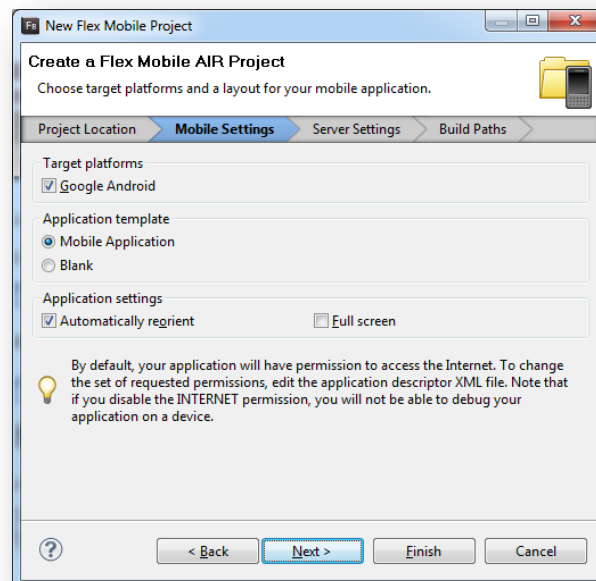


Figure 1 – Mobile Settings

8. For now, select "None/Other" as the Application server type.

Look at the name of output folder, which is "bin-debug" by default. This is where Flash Builder automatically adds and removes the APK, SWF, application descriptor, and asset files necessary to debug your application. This directory is not crucial for exporting release builds of your app, and you typically don't need to edit files in it. Leave it as "bin-debug."

9. Click Next to go to the "Build Paths" screen.

This screen lets you add libraries or source paths that you know you'll need in your application, and allows you to change some file names and parameters. For our purposes, you only need to change the Application ID, which is a unique identifier AIR uses to help identify your application.

10. Set the Application ID to a unique string. If you're creating a test project, any name is sufficient (e.g. "test"), but a real-world application should use something similar to a Fully Qualified Domain Name. For example, the ID "com.mydomain.MyApp" is a good choice, where "MyApp" is the actual name of your application. Choose your Application ID now, and click Finish (Figure 2).

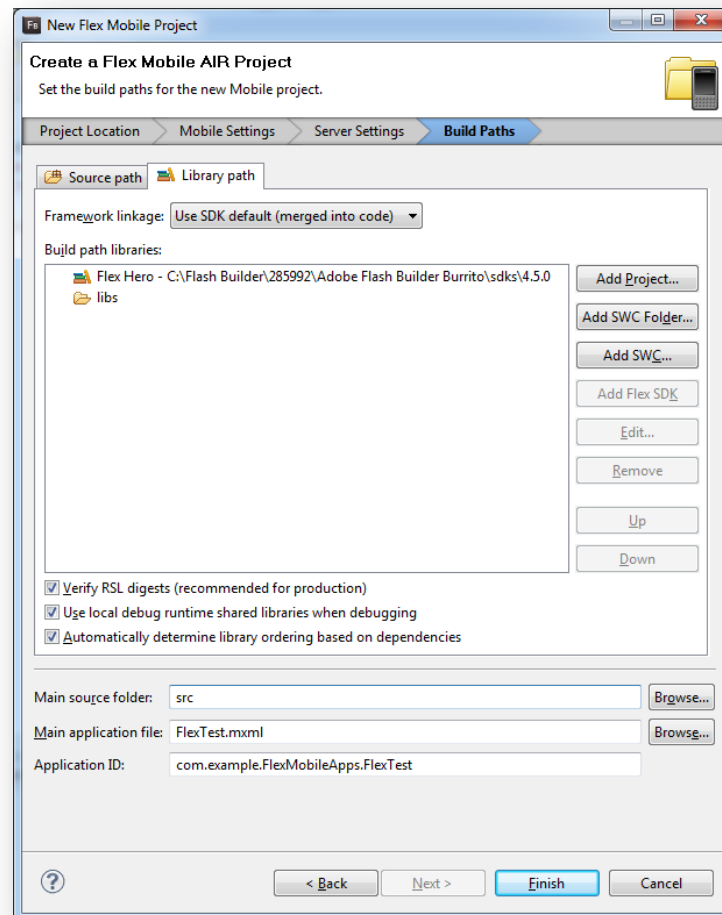


Figure 2 - Build Paths screen. Note the Application ID

At this point Flash Builder should have the code editor open, and it should be displaying the root MXML file in your new Flex Mobile Application.

## Importing an Existing Mobile Project

You may wish to import an existing mobile project, rather than create a new one, though you can skip this step if you don't have a project to import. (If you'd like one, see the "SurveyApe" application in the *References* section of this document.) If you've been provided with a Flash Builder Project in the form of an FXP, FXPL, or ZIP file, or if you have a Flash Builder Project that already exists in a writable directory, the process is straight forward:

1. In Flash Builder, go to File -> Import Flash Builder Project.
2. Choose either "File" or "Project Folder," and click Browse.
3. Navigate to the appropriate file or location, and click Open.

You now have the option to import a copy of the project, merge the imported project with an existing project, or to overwrite an existing project.

4. Make a selection, and click Finish.

The imported project should now appear in the Package Explorer.

If you have a directory of source files which don't include the metadata necessary to import an existing Flash Builder project (which often happens after downloading source code from github or Google Code), you don't want to import an existing Flash Builder project, but rather create a new project on top of your existing directory structure. To do this, follow the steps in the *Creating a New Mobile Project* in this handbook, but make two configuration changes:

1. In the "Project location" screen, set the destination folder to the root directory in your downloaded code (*step 3*).
2. In the "Build paths" screen, make sure that "Main source folder" is set to the directory that contains all of the source code for your existing project (*step 10*). This is very commonly "src," but it can vary.

## Adding Code

As you work on your application, you're certainly going to want to add Flex and ActionScript components. Whether you're typing code by hand or copying and pasting samples from the Internet, the first step is to create the proper type of file in your project. If you're unfamiliar with this process, it's extremely easy to learn:

1. In the Package Explorer, expand your project and locate the source folder (e.g. "src").
2. Right click the folder and click "New," to see a list of available file types. Alternatively, you can click File -> New in the menu.
3. Choose a file type. You are likely to use "MXML Component" and "ActionScript Class" the most. You can also modify any file you create to have a different extension and contain different content; choosing a file type is really just a quick way to have Flash Builder populate the file with a code template. For the purposes of the next steps, click "MXML Component."

The most important setting in this next screen is to determine which package you want the component to be in. Packages are equivalent to directories in a file system, though are all contained as subdirectories under your source folder. (You can easily change the package of an existing class by moving it to another directory and updating the code.)

The typical packages I have in a mobile application are:

- assets (images and XML files)
  - components (discrete, reusable AS or MXML Sprites, DisplayObjects, etc.)
  - database (database access objects, wrappers)
  - model
  - skins
  - utils (I try to minimize these, but they can be static classes for logging, debugging, etc.)
  - views (for Flex "Hero," these are all my s:View components)
4. For now, type "views" into the package field, or browse to it by clicking Browse. (You can also right click a package and insert a component from there, so that you don't have to manually enter the package name each time.)

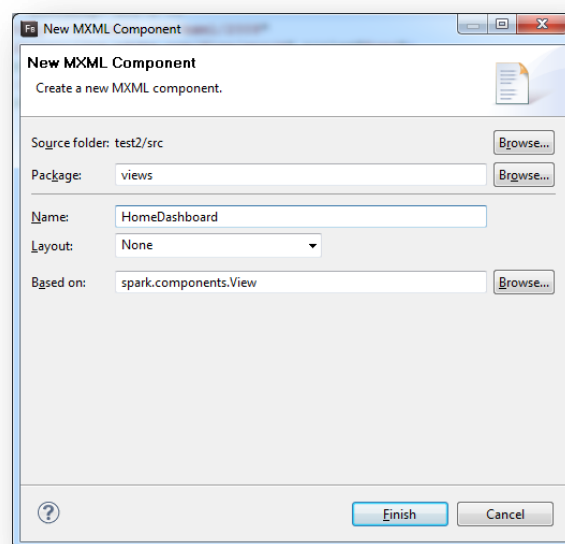


Figure 3 - New MXML Component

5. Enter the name of a component. Class names should be with a capital letter, and should be CamelCase. For now, type in “HomeDashboard.”

spark.components.View is already selected in “Based on.” You can type in or browse for a different component that you wish to extend.

6. Click “Finish.”

## Building

If you’re running Flash Builder on a modern machine, you should have “Build Automatically” checked in the Project menu. You should also be aware of the “Clean” option in the same menu, which will delete the contents of the bin-debug directory, and is useful if you feel you’re encountering unexpected behavior with Flash Builder or your application.

Compiler errors and warnings will appear in the “Problems” panel, which you can add to other parts of the interface by going to Window -> Show View -> Problems. This is especially helpful to add when you’re in the debug perspective, which doesn’t display the panel by default.

## Running and Debugging on the Desktop and on a Device

You should now have enough information to create, import, or compile a mobile application. The following steps will walk you through running debug versions, first on the desktop, and then on a device.



Figure 4 - The Debug Button

### Preparing to Run or Debug

Flash Builder allows you to quickly run and debug applications on the desktop or on a device. Open a mobile project before following the following steps:

1. Locate the Debug button in the toolbar at the top of the window (Figure 4).
2. Click the arrow next to the button, and select Debug Configurations. In this window you can control whether to launch on the device itself, or to simulate it (but not emulate it) on the desktop with ADL.
3. In the “Name” field, enter the name of your project, and then “Desktop,” to unambiguously indicate where this configuration will launch your app. E.g., type in “TestProject Desktop.” **Note that this configuration can be used for both debugging *and* running, but it will always target the desktop.**

4. Make sure that the correct project is specified in the “Project” field. If not, browse or enter it.
5. Select “Google Android” as the target platform, if it’s not already selected..
6. Click the radio button next to “On desktop,” and then choose “Droid 2” or another device from the drop down list. These device configurations control the screen size and pixel density that ADL should simulate. You can click “Configure” and add a new one, but typically you can just pick any Android device with a similar screen size to the device you have in mind.
7. Note the “Clear application data on each launch” check box, but leave it unchecked. This is useful if you want to clear any stored data, to ensure you’re launching the app as if for the first time.
8. Click Apply.

You’ve now created a configuration that will always target the desktop.



Figure 5 – “Duplicate Configuration” Button

This configuration can be used for both running and debugging on the desktop.

The next step is to create a similar configuration, but to target the device itself.

9. Locate and click the “Duplicate configuration” button at the top of the Debug Configurations window, which should still be open (Figure 5).
10. Set the name of this configuration to the name of your project and then “Device.” For example: “TestProject Device.” **Note that this configuration can be used for both debugging *and* running, but, unlike “TestProject Desktop,” it will always target the device.**
11. As in the previous steps, make sure that the project is correctly set, and that the target platform is “Google Android.”
12. Under “Launch method,” click the “On device” radio button.
13. Make sure the “Deploy the application to the device over USB” check box is checked. At this time I don’t recommend trying to deploy to the device using another method, unless you experience problems. (In this case, you could use the Android SDK to deploy the APK to your device via ADB, or drag the APK file out of the bin-debug folder and place it on the phone’s SD-Card, and then install it with an application on the device.)
14. Make sure “Clear application data on each launch” is unchecked, but note that you can check it in the future, should you want to wipe the data before installing your app.
15. Click “Apply” and then “Close.”

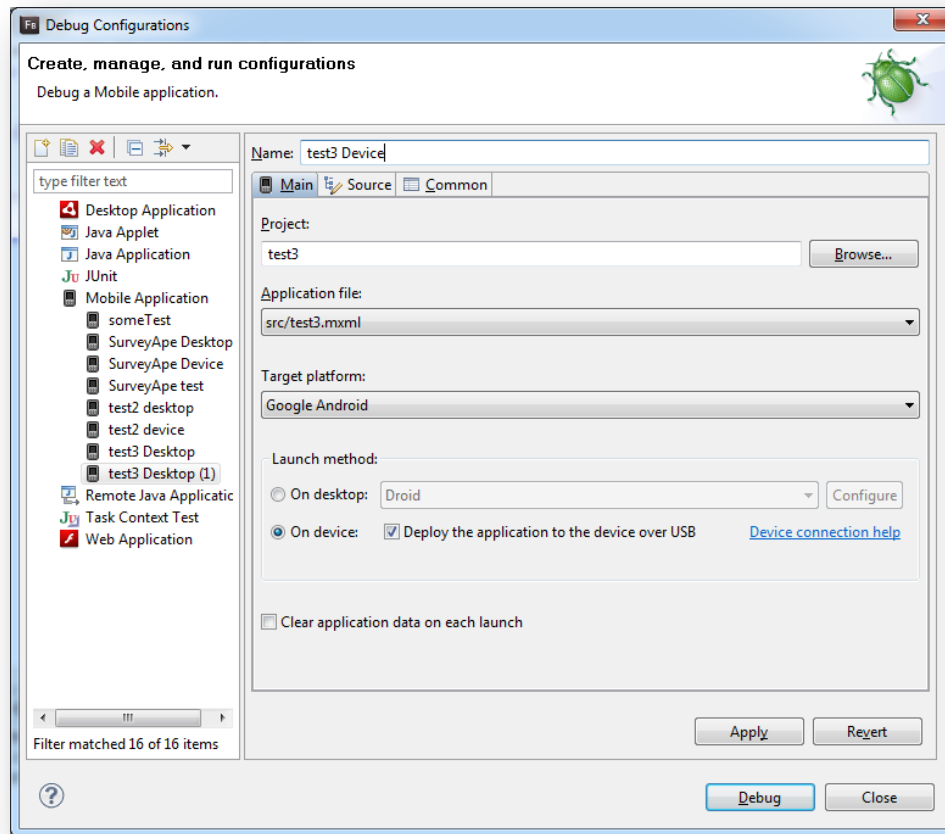


Figure 6 - The Debug Configurations Screen

At this point you've created two configurations: one that targets ADL on the desktop, and one that targets the device. Each of these configurations can be used for either debugging or running...you can debug or run on the desktop, or debug or run on the device.

## Running and Debugging on the Desktop

Both running and debugging on the desktop are straightforward.

### Running

1. With a mobile project open, click the arrow next to the run button (Figure 7).
2. Select and click the "Desktop" configuration you created in the previous steps. E.g., select "TestProject Desktop."

- ADL will run your application, which won't attempt to connect to Flash Builder for debugging.

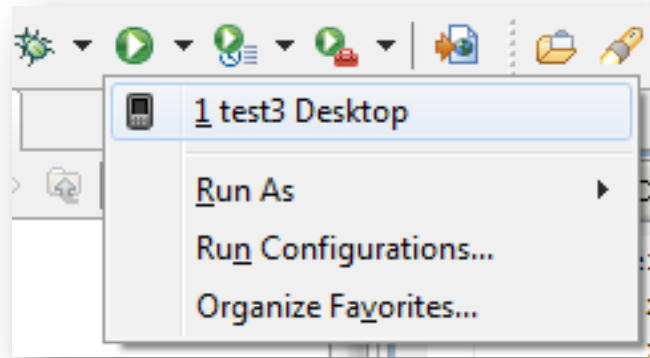


Figure 7 - The Run Button

### Debugging

- Debugging on the desktop is very similar to the “Running” step. Simply click the arrow next to the debug button and select the “Desktop” option.

## Running and Debugging on the Device

### Running

Running a mobile application on a device is quite easy. The following steps will show you to deploy and run an APK to a device connected over USB.

- Plug your device in via USB. If you're at the Adobe MAX Device Lab, you can use the Motorola Droid or Droid X devices to test your applications.
- Make sure that USB debugging mode is enabled on your device. On Android 2.2, launch the system settings application, tap “Applications,” tap “Development,” and make sure “USB debugging” is checked.
- Click the arrow next to the run button in the toolbar at the top of the Flash Builder window, and click the configuration that reads “Device.” E.g., select “TestProject Device.”
- Flash Builder will begin to automatically package an APK containing a SWF and the assets necessary to run your application, and will use the Android SDK to deploy it to the device. Your device should respond, and your application should automatically launch.

5. If you see a window that says “Choose Device,” but you don’t see your device listed, you may have to install the Android SDK. See the step *Installing the Android SDK* in this handbook for more information. After completing those steps, you should be able to launch on the device.

## Debugging

At the time of Adobe MAX 2010, debugging on devices is currently only possible in Flash Builder “Burrito” over WiFi. This is likely to change in the near future. At the moment, however, take the following steps to debug your application on a physical device:

1. If you have a VPN connection open and you think it may disallow LAN access to other hosts, disconnect it.
2. Associate both your laptop and device to the same WiFi access point.
3. Connect the device over USB. Though USB isn’t used for debugging, Flash Builder can still launch the application through it, which simplifies the process.
4. Deploy the application to the device by clicking the arrow next to the debug button, and then clicking “Device.” E.g., click “TestProject Device.”
5. Disable Windows Firewall, OS X’s firewall, or any third party firewall app, to make it easier to connect the first time. (You can then re-enable the firewall and determine which port to forward, once you’ve confirmed debugging can work without it.)
6. Flash Builder will package a debug version of your app in an APK, automatically determine your wireless IP address, and then deploy the APK over USB. You should see the application start on your device.
7. At this point the application will attempt to connect to your computer using the IP address Flash Builder provided. If it’s successful, Flash Builder should respond and remove the progress bar that indicates an operation is in progress. You can then debug the application normally.

8. If the application fails to connect, re-check your wireless settings and try again, or ask for help from one of the experts at the Device Lab.

USB debugging should be possible in Flash Builder in the near future, but unfortunately isn't available in time for Adobe MAX 2010.

## Your First Flex “Hero” App: “Hello, World!”

You now have the knowledge necessary to create, compile, and run a simple Flex “Hero” app using Flash Builder “Burrito.” Refer to the previous sections as you continue through the following steps, as you should apply what you’ve learned on your own.

1. Create a new Flex mobile project inside of Flash Builder. Call it “HelloWorld”

The root component in your application should be a `s:MobileApplication`, inside of an MXML file named “HelloWorld.” Note that it defines a property called “`firstView`.”

`s:View` Objects are `DisplayObjects` that can be pushed or popped onto a `ViewNavigator`. You can think of a `ViewNavigator` as a stack: as the user goes through your application, he or she may go a few screens forward (which will get pushed onto the `ViewNavigator`) and then use the hardware back button or application logic to pop Views off the `ViewNavigator`, and thus go back through the previous screens.

A `View` can be pushed onto a `ViewNavigator` multiple times, and if the `ViewNavigator` only contains one `View`, using the Android back button will close your application and return the user to the last application he or she was using.

`s:MobileApplication` already defines a `ViewNavigator`, and it’s accessible to `s:MobileApplication` or any child component as the member variable called “`navigator`.”

2. Inside of the first `View` of your project, add a `s:VGroup`. Give it a width and height of 100%, set the `verticalAlign` to “middle”, and set `horizontalAlign` to “center.”
3. Inside of the `VGroup`, add a `s:Label` child. Set the `Label`’s text to “Hello, World!”

Your `HelloWorldHome` component should be similar to the following:

```
<?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="Home">
  <s:VGroup width="100%" height="100%" verticalAlign="middle"
    horizontalAlign="center">
    <s:Label text="Hello, World!"/>
  </s:VGroup>
</s:View>
```

4. Using the previous sections, debug this application on the desktop. Verify that it correctly compiles and runs.
5. Using the previous sections, run this application on a phone at the Device Lab.

It's easy to use the `ViewNavigator` to push and pop Views based on user events. As the final part of this section, you'll add a `s:Button` that allows the user to see a new screen of your application, and you'll verify that the back button returns the user to the previous screen.

6. Inside the `VGroup`, as a sibling of the `Label1`, add a `s:Button`. Give it a label called "Continue."
7. In the new `Button`, define the "click" property, and set it to a function. Use Flash Builder's code hinting to generate a click handler for you, or manually type the name of a function that will handle the click event. For example, your `Button` may look like this:

```
<s:Button label="Continue" click="button1_clickHandler(event)"/>
```

8. If Flash Builder hasn't created a handler for you, create a new `fx:Script` tag as a child of the `View`. Flash Builder will automatically insert a `<![CDATA[ ... ]]` block, inside of which you'll write `ActionScript 3` code.
9. Inside of this `Script` tag, if Flash Builder hasn't already done so, create a function that corresponds to the click handler you defined in your `Button`. For example, create this function:

```
protected function button1_clickHandler(event:MouseEvent):void
{
    // Code will go here
}
```

You've now created a `Button` that, when clicked or tapped by the user, will direct program execution to the "button1\_clickHandler" function. Flex will automatically pass in a `MouseEvent` named "event," which will define information about the user-generated interaction (though this information isn't useful to us right now).

We now want to push a new `View` onto the `ViewNavigator`, so that clicking or tapping this button will take the user further into your application.

10. Inside of the `Button` click handler, reference the `MobileApplication`'s "navigator" property, and call the `ViewNavigator`'s "pushView" function. As a parameter, pass in a `Class` reference to a `View` that we'll define in the next steps. For example, add:

```
navigator.pushView(MyNewView);
```

We're now very close to testing this application. However, the View you referenced in the `pushView` function call doesn't exist yet. For this application to compile and run, we have to resolve that reference, and we'll do so by creating a new View component that has the same class name as the one you specified in `pushView` (e.g. "MyNewView").

11. Right click your project in the package explorer, or click "File" in Flash Builder's menu. Go to New -> MXML Component.

You'll see the New MXML Component window. We want to create a View inside of a package called "views." It's a good idea to use packages to organize your components.

12. Under "Package," type in "views." This can be anything you want, and is only logically connected to the type of components you'll decide to put here.
13. Under "Name," type in the name of the Class you referenced when you made the call to `pushView()`. For example, type in "MyNewView."
14. You can keep the Layout as "None," or change it if you wish. Make sure that the component is "Based on" (AKA "inherits from") `spark.components.View`, which is the full package name for the `Flex s:View` component.

Your window should look similar to Figure 8.

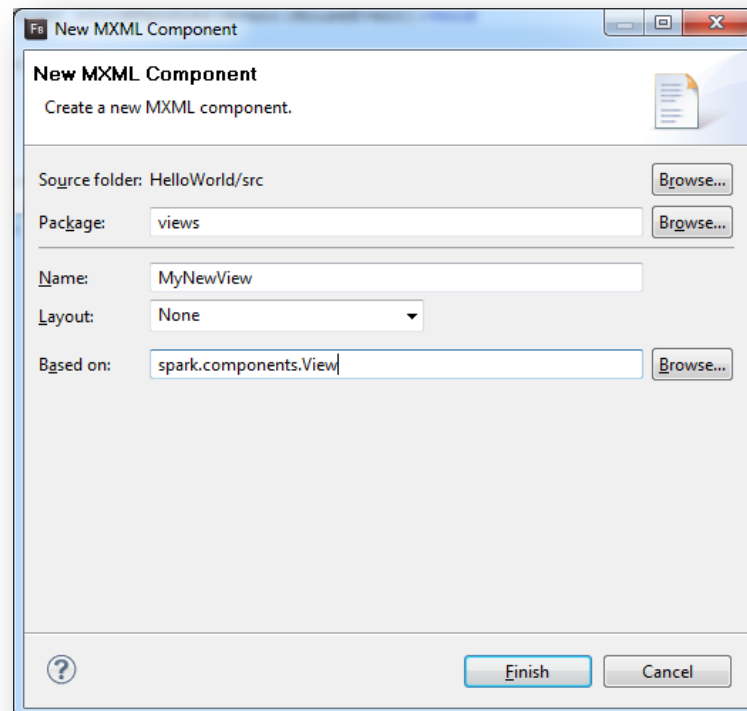


Figure 8 - Creating a new View component

You're ready to test pushing and popping this View in your application. Debug the application on the desktop using ADL.

Your application should look like Figure 9.

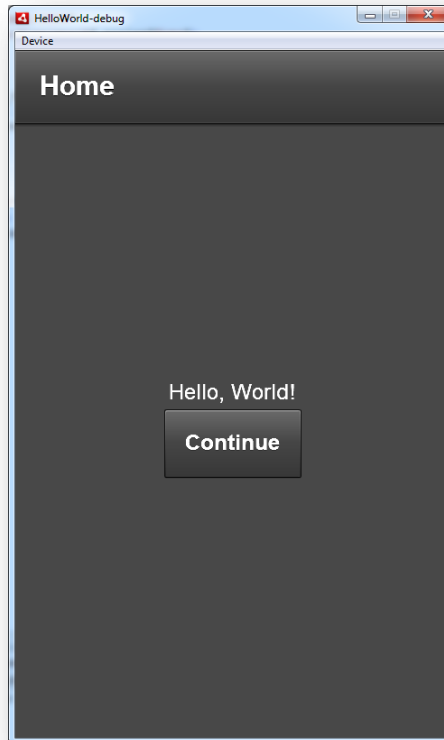


Figure 9 - "Hello, World!" running in ADL

15. Click or tap the "Continue" button. You should see a transition as the ViewNavigator pushes the new View, taking you to a different part of your application.
16. Use the Device -> Back button in ADL's menu, or tap the hardware back button on your Android device. You'll see the ViewNavigator pop this view off the stack, returning you to the firstView in your MobileApplication.

You now know the basics of using Views and the ViewNavigator.

## Packaging for Release

Once you're ready to release the final version of your application, you can export it from Flash Builder as you would any other AIR app:

1. With your project open, click Project -> Export Release Build.
2. Click "Browse" to specify a folder that will contain the exported APK. E.g., choose to save the APK to your desktop.
3. Specify the filename of the APK, or leave the default, which is the project name.
4. Make sure that "Export and sign a platform-specific application package" is selected. If you wish to have this package automatically installed, click the "Deploy application to any connected device" check box.
5. Click "Next."
6. You now have to select a digital signature that is used to sign the application. Trusted signatures are obtained from Certificate Authorities, but you can create a self-signed certificate if you wish. Either locate your existing certificate and enter your password, or click "Create," fill out the information, and save it.
7. Click the "Package Contents" tab.

You can now select the assets that are necessary for your application to run. You don't need to select any assets that you've embedded in Flex with @Embed, but you do need to include application icons, XML files that serve as real time data providers, database files, etc. The assets that you should include will vary from application to application. Note that including more assets will increase the final package size, so carefully determine which assets you want to include.

8. After selecting the appropriate assets, click "Finish."

Flash Builder should now create the APK and place it in the directory you specified earlier. Based on your preferences, Flash Builder may also deploy and install the APK on your device. If no device is connected or recognized, you'll see a warning that it wasn't successfully deployed. (If you have a device connected and USB debug mode is on, you may have to install the Android SDK. See the section *Installing the Android SDK* for more information.)

## Installing the Android SDK

If Flash Builder is having difficulty recognizing a connected device, you may have to install and/or configure the Android SDK. The Android SDK has its own downloader, and the configuration steps vary for each platform. I recommend you use the information on the SDK site at Android.com to configure it for your system:

- <http://developer.android.com/sdk/index.html> or <http://goo.gl/uZmF>

You can ask the Device Lab experts for help with this process, and/or help with getting Flash Builder to recognize an attached device.



## Appendix: Additional Hints

### Installing the USB driver on Windows

If you're running Windows, you need to install the device's drivers before attempting to run or debug on the device. The following devices are currently supported by the driver provided with Flash Builder:

- Google Nexus One
- Motorola Droid
- Motorola Droid 2
- Motorola Droid X
- HTC Droid Incredible
- HTC Evo 4G

These device driver configurations are listed in `android_winusb.inf`. Windows Device Manager accesses this file when installing the device driver. Flash Builder installs `android_winusb.inf` at the following location:

```
<Adobe Flash Builder Burrito Home>\utilities\drivers\android\android_winusb.inf
```

To install the USB device driver for an Android device, follow these steps. (Note that these steps may be slightly different on different versions of Windows.)

1. Connect your Android device to your computer's USB port.
2. If a message bubble pops up indicating that the device driver can't be found, close it.
3. Go to Control Panel and bring up Device Manager.
4. Locate the entry for your device. It should have the name of your device's manufacturer and a reference to "ADB." It will also likely have a warning icon on it, indicating that the driver is not properly installed.
5. Right-click on the item and choose "Update Driver Software..."
6. Choose to manually browse to the driver.
7. Browse to this folder in the Flash Builder installation folder:

```
<Flash Builder>/utilities/drivers/android/
```

8. Follow the steps in the wizard to finish installation.

### Conditional Compilation

It can be extremely useful to use conditional compilation in a mobile project. Using conditional compilation, you can pass arguments into the Flex compiler, which will then selectively choose to either compile or ignore predefined blocks of code; this optional code can

enable behavior that is useful for debugging purposes, but which would be undesirable in a release version of your application.

For example, you can create a block of code that will display logging information in a `s:TextArea` when you set a “`CONFIG::development`” flag, and which will simply be removed from the compilation process when this flag is unset.

This process is accomplished by first configuring the Flex compiler in Flash Builder, and then coding conditional blocks in your application.

### Step One: Configure the Flex Compiler

1. Right click a project in the Package explorer, and click “Properties.”
2. Click “Flex Compiler.”
3. In the “Additional compiler arguments” field, add your conditional compilation flag. For example, add this text to the field:

```
-define+=CONFIG::development,true
```

This will create a new flag called “`development`,” and set it to `true`. The Flex compiler will now look for any conditional blocks of code that use this flag, and use the “`true`” property to determine that it should compile them.

4. Click “OK” to close the project properties window.

### Step Two: Create Conditional Blocks of Code

You can add conditional blocks of code to any ActionScript 3 code segments, so you should first open an AS file, or add an `<fx:Script>` tag to an MXML file.

1. Inside of one of your functions, such as an event handler for a `FlexEvent.CREATION_COMPLETE` event, add a conditional compilation block similar to the following:

```
CONFIG::development
{
    this.outputTextArea.text += "\"development\" flag is
    set.";
}
```

2. You can also check for these flags in MXML, which can help you develop an application that can easily adapt its UI based on compile-time parameters:

```
<s:VGroup visible="{CONFIG::development}">
    <s:TextArea id="outputTextArea" text="(debug
output will go here)"/>
    <s:Button label="Save output to log"/>
</s:VGroup>
```

This code will only display the `s:TextArea` and `s:Button` when the `CONFIG::development` flag is set.

Conditional compilation can be extremely powerful when you want to modify your application's behavior based on whether it's a debugging version or a final release, or in other cases where you're deploying an app and don't have an API to determine some piece of information you can predict at compile time.

## Managing SDKs

Over time, you'll want to target the latest versions of AIR and Flex in the applications you create. This will require you to install new SDKs. The first step is to locate the proper directory and correctly extract the target SDK, the second step is to configure Flash Builder so that it recognizes the new SDK, and to set this SDK for your current and/or future projects.

You can obtain SDKs from the Flex and AIR homepages. Currently, Flash Builder "Burrito" ships with Flex 3.5.0 and Flex "Hero," each with the release version of AIR 2.5.

### Step One: Installing the Flex and AIR SDKs to the Proper Directory

1. Navigate to the following directory:
  - a. On Windows: `C:\Program Files\Adobe Flash Builder Burrito\sdk`
  - b. On OS X: `/Applications/Adobe Flash Builder Burrito/sdk`

Note that these paths may be slightly different for you. If you chose a custom install location, navigate to that directory instead. If you've lost Flash Builder, search for the "sdk" directory on your system.

2. Extract the Flex SDK to this folder, such that it will end up in its own directory.

For example, I downloaded an archive containing Flex 4.5.0. I created a directory named “4.5.0,” and simply dragged the contents of the ZIP file into this new folder.

The Flex SDK will ship with a recent version of AIR, but it’s possible you’ll want to install a more recent build. To do so, you’ll have to overlay files from the AIR SDK over the contents of the Flex SDK.

3. Overlay the AIR SDK, merging the downloaded files with the Flex SDK folder:
  - a. On Windows, this process is very straightforward, since Windows Explorer can merge directories. Simply drag and drop the contents of the downloaded AIR SDK over the Flex SDK directory you created. For example, drag the contents into: C:\Program Files\Adobe Flash Builder Burrito\sdk\4.5.0\
  - b. On OS X, the process is more involved, as Finder is unable to merge directories. Finder will simply delete any target folders that have the same names as folders you’re attempting to copy, which will remove large parts of the Flex SDK. The way around this is to use the Bash shell:
    - i. Open Terminal.app
    - ii. Navigate to your SDK folder by using the “cd” command. Type:

```
cd /Applications/Adobe\ Flash\ Builder\
Burrito/sdk/4.5.0
```

on one line, and hit enter. The backspaces will escape the whitespace between the words.

- iii. Verify that you’re in the correct directory by entering “pwd” and hitting enter.
- iv. Extract the AIR SDK you downloaded into this directory using the “tar” command. Type, verbatim:

```
tar jxvf ~/Desktop/AdobeAIRSDK.tbz2 .
```

which includes the period at the end of this command. This assumes the file is called AdobeAIRSDK.tbz2 and is located at /Users/<your username>/Desktop.

To learn what this command does, view the tar manual (man) page online or in the terminal.

- v. The tar command should immediately begin listing every file that's being extracted. It will correctly merge the directories.

## Step Two: Configuring Flash Builder to Use the New SDKs

You have to configure Flash Builder so that your projects use the SDK you just installed. After you make the SDK available, you can set it as the default, so that all future projects you will automatically use it.

1. Right click an existing project in the Package Explorer, and click "Properties."
2. Click "Flex Compiler."

*Note:* You should now see which SDK this project is using. By default, this is set to "Flex Hero."

3. Click "Configure Flex SDKs" to open the Flash Builder preferences window (Figure 10).

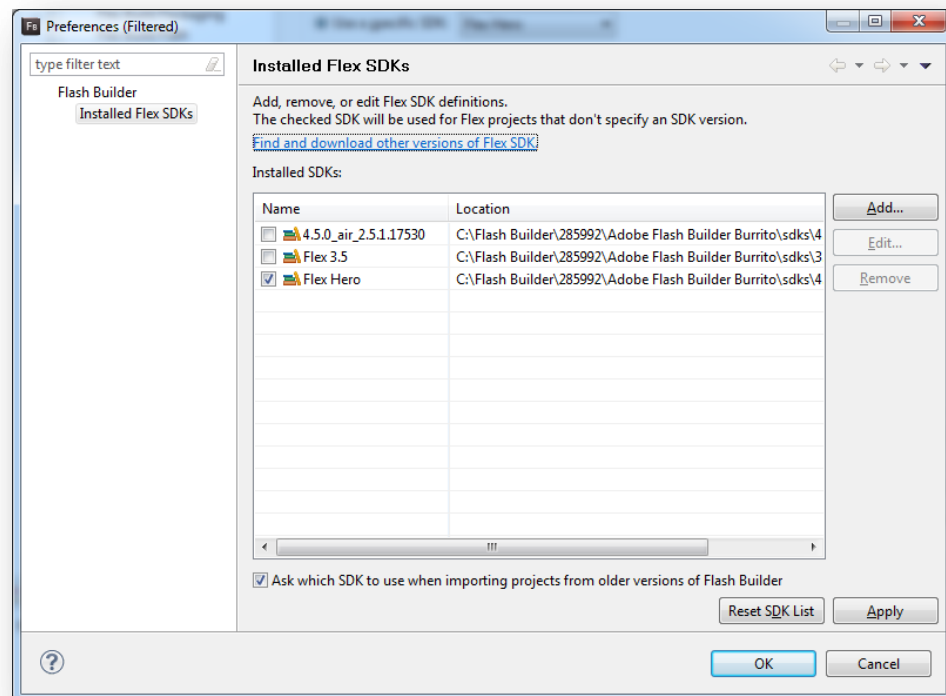


Figure 10 - Flash Builder Preferences - All SDKs

*Note:* This window will show you all the SDKs Flash Builder is aware of, and allow you to choose the default for all new projects.

4. Click “Add.”
5. Click “Browse,” and browse to the “sdks” directory.
6. Select the directory you created in the previous steps. E.g., if you installed Flex 4.5.0, select the folder you created labeled “4.5.0.” Click “OK.”
7. Enter a name for this SDK in the “Flex SDK name” field. Click “OK.”
8. If you wish to make this SDK the default for all future projects you create, click the appropriate checkbox that appears in the list of SDKs.
9. Click “OK” to close the Flash Builder preferences window
10. You should once again be looking at the project properties window. Select “Use a specific SDK” and choose the SDK you just added from the drop down list.
11. Click “OK.”

Your project should now automatically recompile using the new SDK. If you wish to modify any other existing projects, you just have to select the SDK from the drop down list in the project properties window, as you’ve already pointed Flash Builder to the new SDK directory.

## Additional Resources

### Tutorials/Guides

- You can find more information, including additional documents and tutorials, on the Flex “Hero” page on Adobe Labs: <http://labs.adobe.com/technologies/flex/mobile/> or <http://goo.gl/eV2x>

### Sample App/Code

- SurveyApe – an application written by the author, which demonstrates the basics of mobile applications, as well as Database access and camera APIs. This application will be available on Adobe Labs for MAX, or you can ask a Device Lab coordinator (or myself, if I’m there at the time): <http://labs.adobe.com>

## Data Centric Design

- “Data-centric development with Flash Builder 4,” Sunil Bannur. March 8<sup>th</sup>, 2010. [http://www.adobe.com/devnet/flex/articles/datacentric\\_development.html](http://www.adobe.com/devnet/flex/articles/datacentric_development.html)  
or <http://goo.gl/IY25>

## Packaging for Release

- “Digitally signing Adobe AIR applications,” Todd Prekaski. 2010. Accessed October 8<sup>th</sup>, 2010. [http://www.adobe.com/devnet/air/articles/signing\\_air\\_applications.html](http://www.adobe.com/devnet/air/articles/signing_air_applications.html)  
or <http://goo.gl/GRZC>

## Android SDK

- “Download the Android SDK,” Android.com. Accessed October 8<sup>th</sup>, 2010. <http://developer.android.com/sdk/index.html>  
or <http://goo.gl/005L>

## SDKs

### Downloading

- AIR SDK <http://www.adobe.com/cfusion/entitlement/index.cfm?e=airsdk>  
or <http://goo.gl/SVl6>
- Flex SDK <http://opensource.adobe.com/wiki/display/flexsdk/Flex+4>  
or <http://goo.gl/Ymob>

### Installing

- “Installing the Adobe AIR 2 Beta SDK on OS X,” Michael Christoff. December 1<sup>st</sup>, 2009. <http://mchristoff.com/2009/12/installing-the-adobe-air-2-beta-sdk-on-os-x/>  
or <http://goo.gl/MfhK>
- “Using AIR SDK with Flash Builder and Flash Professional,” Adobe.com. June 9<sup>th</sup>, 2010. [http://www.adobe.com/support/documentation/en/air/2/releasenotes\\_developers.html#h](http://www.adobe.com/support/documentation/en/air/2/releasenotes_developers.html#h)  
or <http://goo.gl/Umkn>
- “Downloading and installing Flex SDK builds from opensource.adobe.com,” Peter Dehaan. August 1<sup>st</sup>, 2008. <http://blog.flexexamples.com/2008/08/01/downloading-and-installing-flex-sdk-builds-from-opensourceadobecom/>  
or <http://goo.gl/yqgD>

## Building

### Conditional Compilation

- “How to create conditional compilation definitions (conditional compile blocks),” Andrei Lonescu. April 3<sup>rd</sup>, 2010.  
<http://www.flexer.info/2010/03/04/how-to-create-conditional-compilation-definitions-conditional-compile-blocks/>  
or <http://goo.gl/IGZv>