shortcut

Your Short Cut to Knowledge

Adobe® Apollo in Flight

Michael Givens

SAMS www.samspublishing.com

Contents at a Glance

Visit us on the Web: www.samspublishing.com

ISBN-10: 0-7686-7666-5

ISBN-13: 978-0-7686-7666-2

First release June 2007

ABOUT THE AUTHOR

Michael Givens is the CTO of U Saw It Enterprises, a web technology consulting firm based in Marietta, Georgia. He is an Adobe Community Expert and an Adobe Corporate Champion known to share his experience and evangelism of all things Adobe. Certified in ColdFusion 5 and as an advanced CFMX developer, he has been using ColdFusion since the days of Allaire Spectra. He has written articles for the *ColdFusion Developer's Journal* and the *Web Developer's & Designer's Journal* and blogs regularly at http://www.flexination.info.

Chapter 1

Welcome to Apollo

What Is Apollo?

Apollo is the code name for Adobe's innovative new technology that consists of a cross-platform system runtime that enables developers to leverage their existing web development skills, such as Flex, Flash, HTML, JavaScript, and AJAX, to create rich desktop applications that can be deployed on Windows and Mac operating systems. A Linux version is slated to follow shortly after the 1.0 release. The Apollo technology enables developers to take their established web-based projects, and all the benefits that are inherent in deployment on the web, and further enhance the feature set by adding interaction with the desktop's local resources. Apollo enables such features as file I/O, online/offline capabilities, native windowing, network API support, drag-and-drop support, interaction with the Clipboard, and desktop shortcuts for existing web applications redeployed on the desktop or new hybrid web-enabled desktop applications. By writing a bit of additional code, the developer can have his desktop application detect a loss of network connectivity and smoothly transition to offline-only features. Imagine an MP3 desktop application that has access to both an online music library and a local music repository. When the user is online, the MP3 can play from both music sources, but even when the user is offline, the local music store is still available—loss of connectivity does not break the application. Figure 1.1 shows a sample Apollo application that includes a network connectivity visual indicator (a smiley icon smiles while connected and frowns when disconnected).

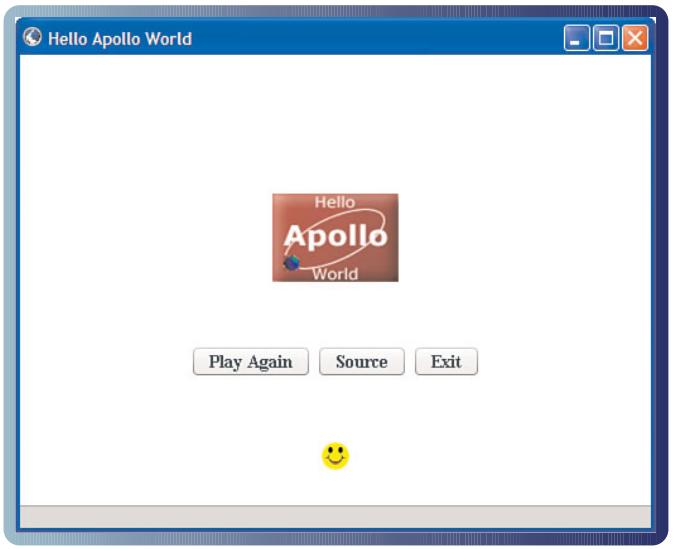


FIGURE 1.1
A sample Apollo application.

NOTE

The Apollo runtime requires only a one-time download and installation.

The Sum of the Parts

Apollo consists of the Apollo runtime and the Apollo SDK. The runtime is what you and users of your Apollo applications need to install to run these new desktop applications. The SDK provides the alpha development SDK for Windows and Macintosh and contains the Apollo framework and command-line tools. For Eclipse IDE users, Flex and Apollo plug-ins are available free. For Flex developers using Flex Builder 2.0.1, there is a separate extension—the Apollo Extensions for Flex

Builder, which integrates nicely in the IDE. Documentation and sample applications are available as separate downloads. Both the Apollo runtime and the SDK are free and currently run on Windows and Mac OS X. According to Adobe, the target date for the Apollo 1.0 release is the second half of 2007. The current alpha of Apollo is available at http://labs.adobe.com/technologies/apollo/.

NOTE

The Apollo Extensions for Flex Builder require the Flex Builder 2.0.1.

Apollo Differentiators

Apollo is the next generation of desktop development. The elusive paradigm of write once and run everywhere is targeted as Apollo applications run on Windows and Mac OS X systems. Apollo is different from competing technologies in the way in which applications are developed. Aside from learning the Apollo APIs, developing an Apollo application utilizes the web development skill set in which a developer already has experience. There is no need to rush out and learn Windows or Mac development. Applications can be built with Flex, Flash, and HTML/JavaScript or a mix of these web technologies. Although not currently available in the alpha version, PDF technology will also be supported as a development tool (Adobe Acrobat Reader must be installed on the client machine to leverage these PDF features). No specific IDE is necessary to build an Apollo application, although the Eclipse plus Flex plug-in and standalone Flex Builder are certainly attractive options. Existing web applications that could benefit from a tighter integration with the desktop are ideal candidates for building Apollo-based versions. Apollo is a lightweight runtime that, according to Adobe, is targeted at 5–9MB.

The Apollo runtime distribution channels will include the Adobe website (similar to the current Flash Player distribution), smart distribution within an Apollo application installer (for example, developer-created installers can test for the presence of the Apollo runtime and install it only if it is missing), as well as other methods that Adobe is currently exploring. Apollo applications are installed in the same manner in which users are already accustomed. The Apollo application is installed on the user's desktop by downloading an application's installer (currently an AIR file) and running it. The application's desktop shortcut defaults to a generic OS-specific icon, or the developer can create his own custom shortcut icon (see Figure 1.2 and Figure 1.3).

FIGURE 1.2
Default Apollo application shortcut icon.

FIGURE 1.3
Custom Apollo application shortcut icon.

TIP

For the curious, an AIR file can be opened with a zip archive utility.

Chapter 2

Preparing to Launch—Installation

Apollo Runtime Installation

The Apollo runtime alpha installation is as easy and painless as it could be. The runtime is required to run any Apollo applications. Navigate to the download location, http://labs.adobe.com/technologies/apollo/, and download the installer file. On the current labs.adobe.com site, you will need to register for a free Adobe membership ID or sign in with your existing credentials, and then accept an Adobe Preview Release Software End User License Agreement. For Windows, the runtime installer will be a file named `apollo_win_alpha1_031907.msi`. For the Mac, the file is `apollo_mac_alpha1_031907.dmg`. The Apollo alpha was released on March 19, 2007, which explains the date in the filenames.

NOTE

The filenames are subject to change as the Apollo software matures.

Windows Installation

1. Double-click the `apollo_win_alpha1_031907.msi` file and the installation will begin.
2. When the installation is complete, click the OK button.

Mac OS X Installation

1. Double-click `apollo_mac_alpha1_031907.dmg`. An Adobe Apollo window displays.
2. Double-click the `Adobe Apollo.pkg` file. An Install Adobe Apollo window displays.
3. Click the Continue button. A Select a Destination page appears.
4. Select the destination volume and click the Continue button. An Easy Install button is displayed.
5. Click the Install button.
6. After the installation is complete, click the Close button.

Setting Up the Apollo SDK

An alternative to using Flex Builder for Apollo development is to download and install the Apollo SDK and to install either a Java Runtime Environment (JRE) or a Java Development Kit (JDK).

NOTE

The Apollo SDK is available on the same download page as the Apollo runtime and contains the following resources: `apolloframework.swc` and `apolloglobal.swc` (the Apollo core libraries and framework classes used to generate the SWF file inherent in an Apollo application), Apollo Debug Launcher (ADL) (used to test Apollo applications without having to install them first) Apollo Developer Tool (ADT) (used to package Apollo applications for distribution) and `application.xml` and `application.xsd` (an application settings file and the schema for it).

Apollo SDK Windows Installation

1. Double-click the `apollo_sdk_alpha1_031907.zip` file to extract the files to `C:\Program Files\Apollo_SDK`.
2. From the Windows Control Panel, launch the System control panel and select the Advanced tab.
3. Select the Environment Variables button and add the following to the Path variable: `C:\Program Files\Apollo_SDK\bin`.

CAUTION

Take care to add a semicolon to separate the new variable from the existing Path variables (for example, ...; C:\Program Files\Apollo_SDK\bin).

Apollo SDK Mac OS X Installation

1. Create an `Apollo_SDK` subdirectory in the `/Applications` directory at the root of the system.
2. Double-click the `apollo_sdk_alpha1_031907.zip` file and extract the files to the new `Apollo_SDK` subdirectory created in step 1.
3. Double-click the `Adobe Apollo.dmg` file in the `/Applications/Apollo_SDK/runtime` directory.
4. Drag the `Adobe Apollo.framework` directory contained in the `Adobe Apollo.dmg` file to the `/Applications/Apollo_SDK/runtime` directory (in the Finder) to install the Apollo runtime used by the Apollo Debug Launcher (ADL).

5. Ensure that the Apollo command-line tools are added to the system path by opening a new window in the Mac OS X Terminal Application.
6. Navigate to the user directory and type **`open -e .profile`** in the terminal window.
7. Enter or add to the PATH the following: `export PATH=$PATH: /Applications/Apollo_SDK/bin/.`
8. Save the changes to the `.profile` file.
9. Type **`source .profile`** to reload the changes.

See the help topic "Apollo:Documentation:Adding the Apollo tools to your class path" for more information: http://labs.adobe.com/wiki/index.php/Apollo:Documentation:Adding_the_Apollo_tools_to_your_class_path.

Apollo Flex Builder Extensions Installation

With the runtime installed, a developer of Apollo applications may choose from several web technologies, including Flex. For Flex developers using Flex Builder 2.0.1, Adobe has included the Apollo Flex Builder Extensions on the labs.adobe.com site.

NOTE

The Apollo Extensions for Flex Builder are available on the same download page as the Apollo runtime.

The extensions extend Flex Builder 2.0.1 to enable development of Apollo applications. Developers can also combine the free Flex SDK with the Apollo SDK to create Apollo applications. For Windows, the Flex Builder Extensions download is a file named `fb_apollo_extensions_win_alpha1_031907`. For Mac, the file is `fb_apollo_extensions_mac_alpha1_031907.dmg`.

Windows Installation

1. Double-click the `fb_apollo_extensions_win_alpha1_031907` file and installation begins (see Figure 2.1).
2. Step through the installation wizard, and click the OK button when the installation is complete (see Figure 2.2).

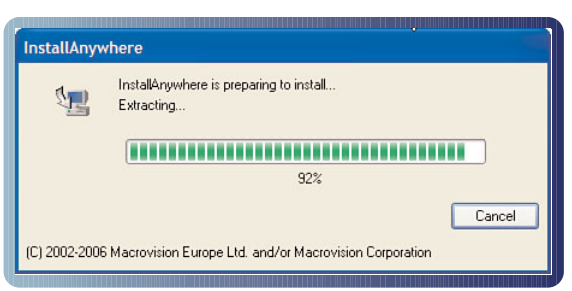


FIGURE 2.1
Installing the Flex Builder Extensions.

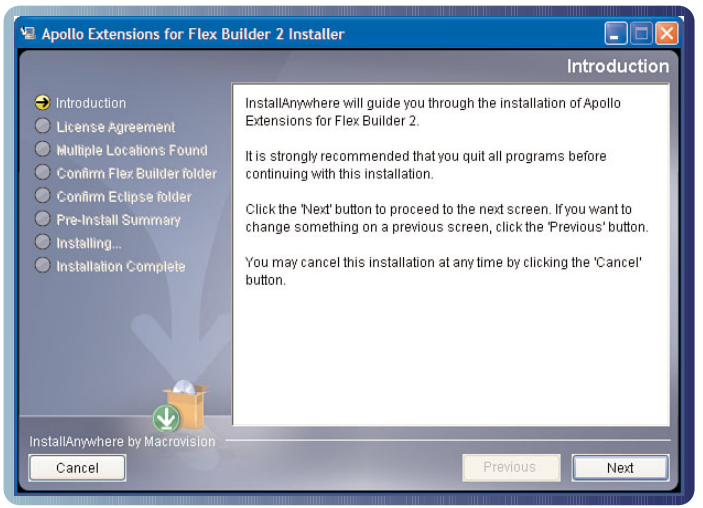


FIGURE 2.2
The Flex Builder Extensions Installation Wizard.

With the Flex Builder Extensions installed, a new menu choice will be available for creating Apollo projects (see Figure 2.3).

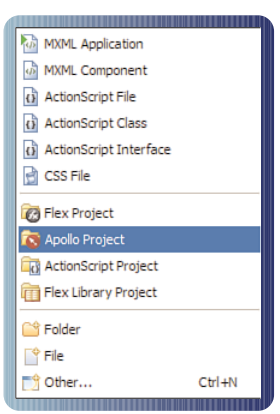


FIGURE 2.3
Creating a new Apollo project in Flex Builder.

Chapter 3

Your First Apollo Mission

Building Your First Apollo Application

You can build your first Apollo desktop application leveraging Flex Builder 2.0.1, the Apollo Flex Builder Extensions, and, of course, the Apollo runtime. Apollo development involves a workflow that is common to all Apollo applications and includes creating an application descriptor XML file (*your_app_name*`-app.xml`), writing and testing your application-specific code, preparing the Apollo AIR file, and distributing the Apollo application. In this chapter, you will see a suggested systematic example of creating an Apollo application. You will create a Hello Apollo World application using Flex Builder 2.0.1 that has the Apollo Flex Builder Extensions installed.

NOTE

The author actually spent several hours developing this example with the goal of demonstrating a higher-level application than the typical Hello World example often seen in other publications. So don't skip it thinking it will be too basic!

If you do not already have Flex Builder 2.0.1, download it from the Adobe website. A free 30-day trial is available at http://www.adobe.com/products/flex/flexbuilder/. Follow the installation instructions included on the Adobe site. After this is complete, follow the instructions for installing the Apollo Flex Builder Extensions given previously in the section "Apollo Flex Builder Extensions Installation" in Chapter 2, "Preparing to Launch—Installation."

Creating a New Apollo Project

Fire up Flex Builder and follow these steps:

1. Choose File, New, Apollo Project from the Flex Builder menu, as shown in Figure 3.1.

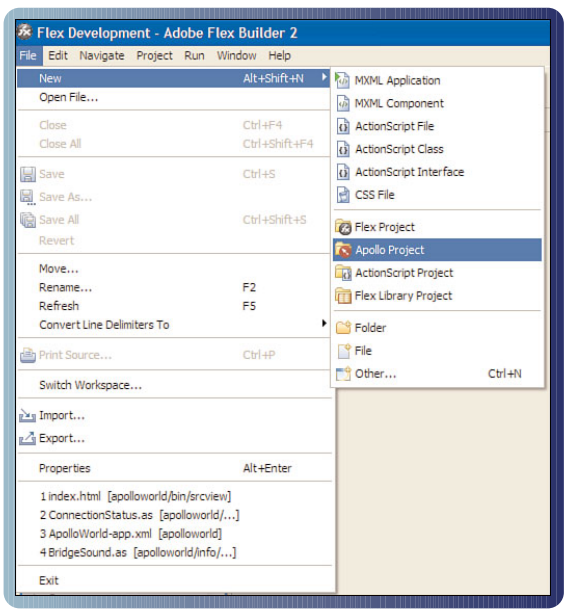


FIGURE 3.1
Initiating a new Apollo Project Wizard.

FIGURE 3.2
How will your Apollo application access data?

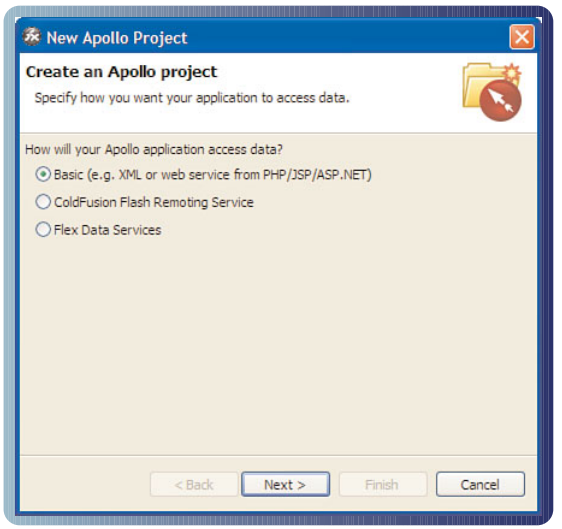
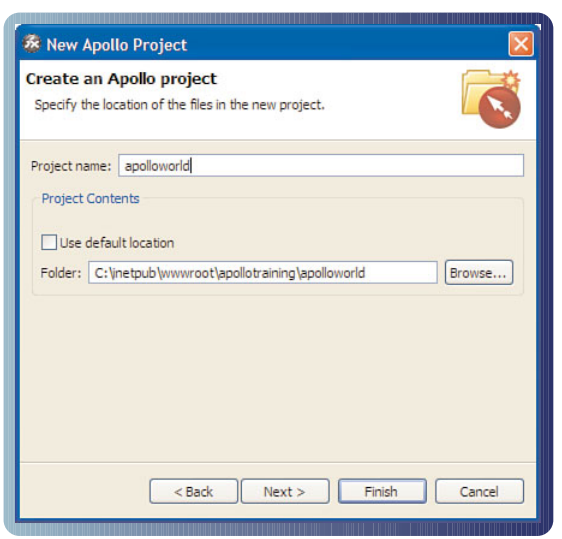


FIGURE 3.3
Enter the project name and location.



2. Choose the Basic radio button for How Will Your Apollo Application Access Data? choice. Click the Next button (see Figure 3.2).
3. Next, specify the name of the Apollo project and location of the files that you will create for your application. Click the Next button (see Figure 3.3).

4. Enter the name you want to use for your main Application file. For this example, use `ApolloWorld.mxml`. The default Output folder, `bin`, can be used. Click the Next button (see Figure 3.4).

5. Enter the application descriptor XML settings for ID, Name, Publisher, Description, and Copyright (at a bare minimum you should enter an ID and Name). Click the Finish button. The initial code is generated automatically (see Figure 3.5).

FIGURE 3.4
Enter the name of the main MXML file and Output folder.

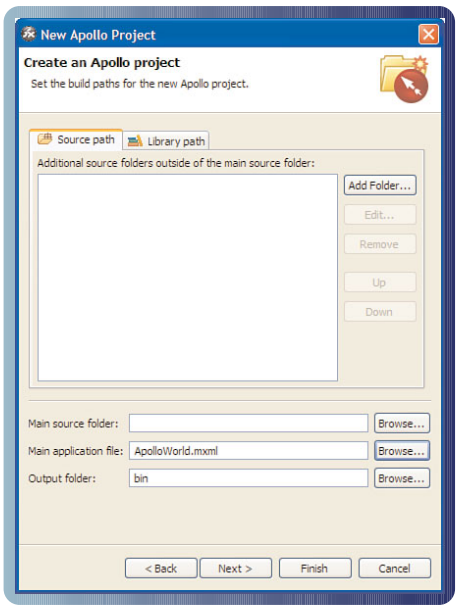
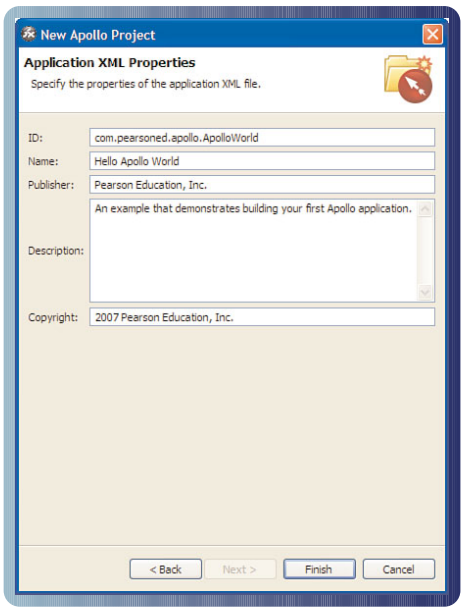


FIGURE 3.5
Enter the descriptor XML settings.



NOTE

The ID should uniquely identify your application. It is common practice to use the URL (in reverse) of the deployment site in the ID (for example, com.pearsoned.apollo.ApolloWorld).

6. The initial `ApolloWorld.mxml` skeleton is shown in Listing 3.1.

LISTING 3.1 `ApolloWorld.mxml` **Skeleton**

```
<?xml version="1.0" encoding="utf-8"?>
<mx:ApolloApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">

</mx:ApolloApplication>
```

7. You can run this application as an initial test of the Apollo runtime. Of course, it will not do much as is (see Figure 3.6).

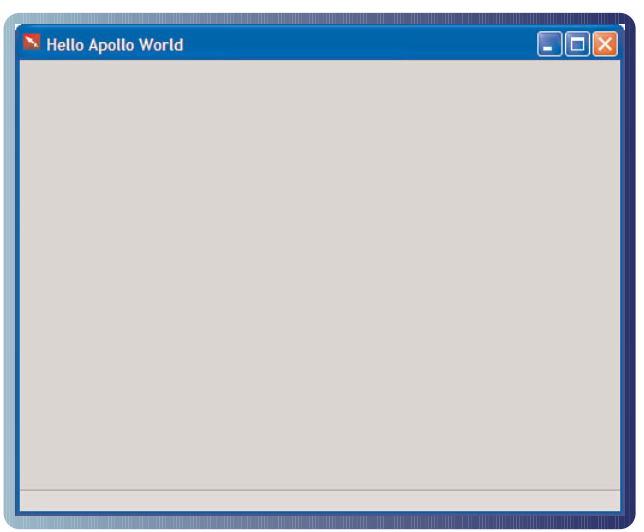


FIGURE 3.6
An empty Apollo application.

Writing the Apollo Application Code

1. The Apollo application you will build is shown in Figure 3.7.

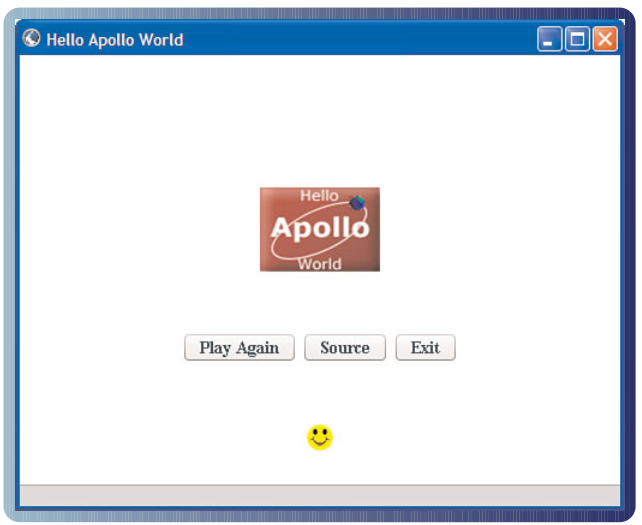


FIGURE 3.7
The Apollo application you will build.

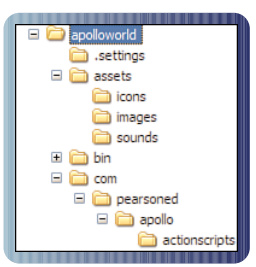


FIGURE 3.8
Apollo application directory layout.

2. Download the source code, `apolloworld.zip`, from the following URL: http://labs.insideflex.com/apollotraining/apolloworld/bin/srcview/ApolloWorld.zip.

3. Extract the zip (maintaining the folder structure) to your ApolloWorld application folder created in step 3 on page 16. You should end up with a directory structure like the one shown in Figure 3.8.

The full code for `ApolloWorld.mxml` is shown in Listing 3.2 (note the code comments are included in the code for your reference).

LISTING 3.2 ApolloWorld.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:ApolloApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    title="Hello Apollo World"
    creationComplete="initApp()"
    layout="vertical"
    backgroundColor="#FFFFFF" width="100%" height="100%">

    <mx:Script>
    <![CDATA[
    // imports needed for this application
    import flash.utils.Timer;
    import flash.events.TimerEvent;
    import com.pearsoned.apollo.actionscripts.BridgeSound;
    import com.pearsoned.apollo.actionscripts.ConnectionStatus;

    // embed images and declare persistent image classes
    [Embed(source="assets/images/helloapollologo.png")]
    [Bindable] private var imgHelloApollo:Class;
    [Embed(source="assets/images/helloapolloworld1.png")]
    [Bindable] private var imgHelloApolloWorld:Class;
    [Embed(source="assets/images/helloapolloworld2.png")]
    [Bindable] private var imgHelloApolloWorld2:Class;
    [Embed(source="assets/images/helloapolloworld3.png")]
    [Bindable] private var imgHelloApolloWorld3:Class;
    [Embed(source="assets/images/helloapolloworld4.png")]
```

```
[Bindable] private var imgHelloApolloWorld4:Class;
// declare a persistent timer (0.2 seconds)
[Bindable] private var myTimer:Timer = new Timer(200, 0);
// declare a persistent counter
[Bindable] private var counter:int = 0;
private var mySound:BridgeSound = new BridgeSound();
// declare a one-time use timer (12 seconds)
private var myTimer2:Timer = new Timer(12000, 1);
// declare a connection to test the connection status
private var connection:ConnectionStatus = new ConnectionStatus();
// declare a persistent public boolean enabled value for the 'Source' button
[Bindable] public var blnConnected:Boolean = true;
// declare a persistent public boolean enabled value for the 'Play Again' button
[Bindable] public var blnPlayEnabled:Boolean = false;

// initial function called in the ApolloApplication creationComplete method
private function initApp():void {
    // add a NETWORK_CHANGE event listener: Listens to connectivity status
    Shell.shell.addEventListener(Event.NETWORK_CHANGE,connection.onConnectionChange);
    // checks the connectivity
    connection.checkConnection();
    /* add a timer event listener: Listens for 0.2 second intervals;
    onTimer() function is called every 0.2 seconds */
    myTimer.addEventListener("timer", onTimer);
    // calls the startHelloTimer() function
    startHelloTimer();
```

```
            // sets the URL for the sound effect
            mySound.url =
"http://labs.insideflex.com/apollotraining/apolloworld/assets/sounds/STbridge.MP3";
            // calls the getSound() method that plays the STbridge.MP3
            mySound.getSound();
            /* add a timer event listener: Listens for a 12 second interval;
            onTimer2() function is called once after 12 seconds */
            myTimer2.addEventListener("timer", onTimer2);
            // start the 12 second timer
            myTimer2.start();
        }
            // soundCompleteHandler() function
                private function soundCompleteHandler(event:Event):void {
                  /* set the boolean, blnPlayEnabled, to true;
                  this is used to re-enable the 'Play Again' button */
                      blnPlayEnabled = true;
                }
        // this function is for re-playing the sound effects
        private function playSoundEffects():void {
            // sets the URL for the sound effect
            mySound.url =
"http://labs.insideflex.com/apollotraining/apolloworld/assets/sounds/STbridge.MP3";
            // calls the getSound() method that plays the STbridge.MP3
            mySound.getSound();
                /* add a SOUND_COMPLETE event listener: Listens for the end of the sound effect;
                soundCompleteHandler() function is called once the listener event occurs*/
```

```
            mySound.song.addEventListener(Event.SOUND_COMPLETE, soundCompleteHandler);
        /* add a timer event listener: Listens for a 12 second interval;
        onTimer2() function is called once after 12 seconds */
        myTimer2.addEventListener("timer", onTimer2);
        // start the 12 second timer
        myTimer2.start();
    }
    // this function starts the 0.2 second timer
    private function startHelloTimer():void {
        // start the 0.2 second timer
        myTimer.start();
    }
    // this function cycles through the animation images for the revolving earth effect
    private function onTimer(event:TimerEvent):void {
        // conditional logic for rotating through 5 possible images
        switch (counter) {
            case 1:
                imgApolloLogo.source = imgHelloApolloWorld;
                break;
            case 2:
                imgApolloLogo.source = imgHelloApolloWorld2;
                break;
            case 3:
                imgApolloLogo.source = imgHelloApolloWorld3;
                break;
            case 4:
```

```
                imgApolloLogo.source = imgHelloApolloWorld4;
                counter = 0;
                break;
            default:
                imgApolloLogo.source = imgHelloApollo;
                break;
        }
        // increment counter
        counter++;
    }
    // this function is the Timer2 handler
    private function onTimer2(event:TimerEvent):void {
        // sets the URL for the sound effect
        mySound.url ="http://labs.insideflex.com/apollotraining/apolloworld/assets/sounds/
         ➥fascinating.MP3";
        // calls the getSound() method that plays the fascinating.MP3
        mySound.getSound();
    }

    private function goThere(sURL:String):void {
        // declare a URLRequest variable that accepts a URL (sURL)
           var u:URLRequest = new URLRequest(sURL);
           // pop a new window using the URL passed in
        navigateToURL(u,"_blank");
    }
    ]]>
    </mx:Script>
```

```
<mx:Style>
    <!-- style for the toolTip displayed when mousing over Flex components -->
    ToolTip { font-family: "Verdanna"; font-size: 14; font-weight: "normal";
    background-color: "0x821313"; color: "0xFFFFFF"; }
</mx:Style>

<!-- Spacer pushes the content 80 pixels downward —>
<mx:Spacer height="80"/>
<!-- Initially displays the image of the apollologo.png and then under AS3 function
control, rotates in images -->
<mx:Image id="imgApolloLogo"
➥source="http://labs.insideflex.com/apollotraining/apolloworld/assets/images/
apollologo.png"/>
<mx:Spacer height="40"/>
<!-- HBox displays content in a horizontal layout -->
<mx:HBox>
    <!-- Button controls playing the sound effects -->
    <mx:Button id="btnPlayAgain" label="Play Again"
    ➥click="playSoundEffects();blnPlayEnabled=false" fontFamily="Verdanna"
        fontSize="14" fontWeight="bold"
    ➥toolTip="{(blnPlayEnabled)?'Click to play the
    sound effects again...':'Button is
    ➥currently inactive...'}" enabled="{blnPlayEnabled}"/>
    <!-- Button controls opening a new browser window to view the source code -->
    <mx:Button id="btnViewSource" label="Source"
    ➥click="goThere('http://labs.insideflex.com/apollotraining/apolloworld/bin/srcview/
    ➥index.html')" fontFamily="Verdanna"
```

```
            fontSize="14" fontWeight="bold"
            ➥toolTip="{(blnConnected)?'Click to view the source code...':'Offline mode -
            button is currently inactive...'}"/>
        <!-- Button exits the application -->
        <mx:Button id="btnExit" label="Exit" click="window.close()"
        ➥fontFamily="Verdanna"
            fontSize="14" fontWeight="bold" toolTip="Click to exit the application..."/>
    </mx:HBox>
    <mx:Spacer height="40"/>
    <!-- Displays the image that corresponds to connectivity (smile or frown) —>
    <mx:Image id="statusPic" width="24" height="24"/>
</mx:ApolloApplication>
```

NOTE

For the curious, the `[Bindable]` metadata tag in the `<MX:Script>` tag is used to make a variable usable as the source for a data binding expression. For more information about data binding, go to http://livedocs.adobe.com/flex/2/docs/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00001038.html.

The ApolloWorld application includes two ActionScript 3 classes, `BridgeSound.as` and `ConnectionStatus.as`. The `BridgeSound.as` class is leveraged to play the sound effect and is shown in Listing 3.3.

LISTING 3.3 BridgeSound.as

```
package com.pearsoned.apollo.actionscripts
{
    // imports needed for this AS class
    import flash.display.Sprite;
    import flash.events.*;
    import flash.media.Sound;
    import flash.media.SoundChannel;
    import flash.net.URLRequest;
    import mx.core.Application;

    public class BridgeSound extends Sprite {
    // declare a persistent, public variable to hold the sound effect's URL
        [Bindable] public var url:String;
    // declare a variable to hold a SoundChannel class
        public var song:SoundChannel;

        public function getSound():void {
        // declare a URLRequest variable that accepts a URL (url)
        var request:URLRequest = new URLRequest(url);
        // declare a variable to hold a soundFactory class
        var soundFactory:Sound = new Sound();
        /* add a COMPLETE event listener: Listens for the sound effect load completion;
        completeHandler() function is called when the listener event occurs */
        soundFactory.addEventListener(Event.COMPLETE, completeHandler);
        /* add an ID3 event listener: Listens for ID3 availability;
```

```
id3Handler() function is called when the listener event occurs */
soundFactory.addEventListener(Event.ID3, id3Handler);
/* add an IO_ERROR event listener: Listens for any I/O errors;
ioErrorHandler() function is called if the listener event occurs */
soundFactory.addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
/* add a PROGRESS event listener: Listens for the progression of the sound effect;
progressHandler() function is called when the listener event occurs */
soundFactory.addEventListener(ProgressEvent.PROGRESS, progressHandler);
// load the sound effect
soundFactory.load(request);
// play the sound effect
song = soundFactory.play();
}

private function completeHandler(event:Event):void {
    trace("completeHandler: " + event);
 }

private function id3Handler(event:Event):void {
    trace("id3Handler: " + event);
}

private function ioErrorHandler(event:Event):void {
    trace("ioErrorHandler: " + event);
}
```

```
        private function progressHandler(event:ProgressEvent):void {
            trace("progressHandler: " + event);
        }

    }
}
```

The ConnectionStatus.as class is used to monitor the network or Internet connectivity and is shown in Listing 3.4.

LISTING 3.4 ConnectionStatus.as

```
package com.pearsoned.apollo.actionscripts
{
    // imports needed for this AS class
    import flash.events.NetStatusEvent;
    import flash.system.System;
    import flash.net.URLLoader;
    import flash.net.URLRequest;
    import mx.managers.CursorManager;
    import flash.events.HTTPStatusEvent;
    import flash.events.IOErrorEvent;
    import mx.core.Application;

    public class ConnectionStatus
    {
        // embed images and declare persistent image classes
        [Embed(source="../../../../assets/images/smile.png")]
```

```
[Bindable] public var imgConnected:Class;
[Embed(source="../../../../assets/images/frown.png")]
[Bindable] public var imgDisconnected:Class;

// checks the connectivity by accessing the URL, http://labs.insideflex.com
public function checkConnection():void {
    // triggers the display of a watch glass or busy cursor
    CursorManager.setBusyCursor();
    // declare a URLRequest variable
var headRequest:URLRequest = new URLRequest();
// set the URLRequest method to HEAD
headRequest.method = "HEAD";
// set the URL
headRequest.url = "http://labs.insideflex.com";
// declare a URLLoader variable and pass the URLRequest
    var response:URLLoader = new URLLoader(headRequest);
    /* add an HTTP_STATUS event listener: Listens for the HTTP_STATUS;
    statusChanged() function is called when the listener event occurs */
    response.addEventListener(HTTPStatusEvent.HTTP_STATUS, statusChanged);
    /* add an IO_ERROR event listener: Listens for an I/O connectivity issue;
    onError() function is called if the listener event occurs */
response.addEventListener(IOErrorEvent.IO_ERROR, onError);
}

// handler function for networkChangeEvent events
public function onConnectionChange(networkChangeEvent:Event):void {
    // check the connection
```

```
checkConnection();
}
// handler function for HTTPStatusEvent events
private function statusChanged(status:HTTPStatusEvent):void {
    // remove the watch glass or busy cursor
    CursorManager.removeBusyCursor();
    // conditional to check the status
    if (status.status == 0) {
        // change the image, statusPic, to the frown.png
        mx.core.Application.application.statusPic.source = imgDisconnected;
        // change the tooltip for the image, statusPic
        mx.core.Application.application.statusPic.toolTip =
        ➥"Offline and not connected to the Internet...";
        // toggle buttons' availability to unavailable
        mx.core.Application.application.btnPlayAgain.enabled = false;
        mx.core.Application.application.blnConnected = false;
        mx.core.Application.application.blnPlayEnabled = false;
        mx.core.Application.application.btnViewSource.enabled = false;
    } else {
        // change the image, statusPic, to the smile.png
        mx.core.Application.application.statusPic.source = imgConnected;
        // change the tooltip for the image, statusPic
        mx.core.Application.application.statusPic.toolTip =
        ➥"Online and connected to the Internet...";
        // toggle button availability to available or unavailable if sound effect is
        ➥already playing
        if (mx.core.Application.application.blnPlayEnabled) {
```

```
                    mx.core.Application.application.btnPlayAgain.enabled = true;
                } else {
                    mx.core.Application.application.btnPlayAgain.enabled = false;
                }
                // toggle buttons' availability to available
                mx.core.Application.application.blnConnected = true;
                mx.core.Application.application.blnPlayEnabled = true;
                mx.core.Application.application.btnViewSource.enabled = true;
            }
        }
        // handler function for IOErrorEvent events
        private function onError(error:IOErrorEvent):void {
            // remove the watch glass or busy cursor
            CursorManager.removeBusyCursor();
            // change the image, statusPic, to the frown.png
            mx.core.Application.application.statusPic.source = imgDisconnected;
                // change the tooltip for the image, statusPic
            mx.core.Application.application.statusPic.toolTip = "Offline and not connected to
            ➥the Internet...";
            // toggle buttons' availability to unavailable
            mx.core.Application.application.btnPlayAgain.enabled = false;
            mx.core.Application.application.blnConnected = false;
            mx.core.Application.application.blnPlayEnabled = false;
        }
    }

}
```

Modifying the Descriptor XML File

Although the Apollo Project Wizard creates the initial Apollo Descriptor file, you might want to modify it to include custom shortcut icons.

CAUTION

In the current Apollo Flex Builder Extensions, the Descriptor XML file that is automatically created contains five non-standard hyphens (— instead of -). There are some cases where this can cause issues. For safety, you should ensure that all — are converted to -.

You can also make changes to the application's ID, name, description, system chrome, or copyright information.

TIP

Take care when modifying the descriptor file. Because it is XML, it must be well formed or you will get an error when you try to export the application's AIR file.

You can add your own custom shortcut icon by following these steps:

1. Create four images of varying size (use 16×16, 32×32, 48×48, and 128×128) of the image you want to use for the shortcut.
2. Add the icon-related tags near the bottom of the descriptor file (see Listing 3.5).

LISTING 3.5 Icon Block

```
<icon>
    <image16x16>assets/icons/ApolloApp_16.png</image16x16>
    <image32x32>assets/icons/ApolloApp_32.png</image32x32>
    <image48x48>assets/icons/ApolloApp_48.png</image48x48>
    <image128x128>assets/icons/ApolloApp_128.png</image128x128>
</icon>
```

Testing Your Apollo Application

1. To test your Apollo application from the Flex Builder IDE, click the green Run icon (see Figure 3.9).

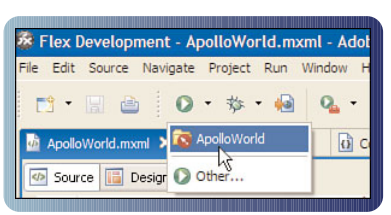


FIGURE 3.9
Run the Apollo application.

2. Iterate between developing the code and testing the application until you achieve the final application that you want to deploy.

Exporting Your Apollo Application AIR File

1. Begin the process by selecting File, Export from the Flex Builder menu, as shown in Figure 3.10.
2. Select the export destination, Deployable AIR File, as shown in Figure 3.11.

FIGURE 3.10
Initiate the AIR file creation wizard.

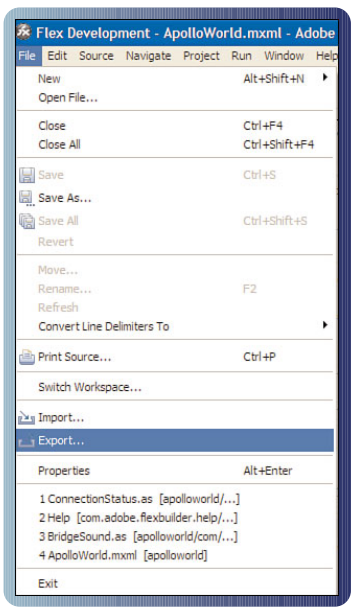
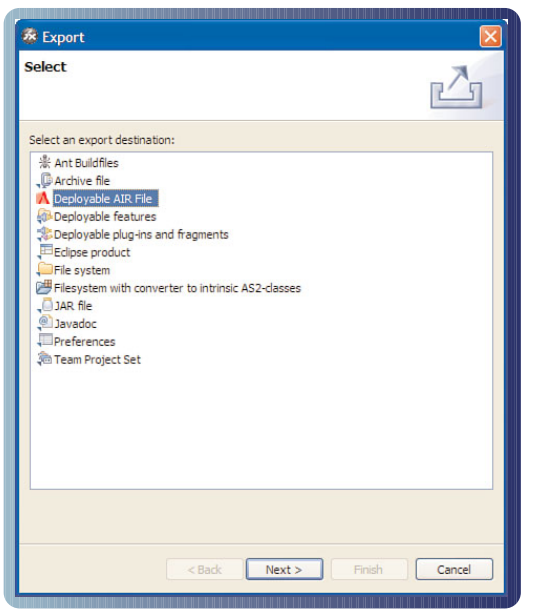


FIGURE 3.11
Selecting Deployable AIR File.



3. Select the files to include in the AIR file and browse to the directory where the AIR file will be created (see Figure 3.12).

4. Click Finish to create the AIR file. The exported AIR file is shown in Figure 3.13.

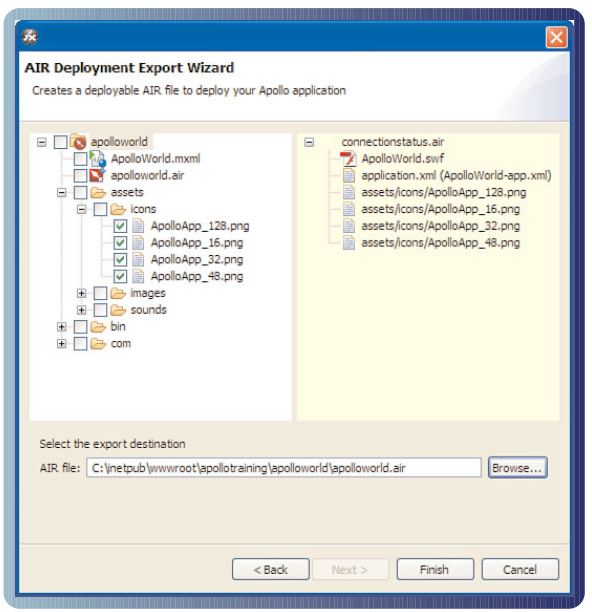


FIGURE 3.12
Select the files to include and the export destination.

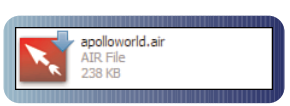


FIGURE 3.13
AIR shortcut icon.

NOTE

If you do not use the Flex Builder IDE to export your AIR file, an alternative is to use the Apollo Developer Tool (ADT) that is included with the Apollo SDK. From a command prompt (Windows) or terminal application (Mac), type the following and press the Enter (or Return) key:

```
adt -package ApolloWorld.air
ApolloWorld-app.xml ApolloWorld.swf
```

This creates the `ApolloWorld.air` file.

5. You can test your AIR file installer by double-clicking it and stepping through the installation screens (see Figures 3.14, 3.15, and 3.16). The Hello World desktop shortcut is shown in Figure 3.17.

NOTE

During the installation of an alpha Apollo application, the exported AIR file displays publisher and system access warnings. Publisher registration, similar to code-signing technology, should be worked out by the Apollo 1.0 release and should improve the user installation experience.

FIGURE 3.14
Install the Hello World application.

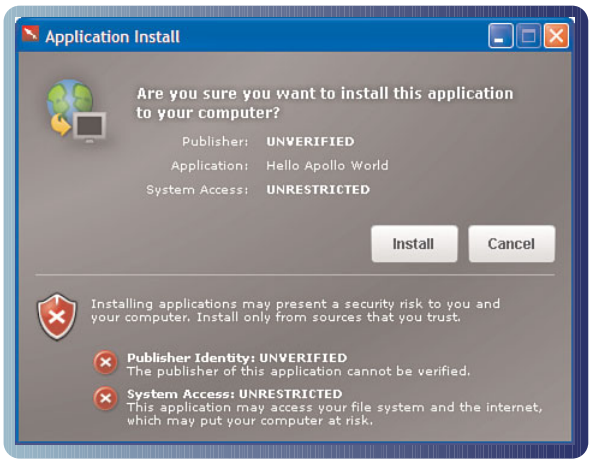


FIGURE 3.15
Installation splash screen.

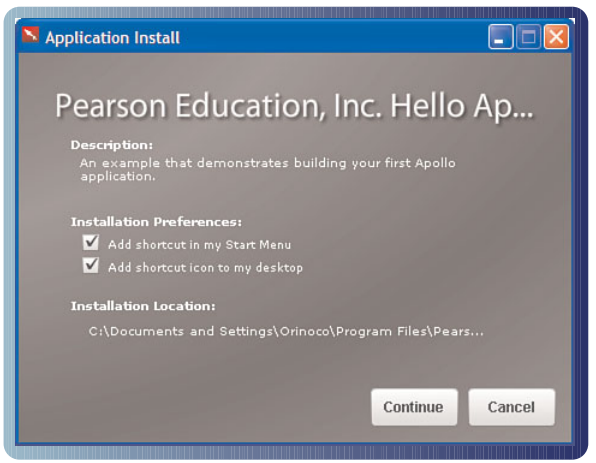


FIGURE 3.16
Installation Completed screen.

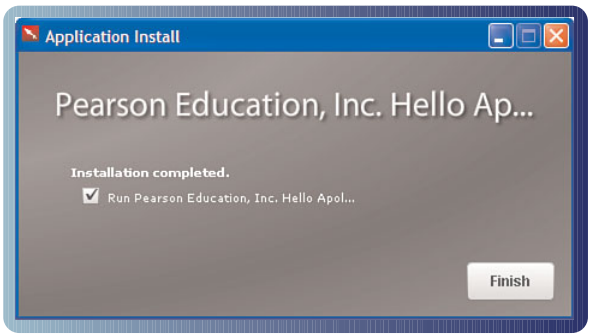


FIGURE 3.17
Your Apollo application shortcut.

Distributing Your Apollo Application AIR File

Distributing your Apollo application is as simple as creating a link on your web server that points to the AIR file.

CAUTION

You will need to add a new mime type to your web server's configuration so it will know what an AIR file is. According to Adobe, you should add the following mime type: `application/vnd.adobe.apollo-install-package` associated with the `.air` extension.

You will also need to include a link to the Apollo runtime on the labs.adobe.com site. You should clearly indicate that the Apollo runtime is required to run your Apollo application. Figure 3.18 shows a sample website that has Apollo application links.

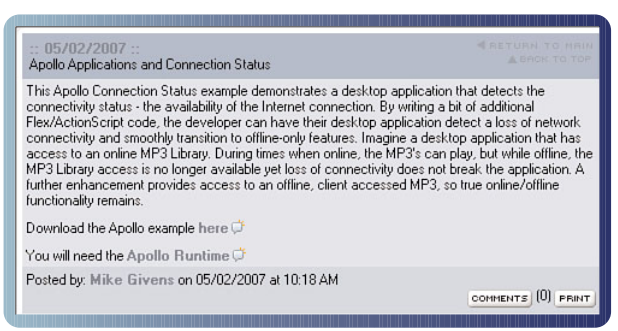


FIGURE 3.18
Sample Apollo application website.

Chapter 4

Docking Apollo with Flex and HTML

Leveraging Flex for Apollo Development

As you saw in Chapter 3, using the Flex Builder IDE, Flex 2, and ActionScript 3 to build your Apollo applications is an ideal and productive workflow. The new Flash CS3 integration and Flex Component Kit for Flash CS3, especially its generation of SWCs for use with Flex, is another possible workflow that enables Flash developers to design the UI. Flex provides the UI interaction with backend systems, and the Apollo framework classes bring it all to the desktop.

NOTE

It is important to note that, for the most part, your existing Flex applications can be converted to an Apollo application by simply changing the opening and closing tags of the main MXML file from `<mx:Application>`...Flex code...`<mx:Application>` to `<mx:ApolloApplication>`...Flex code...`</mx:ApolloApplication>`. Keep in mind that relative paths to assets used in the Flex application will most likely benefit from changing to absolute paths to the assets. See Listing 4.1 for an example. (The `ApolloApplication` tag is subject to change as Apollo matures. Renaming to `WindowedApplication` is likely.)

LISTING 4.1 Relative Versus Absolute Paths

```
// sets the URL for the sound effect with a relative path (Flex)
mySound.url = "assets/sounds/STbridge.MP3";
// sets the URL for the sound effect with an absolute path (Apollo)
mySound.url =
"http://labs.insideflex.com/apollotraining/apolloworld/assets/sounds/STbridge.MP3";
```

Leveraging Flex to create your Apollo applications gives you access to extra Apollo components:

- **HTML Control**—Enables you to display HTML web pages in your Apollo application.
- **FileSystemList Control**—You can display a file system directory.
- **FileSystemTree Control**—You can display a file system directory as a tree.
- **FileSystemComboBox Control**—You can display a combo box for selecting a location in a file system.
- **FileSystemDataGrid Control**—You can display file information in a data grid format. The file information includes the filename, creation date, modification date, file type, and file size.

Listing 4.2 demonstrates the use of the FileSystemTree and FileSystemDataGrid controls on a Windows OS. In this example, the FileSystemTree control displays the files in a directory. Clicking on a directory name in the FileSystemTree control causes the FileSystemDataGrid control to display file-related information in the selected directory.

LISTING 4.2 Code Sample Using Apollo Components

```
<?xml version="1.0" encoding="utf-8"?>
<mx:ApolloApplication xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:HDividedBox height="400">
        <mx:FileSystemTree id="fsTree"
            width="200" height="100%"
            directory="{new File('C:\\inetpub\\wwwroot\\apollotraining')}"
            enumerationMode="directoriesOnly"
            change="fsDataGrid.directory = File(fsTree.selectedItem);"/>
        <mx:FileSystemDataGrid id="fsDataGrid"
            width="100%" height="100%"
            directory="{new File('C:\\inetpub\\wwwroot\\apollotraining')}"/>
    </mx:HDividedBox>
</mx:ApolloApplication>
```

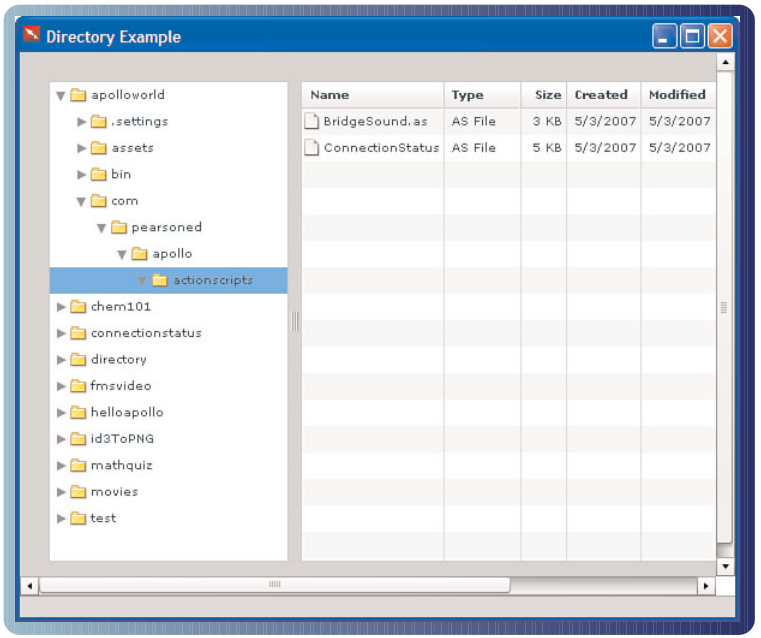


FIGURE 4.1
Example of using Apollo components.

See Figure 4.1 for an example of this finished application.

Additional details and more examples of using the Apollo components are available on the Adobe Labs site at the following URL: http://labs.adobe.com/wiki/index.php/Apollo:Documentation:Using_the_Flex_Apollo_components.

Leveraging HTML for Apollo Development

An alternative to using Flex to develop Apollo applications includes using any combination of HTML, JavaScript, or AJAX frameworks. You will need the Apollo runtime and the Apollo SDK, which contains the command-line tools you will need to launch and package your applications: The Apollo Debug Launcher (ADL) and the Apollo Developer Tool (ADT) are necessary. To use the Apollo command-line tools, Java must be installed on your development machine. You can use the Java Virtual Machine from either a JRE or a JDK (version 1.4.2 or newer). A Java JRE is available at http://java.sun.com/j2se/1.4.2/download.html, and the Java JDK is available at http://java.sun.com/javase/downloads/index.jsp.

NOTE

Java is *not* required for end users that run your Apollo applications (for example, only the Apollo runtime is required for end users).

Additional details on using the ADL are available here: http://labs.adobe.com/wiki/index.php/Apollo:Documentation:Debugging_using_the_Apollo_Debug_Launcher.

Additional details on using the ADT are available here: http://labs.adobe.com/wiki/index.php/Apollo:Documentation:Packaging_an_Apollo_application_using_the_Apollo_Developer_Tool.

The workflow for building an Apollo application with HTML includes creating the project files, creating an Apollo application descriptor file, creating the application HTML page, iteratively testing the application, packaging the final application, and distributing the AIR file.

Every HTML-based Apollo project must have at least two files, an application descriptor file and a HTML page. A sample application descriptor file is shown in Listing 4.3, and a sample HTML page is shown in Listing 4.4.

LISTING 4.3 An Application Descriptor File (ApolloWorld-app.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://ns.adobe.com/apollo/application/1.0.M3"
  appId="com.pearsoned.apollo.ApolloWorld" version="0.1">
  <properties>
    <name>Hello Apollo World!</name>
    <publisher>Pearson Education, Inc.</publisher>
    <description/>
    <copyright/>
  </properties>
  <rootContent systemChrome="standard" visible="true">ApolloWorld.html</rootContent>
</application>
```

LISTING 4.4 HTML Page Example (ApolloWorld.html)

```
<html>
<head>
  <title>Hello Apollo World!</title>
</head>
<body>
  <h1>Hello Apollo World</h1>
</body>
</html>
```

You can test the HTML-based Apollo application with the ADL by opening a command prompt window and entering

```
adl ApolloWorld-app.xml
```

FIGURE 4.2
Hello Apollo World from HTML.

An Apollo window opens, displaying your application as shown in Figure 4.2.

When your HTML-based Apollo application works as desired, you can use the ADT to package the files for installation:

```
adt -package ApolloWorld.air ApolloWorld-app.xml
➥ApolloWorld.html
```

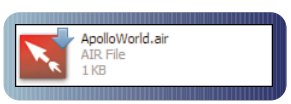


FIGURE 4.3
Hello Apollo World from HTML AIR File.

Figure 4.3 shows the created AIR file.

The AIR file is ready to be deployed. The HTML-based Apollo application is shown in Figure 4.4.

FIGURE 4.4
Hello Apollo World from HTML.

Whether you choose to develop your Apollo applications in Flex or in HTML, the Apollo Flex Builder Extensions or the Apollo SDK allow you to create some highly interactive desktop applications.

Chapter 5

The Apollo File System API

Reading and Writing to the File System

Apollo distinguishes itself from traditional web development in its inherent file I/O capabilities. With traditional web development, files are read or written on the serverside, but with Apollo, client-side file reading and writing are possible. Apollo can create and delete files and directories, copy and move them, list the contents of directories, read and write text or binary files, retrieve information about files and directories, and serialize and deserialize ActionScript objects.

NOTE

Apollo's file system capability is one of its important strengths. The security model for Apollo is a work in progress, and as such, you should download and install Apollo applications only from a trusted source at this time.

Apollo's `FileStream` class provides methods for reading and writing to the file system. The following is the workflow process for reading or writing files:

1. Initialize a file object that points to the path of the file that you want to work with. This can be the path to a file that you will be creating.
2. Initialize a `FileStream` object.
3. Call the `open()` method or the `openAsync()` method.

NOTE

If you want to open the file for synchronous operations, use the `open()` method, or use the `openAsync()` method for asynchronous operations. Use the `File` object as the file parameter of the open method. For the `FileMode` parameter, specify a constant from the `FileMode` class that specifies the way in which you will use the file (`READ`, `WRITE`, `APPEND`, or `UPDATE`).

4. If you opened the file asynchronously (`openAsync()` method), you should add event listeners for the `FileStream` object to handle the possible events (including I/O errors).
5. Include the code for reading or writing data, based on your application's requirements.
6. Call the `close()` method of the `FileStream` object when you are finished with the file.

In ActionScript 3, the code to read a file would look like that shown in Listing 5.1.

LISTING 5.1 Reading a File in ActionScript

```
var myFile:File = File.documentsDirectory.resolve("Hello Apollo World/fileApollo.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.READ);
```

The code to write a file would look like that shown in Listing 5.2.

LISTING 5.2 Writing a File in ActionScript

```
var myFile:File = File.documentsDirectory.resolve("Hello Apollo World/fileApollo.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.WRITE);
```

Keep in mind that you can also use JavaScript to work with Apollo's File System API, as shown in Listing 5.3.

LISTING 5.3 Writing a File in JavaScript

```
var myFile = runtime.flash.filesystem.File.documentsDirectory.resolve("Apollo Test/test.txt");
var myFileStream = new runtime.flash.filesystem.FileStream();
myFileStream.open(myFile, runtime.flash.filesystem.FileMode.READ);
```

Synchronous and Asynchronous Methods

An Apollo file system API discussion would not be complete without talking about the synchronous and asynchronous methods.

NOTE

Synchronous can be thought of as a step-wise process—step 1 is completed, and then step 2 is completed, then step 3, and on and on. *Asynchronous* implies that subsequent steps do not wait on any previous steps in a process—the steps are not synchronized.

Depending on an application's requirements, there are times when a developer needs to programmatically follow a progression of steps (for example, reading in a large XML file which contains data that the application needs before continuing). Apollo's `File` class has synchronous methods for use in these situations. It should be noted that, although the synchronous methods allow you to write simpler code versus the asynchronous methods, important processes, such as display object rendering and animation, may be paused and your application might need additional code to communicate to the end user what is happening (for example, `CursorManager.setBusyCursor()`).

In many Apollo applications you will create, file operations can be handled asynchronously. A code example is shown in Listing 5.4.

LISTING 5.4 Reading a File Asynchronously in ActionScript

```
var myFile:File = File.documentsDirectory.resolve("Hello Apollo World/fileApollo.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(ProgressEvent.PROGRESS, progress_Handler);
myFileStream.openAsync(myFile, FileMode.READ);
var str:String = "";

function progress_Handler(event:ProgressEvent):void
{
    str += myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

For a comprehensive discussion of working with files in Apollo applications, check out this URL: http://labs.adobe.com/wiki/index.php/Apollo:Documentation:Working_with_files#Asynchronous_programming_and_the_events_generated_by_a_FileStream_object_opened_asynchronously.

Serializing and Deserializing Data

Serializing and deserializing ActionScript objects to the file system is an important capability that Apollo brings to the table. By serializing the data that an Apollo application needs to persist, and storing it on the client's file system, you can leverage a cookie-like behavior common in web applications. Although similar in concept to cookies, the serialized data that Apollo works with is not subject to size limitations as cookies are. Apollo's increased storage capacity is also an improvement

over that available for Flex/Flash local shared objects. Some possible uses are as a file-based storage facility—a file-based database. You could permanently store client preferences or temporarily store data to the user's file system while in offline mode. Once online, the data is transferred to a server-side data access component and ultimately pushed to a relational database. Use the `writeObject()` (to serialize) or `readObject()` (to deserialize) methods of the `FileStream` class to read and write the ActionScript objects that you create in your applications.

NOTE

The serialized data is encoded in ActionScript Message Format (AMF), a lightweight format with a low network bandwidth requirement.

An example demonstrates the type of code you write, as seen in Listing 5.5.

LISTING 5.5 Writing/Reading an ActionScript Object

```
<?xml version="1.0" encoding="utf-8"?>
<mx:ApolloApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
➥backgroundColor="#FFFFFF" width="300" height="300">
    <mx:Script>
        <![CDATA[
        import flash.filesystem.FileMode;
        import flash.filesystem.FileStream;
        import flash.filesystem.File;
        import flash.net.registerClassAlias;

        private var oMovieToFS:File;
```

```
private function setMovieToFS():void {
    // register a Dictionary
    registerClassAlias("flash.utils.Dictionary", Dictionary);
    // build the ActionScript Object of Data
    var oMovie:Object = new Object();
    oMovie.Title = "A Beautiful Mind";
    oMovie.Genre = "Drama";
    oMovie.MPAARating = "R";
    oMovie.MovieReleaseDate = "12/21/2001";
    oMovie.RunningTime = "136 minutes";
    // create a file instance that points to the storage location
    oMovieToFS = File.appStorageDirectory.resolve("movie.db");
    var dict:Dictionary = new Dictionary();
    dict["Title"] = oMovie.Title;
    dict["Genre"] = oMovie.Genre;
    dict["MPAARating"] = oMovie.MPAARating;
    dict["MovieReleaseDate"] = oMovie.MovieReleaseDate;
    dict["RunningTime"] = oMovie.RunningTime;
    var fs:FileStream = new FileStream();
    fs.open(oMovieToFS, FileMode.WRITE);
    fs.writeObject(dict);
    fs.close();
    btnDeserialize.enabled = true;
}

private function getMovieFromFS():void {
```

```
                var dict:Dictionary;
                if (oMovieToFS.exists) {
                    var fs:FileStream = new FileStream();
                    fs.open(oMovieToFS, FileMode.READ);
                    dict = (fs.readObject() as Dictionary);
                    fs.close();
                    taMovieInfo.text = "Title: " + dict.Title + "\n"
                    + "Genre: " + dict.Genre + "\n"
                    + "Rating: " + dict.MPAARating + "\n"
                    + "Release Date: " + dict.MovieReleaseDate + "\n"
                    + "Running Time: " + dict.RunningTime + "\n";
                } else {
                    mx.controls.Alert.show("Error deserializing data", "I/O Error");
                }
            }
            ]]>
        </mx:Script>

        <mx:Button label="Serialize Movie Info" click="setMovieToFS()"/>
        <mx:Button id="btnDeserialize" label="Deserialize Movie Info" click="getMovieFromFS()"
        ➥enabled="false"/>
        <mx:TextArea id="taMovieInfo" height="120" width="180"/>
    </mx:ApolloApplication>
```

Figure 5.1 shows a screenshot of the Apollo application from the code in Listing 5.5.

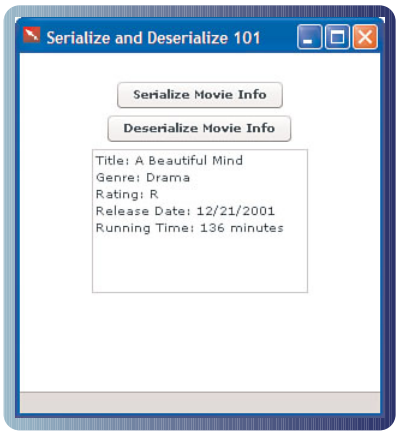


FIGURE 5.1
Serializing and deserializing movie example.

Chapter 6

Go Fly

A Real-World Apollo Application

This Apollo application, written in Flex (the serverside is a ColdFusion component, or CFC), presents a simple user interface that accepts an airline code and a flight number. Clicking the Search button calls a web service that communicates with a third-party, web-based infrastructure that provides close to realtime flight-tracking information (updates are broadcast at approximately six-minute intervals). Knowing the six-minute time lag between flight information updates, the Apollo application uses the Flex `Timer` class to poll the web service every six minutes. Polling that is more frequent would be a waste of resources. By leveraging Flex Data Services (now called Live Cycle Data Services), the Apollo application could detect a real-time update (push technology). Figure 6.1 shows the Apollo UI, and Figure 6.2 displays the flight details.

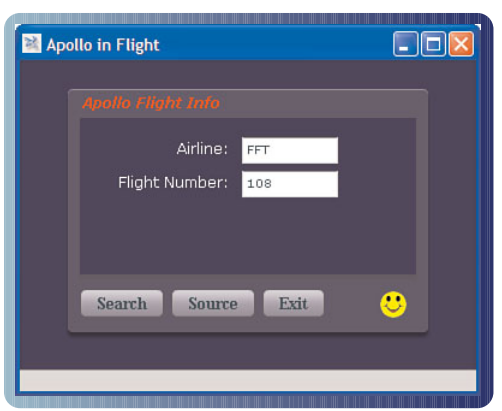


FIGURE 6.1
Apollo Go Fly sample UI.

Building the application includes working with some common techniques used in Flex development. These techniques include using a timer to periodically do something, using a component to display the flight details, using form validation to ensure that the airline and flight number are entered before the web service is called, using a `[Bindable]` metadata tag to bind the flight data to a persistent string variable, and using handlers to perform some unit of work. For this application, the Apollo `HTML` class and `NativeWindow` class are used to display the flight information in an HTML control inside an OS native window.

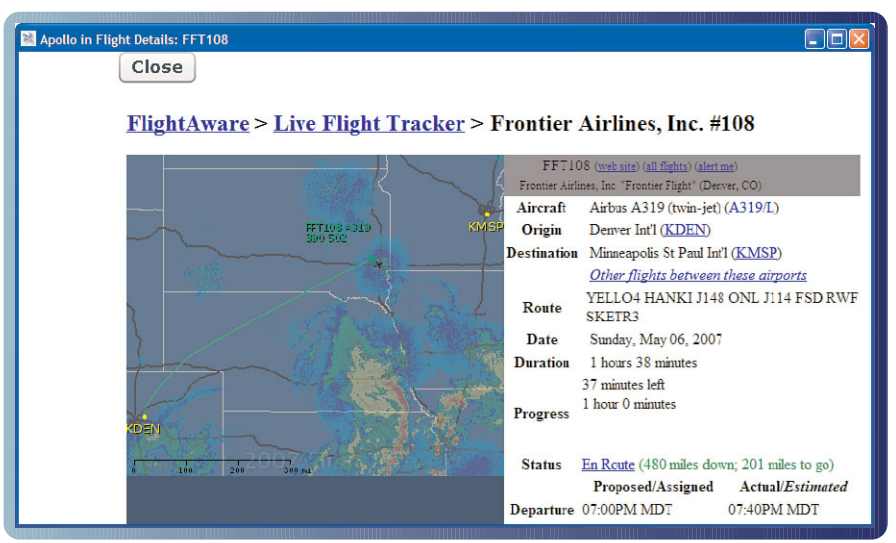


FIGURE 6.2
Apollo Go Fly flight details.

The source code for this Apollo application is available at

http://labs.insideflex.com/apollo-training/gofly/bin/srcview/index.html

The AIR file is available at

http://webmxml.no-ip.info/apollo-training/gofly/gofly.air

Chapter 7

Apollo—Next Steps

Apollo—A Picture-Perfect Splashdown

Back to earth now. We have covered a lot in this Short Cut in a short number of pages. There is much to learn about Apollo development, and although every effort was made to cover the important details of developing Apollo applications, there might have been some unintentional omissions in this writing. With any alpha release of software, the final product can evolve in ways to invalidate some of what has been discussed in this Short Cut. With that considered, I hope you have enjoyed this project and found it useful, and that it has further enticed you to get involved in Apollo development.

Apollo Resources

Apollo resources are springing up all over the Internet, and you can find further information at the following sites:

Adobe Labs: http://labs.adobe.com/technologies/apollo/

Adobe Site: http://www.adobe.com/go/apollo

InformIT.com: http://www.informit.com

Safari.com: http://www.safari.com

ApolloApps.com: http://www.apolloapps.com/

Flexination.info: http://www.flexination.info

Apollo API Docs: http://www.webapollo.info/apps/FlexApolloAPIDocs.air

The only book currently available is O'Reilly's *Apollo for Adobe Flex Developers Pocket Guide*; however, both Que and Sams Publishing will soon be releasing several books on this topic. Check www.samspublishing.com and www.quepublishing.com for upcoming releases.

Appendix

Apollo API Cheat Sheet

TABLE A.1 Apollo ActionScript Classes

Package	Classes
flash.display	flash.display.NativeWindow
	flash.display.NativeWindowDisplayState
	flash.display.NativeWindowIcon
	flash.display.NativeWindowInitOptions
	flash.display.NativeWindowResize
	flash.display.NativeWindowSystemChrome
flash.events	flash.events.FileListEvent
	flash.events.InvokeEvent
	flash.events.HTMLUncaughtJavaScriptExceptionEvent
	flash.events.NativeWindowBoundsEvent
	flash.events.NativeWindowDisplayStateEvent
	flash.events.NativeWindowErrorEvent
flash.filesystem	flash.filesystem.File
	flash.filesystem.FileMode
	flash.filesystem.FileStream
flash.html	flash.html.HTMLControl
	flash.html.JavaScriptFunction
	flash.html.script.JavaScriptObject
flash.system	flash.system.Shell
	flash.system.NativeWindowCapabilities
	flash.system.Updater

TABLE A.2 Flex Apollo Component Classes

Package	Classes
mx.core	mx.core.ApolloApplication
mx.controls	mx.controls.FileSystemComboBox
	mx.controls.FileSystemDataGrid
	mx.controls.FileSystemEnumerationMode
	mx.controls.FileSystemHistoryButton
	mx.controls.FileSystemList
	mx.controls.FileSystemSizeDisplayMode
	mx.controls.FileSystemTree
	mx.controls.HTML

TABLE A.3 Changes to Existing Classes for Apollo

Class	New Property or Method
flash.display.Stage	window property
flash.events.Event	DOM_INITIALIZE constant
	HTML_BOUNDS_CHANGE constant
	HTML_RENDER constant
	LOCATION_CHANGE constant
	NETWORK_CHANGE constant
flash.system.Security	APPLICATION constant
flash.utils.ByteArray	deflate() method
	inflate() method
flash.net.URLRequest	followRedirects
	manageCookies
	shouldAuthenticate
	shouldCacheResponse
	useCache