Chapter 1

Getting Started with Adobe AIR

In This Chapter

- Understanding exactly what Adobe AIR is
- Discovering the significance of a new acronym — RIA
- Exploring the AIR security and signing model
- Setting up your development environment for Adobe AIR

Web developers, unite! For all too long, Web developers have been oppressed by the shackles of the browser window, their creativity stifled by cross-browser compatibility issues, their self-image hurt by the scoffs of desktop app programmers who trivialize browser-based solutions. . .

But that was then; this is now. Or, to mimic the voiceover from an overly dramatic movie trailer, *Everything you know about Web development is about to change. Introducing Adobe AIR. . .*

Adobe AIR promises to liberate developers from the snares, toils, and oppression of their browser-based prisons and enable them to create "rich Internet applications" (RIAs) for the desktop. In true *Braveheart* fashion, maybe you will find yourself shouting from your office or cubicle, "You can take my life, but you can never take my Adobe AIR!"

Okay, perhaps I am guilty of being just a wee bit over-the-top as I introduce Adobe AIR, but I hope the melodrama does serve a purpose. It helps show you that AIR really is not just another flavor of the week. AIR really does provide a greater freedom to do things that HTML/Ajax, Flash, and Flex developers can't do inside the browser.

In this chapter, I introduce you to this "breath of fresh AIR" and get you started working with it. *Viva la RIAs!*

Discovering Adobe AIR

Adobe AIR enables Web developers to create cross-platform desktop applications using and combining familiar Web technologies that they are already skilled in — such as HTML, JavaScript, Ajax, Flash, and Flex.

Even though the technologies used to create it are Web based, an AIR application looks and feels like a normal Windows or Mac OS X program. It runs in its own window, has its own icon, and integrates with the menu system or taskbar. And it generally has the performance you would expect from a native operating system application. In fact, users will interact with an AIR app (see Figure 1-1) just the same as they do with any other application on their desktop.

Creating Internet-savvy apps

An AIR application is technically not standalone. It is actually "powered by" the Adobe AIR runtime that must be installed on any computer in order to run the application. Therefore, when an AIR app is launched, the AIR runtime is automatically loaded behind the scenes prior to the loading of the app.

Figure 1-1: Analytics Reporting Suite delivers a traditional Web application to the desktop.

When you create an AIR application, you build the app using Adobe Dreamweaver, Adobe Flex, Adobe Flash, or any text editor. (In Chapter 2, I show you how to create a basic HTML-based app in a text editor and Dreamweaver. Chapter 3 shows you how to create a basic app in Flex and Flash.)

As you can see, many parts of the application use Web techniques and technologies that you're already used to working with. However, core to Adobe AIR is an application programming interface (API) that you can tap into to do real "desktop stuff," such as get access to local files, open native UI windows, create menus, and so on. I walk you through the API in Chapter 4.

As you begin to explore the AIR API, you will see that the key strength of Adobe AIR is not in creating word processors or spreadsheets (although you can), but rather in enabling Web developers to shed the browser and safely deploy Internet-savvy apps onto the desktop.

An AIR application is easily delivered to users with a single downloadable installer (which has an `.air` extension) regardless of the operating system. (See Chapter 14 for more on deployment.)

Developers can create Internet-based desktop apps to some extent through widgets and Java, but both of these technologies have restrictions or limitations that have kept them as niche players. Widgets are intended for limited single screen, display-oriented purposes (such as a stock ticker). Cross-platform applications using Java runtime have traditionally suffered in comparison to native OS apps — in terms of both performance and "look and feel" issues. Also, both widgets and Java apps are much weaker in working with rich media than Flash has been.

In fact, you may want to jump over to Chapter 16 to take a quick look at ten great AIR applications that help demonstrate the power of the platform.

Peeking inside Adobe AIR runtime

The Adobe AIR runtime may be a relatively new platform, but it actually embeds three highly mature and stable cross-platform technologies to power AIR applications. These are the following:

- **WebKit:** Used for rendering HTML content inside an AIR app. WebKit is an open source, cross-platform browser and is the underlying rendering engine on which Apple's Safari browser is built.

  WebKit is known for its strong support of W3C standards, such as HTML, XHTML, Document Object Model (DOM), Cascading Style Sheets (CSS), and ECMAScript. However, it also provides support for enhanced functionality — enabling the creation of cool stuff such as rounded corners using CSS. Because you're developing solely for WebKit and not for every

browser under the sun, you're free to take advantage of these nonstandard extensions.

For more info on WebKit, go to `www.webkit.org`.

- **Adobe Flash Player:** Used for playing Flash media (SWF files). Flash Player is a cross-platform virtual machine used to run media created in the Adobe Flash authoring environment and full SWF-based applications created using Adobe Flex. Flash Player has an embedded JavaScript-like scripting language called ActionScript 3.

  Inside your app, you can access existing Flash Player API calls as well as some enhanced functionality for vector-based drawing, multimedia support (see Chapter 13), and a full networking stack (see Chapter 12).

- **SQLite:** A database engine for enabling local database access. It's an extremely lightweight, open source, cross-platform SQL database engine that is embedded in many desktop and mobile products. In contrast to most SQL databases, it doesn't require a separate server process, and it uses a standard file to store an entire database (tables, indexes, and so on). If you'd like to explore how to work with SQLite to create database apps, see Chapter 11.

  For more info on SQLite, go to `www.sqlite.org`.

Figure 1-2 shows an overview of the AIR runtime architecture.

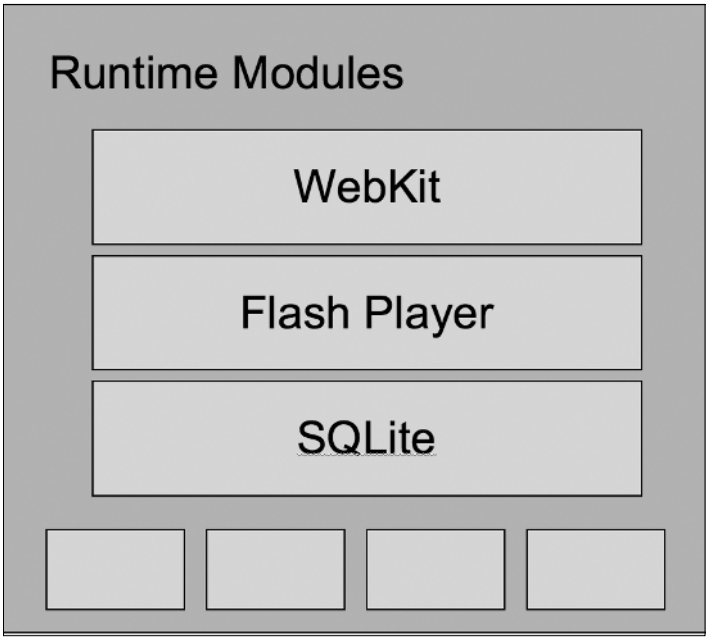

Figure 1-2: Simplistic view of Adobe AIR runtime.

Blurring the lines between HTML and Flash

Having Flash Player and the WebKit rendering engine integrated inside AIR so tightly opens many possibilities for AIR developers. An AIR app can consist of several different possibilities:

- HTML/JavaScript only
- HTML and Ajax
- Flash only
- Flex only
- Flash/Flex and HTML

In fact, AIR blurs the lines between Flash media, a Flex app, and a traditional HTML-based app. In many cases, an AIR application can be a combination of all these. Consider how these technologies can speak to each other:

- You can access the Flash Player and ActionScript Library APIs from within JavaScript. (See Chapter 5 for more details.)
- ActionScript inside Flash can call JavaScript and access and modify the HTML DOM. (See Chapter 5.)
- You can register JavaScript and ActionScript events anywhere — in Flash, Flex, or JavaScript. (You can thumb over to Chapter 6 to dive fully into events.)

Because an AIR app can use all these technologies interchangeably, you can see that Adobe AIR breaks down the traditional walls that have existed in Web development architecture.

Understanding the AIR Security Model

One of the concepts that is important for you to understand from the get-go is application security. Desktop apps get permission in terms of what they can do and cannot do from the OS and the available permissions of the currently logged-in user. They receive this level of access because the user needs to explicitly install the app — effectively telling the computer that the user trusts the app he or she is about to launch. As a result, native apps have access to read and write to the local file system and perform other typical desktop functions.

Web apps, however, are far more restrictive because of the potentially malicious nature of scripting. Consequently, Web apps limit all local file access, can perform web-based actions only inside the context of a browser, and restrict data access to a single domain.

Playing in sandboxes

The hybrid nature of an AIR application puts it somewhere in between both of these traditional security models. On the one hand, with AIR, you create a desktop application that runs on top of the normal OS security layer. Therefore, it can read and write from the local file system. However, because AIR uses Web technologies that, if unchecked, could be hijacked by a malicious third party and used in harmful ways when accessing the local system, Adobe AIR has a security model to guard against such an occurrence. Specifically, AIR runtime grants permissions to each source or data file in an AIR application based on its origin and places it into one of two kinds of containers it calls sandboxes.

The *application sandbox* contains all content that is installed with the app inside the home directory of an application. These are typically HTML, XML, JS, and SWF files. You can think of files inside the application sandbox as the equivalent of premium frequent flyer members that get full access to the special airport restaurants. Only these files have access to the AIR API and its runtime environment.

Adobe AIR does allow you to link in other local and remote content that is not inside the root directory of the application, but places that content in a *nonapplication sandbox.* Content inside the nonapplication sandbox is essentially handled from a security standpoint just as a traditional Web app is, and is not granted access to the AIR APIs (see Figure 1-3).

Check out Chapter 17 for more on application security and sandboxing.

Additional restrictions within the application sandbox

AIR places strict restrictions over script importing of remote content and the dynamic evaluation of JavaScript code — even inside the application sandbox. Many JavaScript programmers use the `eval()` function as a way to generate executable code on the fly. However, if you're loading data from a remote source, a hacker could potentially inject malicious code into your app without your knowledge. To prevent these security vulnerabilities, `eval()` and other dynamic code methods are prohibited after the `onload` event occurs.

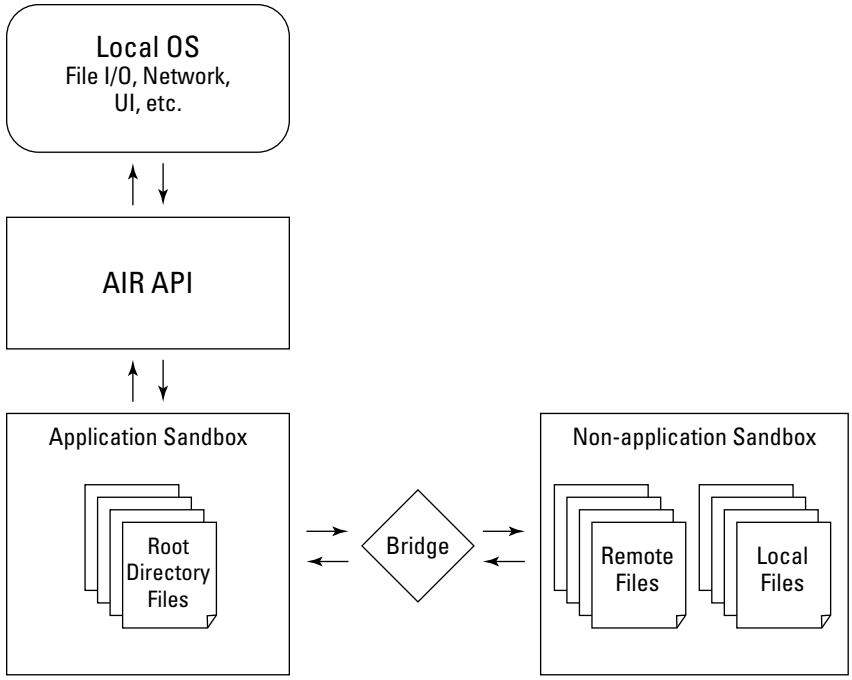

Figure 1-3: Sandboxing is an important part of Adobe AIR architecture.

As it is in Web applications, code being executed inside the application sandbox is free to load data using Ajax (the `XMLHttpRequest` object). However, any content received using `XMLHttpRequest` is treated purely as data and cannot be dynamically changed into executable JavaScript code (such as by using `eval()`).

Table 1-1 lists the specific restrictions of what can be done inside an application sandbox.

Table 1-1 Allowed and Nonallowed JavaScript Activities

Language component	*Before onload*	*After onload*
`eval()`	Permitted.	Not permitted after an application loads, except when you use with a JSON type parameter to convert JSON strings into objects.
`document.write()`	Permitted.	Not permitted.
`Function` constructor	Permitted.	Not permitted.

(continued)

Table 1-1 ***(continued)***

Language component	*Before onload*	*After onload*
`setTimeout()` and `setInterval()` timing functions	Permitted.	Not permitted when using string parameters.
JavaScript protocol URLs (`javascript:`)	Not permitted.	Not permitted.
`innerHTML`, `outerHTML` properties	Permitted.	Attributes of inserted elements cannot be transformed into executable code.
`XMLHttpRequest`	Synchronous calls outside the application sandbox prohibited.	Asynchronous calls triggered in `onload` always finish after onload.
Remote URL for a `<script> src` attribute	Not permitted.	Not permitted.

Digitally Signing an Application

Because users open their computer to an AIR app, their trust in the software publisher is crucial. They need to know that you won't do bad things to their private data or trash their hard drive. That's why digital signing is a required final step of the AIR application development process before you can deploy it.

To provide a degree of confidence and trust, an AIR application must be signed by a code-signing certificate. There are two types of certificates:

- **Self-signed certificates:** "Do-it-yourself" certificates that you can generate with the AIR SDK and then sign your app with. Self-signed certificates provide a minimal degree of trust, but because you have no outside confirmation that you are who you say you are, you are, in effect, telling users, "Hey, you can trust me. Really. Really!" When users install an app with a self-signed certificate, they are warned that the publisher is UNVERIFIED (see Figure 1-4).

  Self-signed certificates are intended mainly for internal use when debugging and testing your app.

Figure 1-4: Self-signed certificates give no assurance to users.

- **Commercial code-sign certificates:** These certificates are purchased from a certification authority (CA), such as Verisign and Thawte, who authenticate your identity. A commercial certificate enables you to be considered a "trusted" publisher and gives users a much higher degree of confidence in working with your app. A commercial certificate enables users to verify the corporate or organizational affiliation of the application and ensures that users can say, "They are who we thought they were!" (see Figure 1-5).

  Commercial certificates, however, are not cheap. Fees are generally around $300 for one year and $549 for two years for a code-sign certificate.

Figure 1-5: Commercial certificates add trust.

Setting Up Your AIR Development Environment

As you begin to work with Adobe AIR, you should begin by configuring your development environment. First, you should install the runtime and SDK. The SDK comes with two command-line tools that you can use to debug and deploy Adobe AIR apps:

- ADL is used for testing purposes only, enabling you to run an app without installing it.
- ADT is used for deploying your app. It packages the app into an installation package.

Adobe also integrates the ability to package AIR apps inside Adobe Flash, Flex, and Dreamweaver (CS3 and later). However, if you use Dreamweaver, you should install the AIR extension to enable you to create AIR apps directly inside the Dreamweaver environment.

The instructions to set up your environment are explained in the sections that follow.

Installing the Adobe AIR runtime

Adobe AIR runtime is the underlying engine that drives any AIR application. As a developer, you need the runtime installed on your machine in order to test and debug your apps. Users also need to download and install it on their computers in order to run an AIR application.

Fortunately, installing the runtime is a quick, "no brainer" process. To install it, follow these four steps:

1. **Go to `get.adobe.com/air` in your browser.**

   The Adobe AIR Web page opens.

2. **On the page, click the Download Now button.**

   The installer file is downloaded onto your computer.

3. **Double-click the downloaded Adobe AIR Installer to launch the setup process.**

4. **Follow the on-screen instructions to complete the setup.**

Installing the Adobe AIR SDK

Although the Adobe AIR runtime has a standard installer that you can use for installing on your computer, installing the SDK involves a few more manual steps. Follow these instructions to get it working on your computer:

1. **Go to** `www.adobe.com/products/air/tools/sdk` **in your browser.**
2. **After reading the Adobe AIR SDK license, indicate that you agree with its terms by selecting the check box.**
3. **Click the download link appropriate for your computer (Windows or Mac).**

   The compressed SDK file — `AdobeAIRSDK.zip` (Windows) or `AdobeAIRSDK.dmg` (Mac) is downloaded to your machine.
4. **Create a folder on your machine for the SDK.**

   I recommend something easy such as `c:\airsdk` for Windows or `/Users/[username]/airsdk` for Mac.
5. **Uncompress the SDK file and copy the folders and files into the SDK folder you created in Step 4.**

   The directory structure under your SDK folder (for example, `c:\airsdk`) will look like this:

```
\bin

\frameworks

\lib

\runtime

\samples

\src

\templates
```

   You now need to add the `bin` subdirectory to your system path before being able to execute the SDK utilities. Follow the appropriate steps below, depending on your operating system.

Setting the environment path in Windows Vista

1. **Press the Windows key and the Pause/Break key at the same time.**

   The System section of the Control Panel is displayed.

2. **Click the Advanced System Settings link.**

   A User Account Control dialog box is displayed.

3. **If required, enter the password for an Administrator account.**
4. **Click the Continue button.**
5. **Click the Advanced tab in the System Properties dialog box.**
6. **Click the Environment Variables button.**
7. **Edit the system variable named** `Path`**.**
8. **At the far right end of the existing path value, type a semicolon and then the path for the** `bin` **subdirectory of the Adobe AIR SDK.**
9. **Test the new path by opening a new Console window and typing** adt **at the command prompt.**

   If you see a listing of the various `usage` options available when calling the utility, then you know you have successfully installed the SDK. If not, go back and check to ensure that you correctly added the SDK `bin` path.

Setting the environment path in Windows XP

1. **Press the Windows key and the Pause/Break key at the same time.**

   The System Properties dialog box is displayed.

2. **Click the Advanced tab in the System Properties dialog box.**
3. **Click the Environment Variables button.**
4. **Edit the system variable named** `Path`**.**
5. **At the far right end of the existing path value, type a semicolon and then the path for the** `bin` **subdirectory of the Adobe AIR SDK.**
6. **Test the new path by opening a new Console window and typing** adt **at the command prompt.**

   If you see a listing of the various `usage` options available when calling the utility, you know you have successfully installed the SDK. If not, go back and check to ensure that you correctly added the SDK `bin` path.

Setting the system path in Mac OS X

Follow these steps to add the path of the AIR SDK to your system path:

1. **Open the Terminal application in your** `/Applications/Utilities` **folder.**

   By default, you will be in your home directory.

2. **Enter** ls –la **at the command prompt.**

   Terminal will display a list of all files in your home directory.

3. **Check to see whether a file called** `.profile` **exists.**

   If so, go on to Step 5. Otherwise, go to Step 4.

4. **If needed, create the** `.profile` **file by typing** touch .profile **at the command prompt.**

5. **Type** open -a TextEdit .profile **at the command prompt.**

6. **Add your AIR SDK** `bin` **subdirectory to the** `export PATH=$PATH:` **line.**

   Here's how mine looks:

   ```
   export PATH=$PATH:/Users/rich/airsdk/bin
   ```

   If you already have an `export PATH` line, add the SDK bin folder to the far right, separating it with a semicolon. For example:

   ```
   export PATH=$PATH:/usr/local/bin;/Users/rich/airsdk/
          bin
   ```

7. **Save the file.**

8. **Quit Terminal.**

9. **Restart your computer.**

10. **Open Terminal.**

11. **Type the following in a Terminal window to load the new settings:**

    ```
    . .profile
    ```

12. **Confirm the path by typing** echo $PATH **at the command prompt.**

    You should see the SDK `bin` path in the output line.

13. **Test the SDK installation by typing** adt **at the command prompt.**

    If you see a listing of the various `usage` options available when calling the utility, you know you have successfully installed the SDK. If not, go back and check to ensure that you correctly added the SDK `bin` path.

Prepping Dreamweaver and Flash for AIR

If you use Dreamweaver or Flash CS3 or higher, you can package and preview applications directly inside the authoring environment, eliminating the need to use the command-line SDK tools.

To do so, begin by going to `www.adobe.com/products/air/tools` and downloading the appropriate software. For Dreamweaver, Adobe provides an MXP extension that you can install using the Adobe Extension Manager. For Flash CS3, you need to install a software update to enable this functionality.