# Chapter 1

# Clearing the AIR



**IN THIS CHAPTER**



The goal of this book is to teach you how to use Adobe Integrated Runtime (AIR) to create desktop applications. You can use JavaScript or ActionScript to develop AIR applications, and you don't have to purchase any software package from Adobe to get started.

From the user's perspective, AIR is similar to the Flash Player, but for a desktop instead of a browser. Immediately after users go to Adobe's site to download AIR, they are able to install and run any AIR application. AIR applications run on PCs as well as on Macs, and a run time for Linux will be available soon. Once installed, an AIR application behaves the same as any other application you have — it has an application-specific desktop and shell icons, windows and themes, and uninstallers.

From the developer's perspective, developing for AIR is very similar to developing for the Web. You use familiar tools to create HTML, JavaScript, and ActionScript. However, instead of deploying them to the Web, you generate an application install package for distribution. AIR provides an API to add additional behavior to JavaScript and ActionScript, so that you can work within the desktop environment. This includes reading and writing to the file system, customizing shell window appearance and functionality, and creating local databases.

## Why Use AIR?

If you are a Web developer, then you have probably been having a bit of a panic lately. It isn't that things are changing — change is the bread and butter of Web developers. It's that *everything* is changing! There are so many new things to learn, it's difficult to focus on which ones are the most important or even the most interesting. There are tools such as Flex, ActionScript 3,

AJAX, Ruby on Rails, Cairngorm, PureMVC, Papervision, Silverlight, and JavaFX. Then there are APIs opening up everywhere — Flickr, Twitter, Google Maps, Last.fm — the list goes on and on. It's definitely enough to make your head spin.

Now something different is thrown into the mix — now you are able to create a desktop application. Why exactly would you want to do this? There are several things that a desktop application can do that a browser application cannot, but the most important difference is that a desktop application can read files from and write files to the file system. This may sound like a minor difference, but consider all the desktop applications you use regularly, and reasons why you prefer to have them running locally and storing data locally. *Offline modes* are essential; you can't always be connected to the Internet (not yet anyway). *Speed* is also an important difference — disk read/write is a bit faster than upload and download. Another important reason is *privacy*.

Clearly there are reasons to have desktop applications, but why use AIR to build them? More to the point, why build them using a scripting language designed for the Web? There are some very powerful tools already established for developing desktop applications. Languages such as C++ give you complete control over system resources and libraries, and there are WYSIWYG (what you see is what you get) editors to simplify the process for developers. Even still, developing an application in C++ is no small undertaking, and a complex application can take a team of developers several months or even years to finish. With AIR, however, familiar scripting languages allow developers to focus on the application user interface, while the run time itself handles the details of the various operating system APIs.

The process of installing AIR and an AIR application is actually very similar to the way Java programming works for desktop applications — the user downloads and installs the Java Runtime Environment (JRE) and is then able to install Java applications. A Java application is going to be able to perform complex routines faster than an application written in JavaScript or ActionScript, but again development for Java tends to be more involved and time consuming than in JavaScript and ActionScript. The download size of JRE is usually around 10MB, which is about the same as that for AIR. The exception to this rule is Mac users — the JRE for Mac OS X Leopard is about 8MB. So again the first major difference you see is in the balance between runtime power and performance (which you will get more of with a lower-level environment) and development time (which you will also get more of with a lower-level environment).

When you start to compare AIR with other technologies, you can find several reasons to use AIR:

- Shorter development time
- Simplicity
- Small file size
- Platform adaptability
- Superior design

First notice that AIR is a bit easier to use than some other options. AIR is still quite new, so it's impossible to give an absolute figure, but it's not unreasonable to estimate that the development cycle for an AIR application would take between 30 and 60 percent less time than the development

cycle for an application with the same functionality in a lower-level language. The reason for that is simple — well, actually, the reason is *simplicity* — AIR only adds a few additional class packages to facilitate working in a desktop environment, as opposed to the host of libraries you may need to learn for a lower-level language.

The second reason to use AIR instead of a similar technology is *small file size*. AIR applications and the run time itself are relatively small downloads. It's a fundamental truth on the Internet that download size can make or break a technology. The Flash community has a special appreciation for that rule — the small file size of the Flash Player and of Flash applications continues to drive the acceptance, the success, and the ubiquity of our lil' pal Flash.

Another important factor is *platform adaptability*. Of course, AIR applications will run on Windows Vista and XP, on OS X Tiger and Leopard, and on Linux. But given that they are written in either JavaScript or ActionScript, the same application can be modified to run in a browser. The extent to which this can be done depends on the application, but it's still a pretty powerful possibility. There aren't many applications out there that have a desktop and a Web interface, and there are almost none that have the same interface for both. If you are building AJAX applications, you can easily have a version that works for iPhones and for the Web, and another version that runs on the desktop!

One more factor worth mentioning is *design*. ActionScript and JavaScript developers are more likely to focus on design and usability than developers of lower-level languages. This one may sound like a shot at other developers just to fire up controversy, since there is no real reason why any other development environment couldn't be used to create a rich user interface. But there is actually inherent truth to this: People who choose to develop in ActionScript or JavaScript choose to focus on the user interface. This may be true for a variety of reasons, but one of the biggest is that they are personally interested in design and usability. You don't need to look far to find an example of an application for which the user interface could be significantly improved. People recognize the demand for this improvement, but some are responding more slowly than others.

It remains to be seen whether the developer community will fulfill the promise of improved application visual design, but a couple of things are certain. One is that it's perfectly capable of doing so — there's little question that the ActionScript and JavaScript communities have been a driving force in improving the visual quality of user interfaces.

More important, it's clear that this improvement is demanded. Recent technology industry battles have played out to clearly demonstrate the demand for solid and friendly user interfaces. The successes of iPod, Wii, and Google all point to user-interface simplicity. Put simply, you would have to be crazy to bring a product to market without making the user interface a primary consideration.

# Comparable Technologies

A wide variety of technologies are being compared to AIR, from other emerging technologies to some more established platforms.

## Flash and AJAX for the Web

One obvious comparison is to the traditional environment for Flash and AJAX.

Some have suggested that the added benefits gained from running an application locally will make browser applications obsolete. This is not likely to be the case; AIR is not going to bring an end to browsers, nor is that the intention. There are limitations to applications that run in the browser, but they're necessary and self-imposed. The fact that browser applications cannot delete files from your hard drive and that you still have some files on your hard drive are not coincidental. The limitations of the browser are *good*, and the usefulness of browser-based applications is not likely to change in the near future.

However, there are applications that will not work within the context of a browser's limitations. It's difficult to imagine any one tool being perfect for every job. Some applications make little sense without the ability to read, write, and delete files from your local hard drive. Other applications make little sense in offline mode and have no added value as a downloaded and installed application other than as a Web site. It's important to use the right tool for the right job.

## Silverlight

Comparing Silverlight and AIR is something of an error. Silverlight is more comparable to Flash, because it runs in a plugin available for most browsers. There is a planned release of Silverlight 2.0 that will have limited file system access. However, you will only be able to open files in read-only mode, and write access will only be available to specific directories. This is similar to the concept of Shared Objects in Flash applications. Developers will also be limited to Windows Vista or XP, which is an increasingly unfortunate limitation in today's Web development community.

On the Web, Silverlight has proven itself to be valuable for some tasks. Some sites that require video with Digital Rights Management (DRM) have been turning to Silverlight, for example. However, Silverlight is still generally unstable and difficult to develop for. It may eventually shape up to be a viable competitor to Flash, but there is no real comparison between the two yet. Flash is far more widely adopted, is available for a wider variety of systems, is easier to use, and creates better-looking content. Once Silverlight is able to create desktop applications, these same factors will make AIR a superior choice as well.

## Google Gears

Google has released an open-source plugin that will also provide offline storage of Web applications. Gears, like Silverlight, runs in the browser. The operating system integration is not as flexible as an AIR application, so things such as customized windows and system menus will not be available. However, the download size of the Gears plugin is quite small, and developers can use JavaScript to access the Gears API.

Gears doesn't provide the same sort of application experience as AIR, but it does provide some similar functionality, such as offline modes, local SQLite database storage, and local file access. For JavaScript developers, Gears may be a viable alternative to AIR, particularly if the custom application experience is not desired.

Mozilla Prism, another emerging technology, allows Web applications written in JavaScript to run in an application-specific browser window, so the application appears to be running outside of the browser. Prism allows the application to be installed and integrated with the operating system just like any other application. Prism does not support any offline functionality by itself though, so while it bears some similarity to AIR, it is not the same. However, the combination of Mozilla Prism with Google Gears would actually provide a similar set of functionality as AIR, so you should expect to see some interesting mashups between these two technologies.

There are actually a few other technologies similar to Gears or related to Gears, such as the Dojo Offline Toolkit for JavaScript. Most of these technologies are conceptually similar — JavaScript developers can create a Web application that runs from a desktop icon, looks similar to a standard desktop application, and takes advantage of some local storage capacity. So far, none of these technologies offer a feature set quite as rich as AIR provides, though most have a smaller plugin than the AIR installer to download.

### Java and .NET

The most comparable tools to AIR may actually be Java and .NET. This suggestion might raise some eyebrows because Java and .NET are both more powerful and robust languages than scripting languages like ActionScript and JavaScript. However, there are some close similarities, as users need to download and install the JRE or the .NET Framework before they can install a Java or .NET application. As mentioned before, the download sizes of those environments are comparable to those of AIR.

One major difference is the contrast between the processing power of a language such as Java and the processing power of a scripting language such as ActionScript. The other difference is the contrast in development time that this level of power and control tends to demand. This could be seen as the choice that developers are now given, depending on the needs of the application: Some applications require more processing power while others benefit from a richer user experience. Another clear contrast is that AIR developers will not be able to access files such as DLLs or JAR files, which can provide significant functionality to an application.

However, this contrast may not be as large as it seems on the surface. Java applications have a reputation for providing a sluggish user experience for even relatively plain-looking applications. Also, many desktop applications don't actually require a great deal of processing power. For most daily use applications, the average user loses more time to interfaces that do not respond to his needs appropriately than he loses to long processes. Because of this, AIR could quickly become a threat even to these well-established technologies.

## AIR Development Platforms at a Glance

You can develop AIR applications using JavaScript, Flash, or Flex.

JavaScript applications will run in the WebKit implementation included in AIR. This is the same code base used in Safari, so coding in JavaScript and HTML for AIR is the same as coding in JavaScript and HTML for Safari. Developers use their editor of choice to write code and then publish using a command-line tool.

You can develop Flash applications in the Adobe Flash CS3 IDE, as well as change their publish settings to compile AIR applications instead of Web deployments.

Flex developers can use Flex 3 to easily create AIR applications.

In all these cases, an AIR application is not restricted from doing anything that a Web application can do. The AIR API is easily accessed from a small set of libraries added to the tools that you are already familiar with.

# Summary

AIR is a runtime environment that allows JavaScript and ActionScript developers to create desktop applications. By choosing JavaScript and ActionScript, you choose to focus your development on user interfaces, so AIR can very easily breathe new life into the desktop. There is a large shift taking place as different groups develop technologies that combine the best of Web technology with the desktop, and AIR is easily one of the most compelling technologies in this movement.