1

The Development Environment

A reliable development environment will save you loads of time when you are ready to debug and deploy your applications. This chapter explains how to install and use command-line tools in building an AIR (Adobe Integrated Runtime) application. This chapter follows a slightly different format than the following chapters, as it introduces the development process in building applications in AIR, and provides an overview of common tools.

The following section, ''SDK Installation,'' explains where to download and how to install the Flex 3 SDK, which includes all required tools and APIs (Application Programming Interfaces) to build Flex and AIR applications. This section covers the basics of the development environment and provides a brief overview of the command-line tools that are used throughout the book to deploy AIR applications.

The section ''Building Your First Application'' walks you through the process of building your first AIR application, focusing on how to use the command-line tools to deploy, debug, and package your application for installation.

The section ''Installing and Uninstalling'' explores the installation process of an AIR application and the storage of application files on a user's machine.

The final section, ''Extra Tools,'' addresses two additional ways for developers to ease the development process: an Apache Ant build scheme to help automate the build process and a simple debug panel to receive basic trace statements from AIR applications.

SDK Installation

Adobe AIR is a cross-platform runtime that runs desktop applications compiled under the AIR framework. By containing two primary web technologies, Flash and HTML, Adobe AIR enables developers to deliver the same Rich Internet Application (RIA) experience to the desktop. It is

not assumed that you have an Integrated Development Environment (IDE), such as Adobe FlexBuilder, which combines a text editor, the necessary compilers, a debugger and other useful tools for your AIR applications. Although you are welcome to use any IDE that you choose to facilitate in the development of your applications, all the examples within this book are deployed and packaged using the Software Development Kits (SDKs) discussed in this section and the command line. Later in this chapter you will learn how you can make this process even smoother by using an Apache Ant build scheme, all while building your first AIR application — the ubiquitous *Hello World*.

Necessary Files

First and foremost, you need to install AIR (Adobe Integated Runtime). If it is not already installed on your machine, go to `www.adobe.com/go/air` and download the AIR runtime. Because this is a book on developing AIR applications and not necessarily an overview of the AIR runtime or Flash software itself, it is recommended that you follow the documentation on installing the runtime provided by Adobe, and refer to the manufacturer's guide should you decide to customize your installation of the AIR runtime.

Once the AIR runtime is installed, it is time to start setting up a reliable environment that will enable you to deliver applications without having to think about that aspect of the process during development. You'll be free to code, and when it comes time to debug or deploy, the system will be in place to handle all aspects of those stages.

For this development environment, you need to download the Flex 3 SDK, which includes the AIR API and all the required command-line tools. The AIR SDK is also available for developers creating applications using HTML or AJAX, though the examples in this book are built primarily in Flex.

SDK Installation

This section explains how to download and install the Software Development Kits (SDKs) available from Adobe, as well as how to configure the setup to use the command-line tools found within.

Flex 3 SDK

Download the Adobe Flex 3 SDK from `http://opensource.adobe.com/wiki/display/flexsdk/Flex+SDK`. Once the download is complete, a file titled `flex_sdk_3` (or similar) will be available within the directory to which you downloaded the file. In this case, that was the desktop. Unzip the contents of `flex_sdk_3` into a directory whose location reflects how you work and organize files. For the purposes of the following examples, the Flex 3 SDK was installed at `C:/flex_sdk_3` on Windows and `/Applications/flex_sdk_3` on Mac OS X. If your installation directory is different, then change all references to this installation directory to where you have installed the Flex SDK.

Add a path to your system properties to enable you to call some valuable executables during our development process. The files you are primarily concerned with — since they relate to debugging and deploying — can be found in the `/bin` folder within the Flex SDK installation directory. The executables for the compilers call the Java programs, which can be found in the `/lib` folder within the directory. You have the option of adding the Java programs themselves to your `CLASSPATH` variable, but you will be accessing these native executables themselves — namely, `adt.bat`, `adl.exe`, `mxmlc.exe`, and a`mxmlc.bat`. In order to call these directly from the command line without having to provide the full path each time (such as `C:/flex_sdk_3/bin/fdb`), add the `/bin` directory to the `Path` variable.

On Windows:

1. Open System from the Control Panel.
2. Click the Advanced tab from the menu.
3. Click on Environment Variables.
4. Within the System variables grid, navigate to and double-click on Path.
5. In the Variable Value field, if the last character is not a semicolon (;), enter a semicolon and then the path to the `/bin` folder within your Flex SDK installation directory. For the purposes of this example, that would be `;C:\flex_sdk_3\bin`.

On Mac OS X:

1. Open the Terminal and type > **open -e .profile**.
2. If you do not have a `PATH` variable set, add the following line:

```
export PATH:$PATH://Application/flex_sdk_2/bin/
```

   If you do have a PATH variable set, add the path to the bin folder after first entering a colon (:) to delimit the paths.

Now you are ready to access the command-line tools directly. These tools require Java to be installed on your machine as well. You probably already have Java installed; if not, download the JRE from `http://java.sun.com` and install it in a directory of your choice. Try to obtain the latest stable release, but note that all the code within this book — including the Apache Ant build scheme — has been tested with Java 1.5 + . Remember where you install the JRE, or remind yourself where it is already installed, as you may need that directory path when you are ready to debug your Hello World application.

Command-Line Tools Overview

Within the `/bin` folder of the Flex SDK installation directory are the command-line tools that you will use for debugging, deploying, and packaging AIR applications. The `.exe` files with the Flex logo icon (a black circle with the text `fx`) are the executable files included with the Flex SDK package. The `.bat` files and `adl.exe` (with the AIR logo icon) are the executable files that come with the AIR SDK.

The `mxmlc` executable is the compiler responsible for generating Shockwave Flash (SWF) files out of MXML and ActionScript code. Its AIR ''sister'' — the `amxmlc.bat` file — uses the `air-config.xml` file found in the `/frameworks` folder to configure the compiler for AIR. Running `mxmlc` itself uses the `flex-config.xml` file, also in the `/frameworks` folder. The `amxmlc` compiler is invoked for both deploying and debugging the SWF file to be packaged in your AIR application installer. AIR application installer files are created by invoking the ADT batch file (`adt.bat`) found in the `/bin` folder of your installation directory.

In this context, the term *deploy* refers to the creation of the SWF file that will be packaged with your AIR application. By invoking the `amxmlc` compiler with your main `.mxmlc` file as an argument, a SWF file is generated. The SWF file, however, is not launched during this operation, and using the `amxmlc` compiler alone will not enable you to preview the Flash movie running within the AIR runtime environment. Running the AIR Debug Launcher (ADL) will launch the SWF file generated by the `amxmlc` compiler.

In addition, if the `amxmlc` is set to debug mode when compiling the SWF file, you are given the opportunity to debug your applications by running the ADL and Flash Debugger.

When you create an installer file for your AIR application you use the AIR Development Tool (ADT). AIR installer files need to be code signed. It is recommended that you sign your AIR application by linking a certificate from a certificate authority (such as VeriSign and Thawte), as it verifies that you (the publisher) are the signer and that the application has not been maliciously altered — ensuring end users that they are not installing a counterfeit version. You can create a self-signed certificate using the ADT tool as well. Note that when using a self-signed certificate, your identity as the publisher cannot be verified during installation. Moreover, if you later release an update to an existing application with a self-signed certificate, then you need to use the original certificate when creating the installer file. The examples in this book include the process of creating self-signed certificates.

In the next section you'll jump right in and start building in order to see how the development environment you have set up by installing the SDK and setting System Paths to the command line tools will aid your efforts to debug, deploy, and deliver your first AIR application — the familiar *Hello World*.

Building Your First Application

Now that you have the necessary files for deploying and packaging AIR applications, you should carefully consider how you intend to develop your applications. Creating a directory in which you will develop is just as important, as it is where you will spend most of your time in delivering a product.

> *Although the examples in this book use the project directory structure outlined in the following sections, it is not intended to replace any organization techniques you may already have. If the structure used here does not agree with your workflow, you are encouraged to employ your approach and edit accordingly.*

Some IDEs actually create a development directory within the `Documents` directory on your machine — User ➪ My Documents on Windows and User ➪ Documents on Mac. Though that is a viable solution for your development environment, for the purposes of this book, create a folder called `Development` in the root drive on your machine — `C:/` on Windows, the hard drive name on Mac. Within that folder will reside a directory of the type of software you are using to develop your applications, including any subdirectories of the projects therein.

On Windows:

1. Navigate to the root of the `C:/` drive.
2. Create a folder titled `Development`.
3. Within that `Development` folder, create a folder titled `AIR`.

On Mac:

1. Navigate to the hard drive.
2. Create a folder titled `Development`.
3. Within that `Development` folder, create a folder titled `AIR`.

Within this directory — `C:/Development/AIR` or `/Development/AIR` — will reside your AIR project subdirectories. From here on, this will be referred to as the ''development directory.''

Your first project will create a simple *Hello World* application. There is nothing special about this application and it will not involve much of the AIR API other than the `WindowedApplication` container. It is intended to introduce you to compiling, debugging, and packaging your application.

Within your development directory, create a folder named `HelloWorld`. You need at least two files in order to compile and deploy AIR applications:

- A main file — This can be either an MXML (`.mxml`) or ActionScript (`.as`) file.
- An application descriptor file using the XML markup

Most, if not all, of the examples found in this book have an MXML file as the application's main file. This is the file you will supply as the main file argument to the `amxmlc` compiler when it comes time to compile. In the case of the *Hello World* application, this will be `HelloWorld.mxml`.

The naming convention for the project's application descriptor file used throughout this book appends the string `-app` to the project's name, though you can name the file as you wish. In the case of the *Hello World* application, the application descriptor file is named `HelloWorld-app.xml`. The generated SWF file and the application descriptor will be used to deploy and preview your application prior to packaging. To create an AIR installer, you will need a signed certificate, the generated SWF, the application descriptor file, and any ancillary files required (such as graphics files).

Because the Flex and AIR command-line tools are available within the SDK you have installed, you are not reliant on an IDE and can continue development by firing up your favorite text editor.

Code and Code Explanation

Open your favorite editor and create a new document. Then, complete the following steps:

1. Enter the following markup (alternatively, you can find this example in the code files for this chapter on the accompanying website):

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://ns.adobe.com/air/application/1.0">
    <id>com.aircmr.HelloWorld</id>
    <version>0.1</version>
    <filename>HelloWorld</filename>
    <name>Hello World</name>
    <description>An AIR app to say Hello</description>

    <initialWindow>
        <content>HelloWorld.swf</content>
        <title>Hello World</title>
        <systemChrome>standard</systemChrome>
        <transparent>false</transparent>
        <visible>true</visible>
    </initialWindow>
```

```
    <installFolder>AIRCMR/HelloWorld</installFolder>
    <programMenuFolder>AIRCMR</programMenuFolder>
</application>
```

2. Save the file as `HelloWorld-app.xml` in the project directory you previously set up — `C:\Development\AIR\HelloWorld` or `/Development/AIR/HelloWorld`, Windows and Mac, respectively. Create a new document and enter the following:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    windowComplete="completeHandler();">

    <mx:Script>
        <![CDATA[

            // window has completed initial layout and is made visible.
            private function completeHandler():void
            {
            }

        ]]>
    </mx:Script>

    <mx:Label text="Hello World" />

</mx:WindowedApplication>
```

3. Save the file as `HelloWorld.mxml` in the project directory along with `HelloWorld-app.xml`.

The first snippet, saved as `HelloWorld-app.xml`, is the descriptor file for the *Hello World* application. Although the descriptor file is not needed to compile a SWF using the `amxmlc` compiler, it is necessary for debugging and previewing, as well as packaging the application for installation. The descriptor file specifies properties associated with the AIR application. It contains values that will be displayed during installation, and data to be displayed by the operating system after installation. The descriptor file also allows you to customize the appearance of the AIR application chrome — or visual display window — within the `<initialWindow>` element node. The properties defined in this example are only a subset of those available. Many are optional and have been left out. Subsequent chapters use them as they pertain to the specific applications you will be building.

The `application` tag is the root tag for the application descriptor file. The `xmlns` attribute is the URI reference for the AIR namespace. The last segment of the namespace (1.0) is the runtime version required to run the AIR application. The required child elements of the root application tag are `id`, `filename`, `version`, and parent `initialWindow`. The value of the `id` element must be unique — meaning no two AIR applications installed on your machine can share the same application ID. As such, best practice is to use a dot-delimited reverse-DNS-style string as an identifier. Supported characters for `id` include 0–9, a–z, A–Z, dot, and hyphen. Any special characters (aside from dot and hyphen), including spaces, will throw an Invalid Application error upon launch of the AIR application. For this example, the unique ID is given the value `com.aircmr.HelloWorld`, as it relates to the examples found in the code for this chapter on the accompanying website.

The `filename` element reflects the name given to the application and is displayed as the value for ''Application name'' within the installer window. With the optional `programMenuFolder` property set in the descriptor file, the filename will also be displayed within the subpath of the Windows Start ➪ Programs menu.

The `version` attribute is an application-defined designator and does not relate to the version of AIR. Its value is specified by the developer, and typically conforms to the standard `MajorVersion.MinorVersion` form. Within this example, the `version` value is `0.1`.

The `initialWindow` element has various optional child elements pertaining to the properties applied to the initial application window. Properties, such as positioning and size, are applied as children of the `initialWindow` node, but most have a default value and are considered optional. The `title`, `systemChrome`, `transparent`, and `visible` properties are all optional but are included in this example with their default values because they are frequently used to customize the visual aspects of your application. The only required property of `initialWindow` is the `content` element. The value for `content` is the URL of the SWF file generated when you compile your application.

The `name`, `description`, `installationFolder`, and `programMenuFolder` properties are all optional. The `description` value is displayed within the installation window. The `name` value appears in the title of the installation window; it is the application name available in the uninstall panel and is the name given to the installation folder. The `installFolder` element's value specifies the installation location of the application within the application directory of your operating system — `C:\Program Files` on Windows and `HD name/Applications` on Mac OS X. The `programMenuFolder` is the subpath displayed in the Windows Start ➪ Programs menu, where the application can be found under the value of `filename`. This setting is ignored by any operating system other than Windows. The `installFolder` and `programMenuFolder` property values follow the same character restrictions as the file and folder naming conventions of your operating system.

Many more properties can be set in the application descriptor file that are specific to the operating system (such as associated file types) and the look and feel of your AIR application (such as icons). These are optional and will be addressed in applications throughout this book.

The second snippet you saved as `HelloWorld.mxml` is your main application file. This book does not cover the Flex language and MXML markup, but it does describe additions and changes to the Application Program Interface (API) that pertains to AIR. A good first start is the `WindowedApplication` container:

```
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    windowComplete="completeHandler();">
```

The `WindowedApplication` tag defines the container for the application. Those familiar with the Flex language are used to defining `Application` as their application container. In fact, `WindowedApplication` extends the `Application` component. It has extra attributes that can be set in the descriptor file and overridden within this tag, as well as specified events and styles unique to `WindowedApplication`. The `windowComplete` event is one such unique event, which is dispatched when the window's initial layout is completed and made visible. The `completeHandler()` method within the `<mx:Script>` tag is the handler for this event. Currently, no operations are performed in this method, but one will be added when the ADL tool is discussed in the next section.

To top off this mug of magic, a `Label` component is included, with the `text` property set to `"Hello World"`:

```
<mx:Label text="Hello World" />
```

With the `layout` attribute of the `WindowedApplication` tag set to `"vertical"`, this label appears centered at the top of the container in the launched application. So far, discussion has concerned what will happen once the application is installed and running. Now it's time to launch it.

Compiling, Debugging, and Packaging

In the previous section you created the necessary files for compiling and previewing your application. In this section you will use the command-line tools provided in the Flex 3 SDK to compile, preview, and package your first AIR application.

Compiling

To compile the AIR application, use the `amxmlc` compiler found in the `/bin` folder where you installed the Flex 3 SDK — for the purposes of this example, that is `C:\flex_sdk_3\bin` on Windows and `/Applications/flex_sdk_3/bin` on Mac OS X. The `amxmlc` compiler invokes the `mxmlc` compiler with an additional parameter pointing to the `air-config.xml` file found in the `/frameworks` folder of your installation directory. Compiling a Flex application using the `mxmlc` utility will load the `flex-config.xml` configuration file. The difference between the two configuration files is the addition of SWC files related to AIR, including `airframework.swc` and `airoglobal.swc`. Take a look at the `air_config.xml` file within the `/frameworks` folder of your SDK installation directory. The `amxmlc` compiler tool defaults to using `air-config.xml`, but you may also specify your own configuration file by using the `-load-config` compiler option.

To generate the application SWF file, begin by opening a command prompt. Change the directory to the `HelloWorld` project directory — for this example, that location is `C:\Development\AIR\HelloWorld` on Windows and `/Development/AIR/HelloWorld` on Mac. Enter the following.

On Windows:

```
> amxmlc -load-config "C:\flex_sdk_3\frameworks\air-config.xml" -output
   HelloWorld.swf HelloWorld.mxml
```

On Mac:

```
> amxmlc -load-config "/Applications/flex_sdk_3/frameworks/air-config.xml" -output
   HelloWorld.swf HelloWorld.mxml
```

The `air-config.xml` option points to the location where you have installed the Flex 3 SDK. Compiling the application with `amxlmc` produces a SWF file and does not launch the application on your system. If you click on the SWF file created, you won't see anything except the default background color. That's because it is running in the Flash Player, not the AIR runtime. You will launch the application using the ADL executable file later in this section, as well as package the application for installation using the ADT. To perform these operations, however, you first needed to generate the SWF file of the application using the `amxmlc`.

Basic options were purposely included in the entered command, and in fact you could have entered `> amxmlc HelloWorld.mxml` and achieved the same result. For the purposes of this example, the `-load-config` and `-output` options are included, as they are common settings you might override during compilation of your application.

> *The* `-benchmark` *option can be added to display not only the operations that take place when generating a SWF file from the* `amxmlc` *compiler, but also the amount of time for each operation.*

The `-load-config` option points to the configuration file to be read by the `amxmlc` compiler. It is included here to show that you can specify a different configuration file for your project. You also have the opportunity to add a custom configuration to the `air-config.xml` file by specifying the following:

```
-load-config+=myConfig.xml
```

The `-output` option specifies the SWF file to which the application is compiled. When this option is left out, the compiler defaults to naming the SWF file whatever the main MXML or ActionScript file supplied as the file parameter. For the purposes of this example, the generated SWF is `HelloWorld.swf`. Note that the name of the generated SWF file must match the value given to the `content` element of the application descriptor file. Therefore, if you override the default output, make sure that the name matches the SWF that will be run within the AIR runtime for your application.

Also available are a handful of other options, which mainly involve adding SWC files to the library path and including Runtime Shared Library (RSL) files. These are discussed as they relate to projects later in this book but have been left out of this simple example.

Previewing and Debugging

The SWF file was generated using the `amxml` compiler, yet you have seen that the application does not launch upon completion of that command, and opening the SWF file in the Flash Player does not display anything other than the background color of the movie. To launch and preview the application, run the AIR Debug Launcher (ADL) tool found in the `/bin` folder of your installation directory. A single argument of the application descriptor file is needed to run the ADL. If you look at the descriptor file (`HelloWorld-app.xml`), you'll notice that the `content` value of the `initialWindow` element is set to the SWF you previously compiled.

Open a command prompt, change the directory to the project development directory, enter the following command, and press Enter:

```
> adl HelloWorld-app.xml
```

Your launched application is now running within the AIR runtime. Everything looks perfect and you can see the text ''Hello World'' rendered in your label. You can run the ADL alone to preview your application, but in developing a more complex application, you need some support for debugging. The ADL has not been utilized to the potential from which its acronym is derived — AIR Debug Launcher. To do so, first add a `trace` statement to the `HelloWorld.mxml` main file and recompile with the `-debug` option set for the `amxmlc`.

1. Open `HelloWorld.mxml` in a text editor and add the following shaded line:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication
```

```
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    windowComplete="completeHandler();">

    <mx:Script>
        <![CDATA[

            // window has completed initial layout and is made visible.
            private function completeHandler():void
            {
               trace( "Hello World!" );
            }

        ]]>
    </mx:Script>

    <mx:Label text="Hello World" />

</mx:WindowedApplication>
```

2. Save the file as `HelloWorld.mxml`. Open a command prompt and change the directory to the project development directory — for this example, that would be `C:\Development\AIR\HelloWorld` on Windows and `/Development/AIR/HelloWorld` on Mac. Enter the following and press Enter:

```
> amxmlc -debug HelloWorld.mxml
```

   In comparison to the previous `amxmlc` command, the `-load-config` and `-ouput` options were omitted but generate the same result. Also added was the `-debug` option to generate a SWF file that has debugging enabled. A SWF file with debugging enabled allows communication between the application and the Flash Debugger tool.

3. Before running the ADL again, you need to run the Flash Debugger (FDB) in order to view the `trace` statement you added to the main file (`HelloWorld.mxml`). The FDB command-line tool can be found in the `/bin` directory where you installed the Flex 3 SDK. Since you added a path to the `/bin` folder of the installation to the system variables earlier in this chapter, you only need to enter `> fdb` into the command prompt to start the debugger.

4. Open a separate command prompt and start the Flash Debugger by entering the following and pressing Enter:

```
> fdb
```

   The printout will end with the FDB display prompt `<fdb>`, meaning you have started the Flash Debugger and it is ready to be run and connected with a launched debug SWF file. Running the debugger enables you to see any `trace` statements you have in your ActionScript code output in the console. Numerous other commands can be performed by the FDB, including setting breakpoints and handling faults; enter the **help** in the console after the command prompt to view these options.

If the preceding operation resulted in output on the console that states

```
Error: could not find a JVM
```

then you will have to manually enter the directory in which Java is installed on your machine. To do so, open the `jvm.config` *file located in the* `/bin` *folder of your Flex 3 SDK installation directory. The first variable listed,* `java.home`*, needs to point to the location of your installed Java Virtual Machine. Following the equals sign (=), enter that location. For example, the JRE directory might be* `C:/Program Files/Java/jre`*.*

Windows users must use forward slashes or double back-slashes for the location. If you don't, then the error will be thrown again when you try to start the FDB.

5. To run the debugger, enter the command **run** in the console and press Enter:

```
<fdb> run
```

If you are prompted with a security alert requesting assistance regarding running the FDB, select Unblock from the options. The console will display ''Waiting for Player to Connect.''

6. Open a separate command prompt and change the directory to the project development environment. For the purposes of this example, that is `C:\Development\AIR\HelloWorld` on Windows and `/Development/AIR/HelloWorld` on Mac. Enter the following command and press Enter:

```
> adl HelloWorld-app.xml
```

7. Switch to the console in which you started the Flash Debugger. You will see that the Flash Player has connected and the debugger is waiting for additional commands to perform. Enter the following and press Enter:

```
<fdb> continue
```

This will launch the Hello World application and display the following within the console in which the FDB running:

```
Additional ActionScript code has been loaded from a SWF or a frame.
Set additional breakpoints as desired, and then type 'continue'.
```

8. Enter the **continue** command again and press Enter. The console will print out the `trace` statement you added to `HelloWorld.mxml`:

```
[trace] Hello World!
```

By launching the debug version of the application using the ADL, you can do simple debugging with `trace` statements and have the opportunity to have a more robust debugging platform by setting breakpoints and fault handling if necessary. To stop the Flash Debugger, select the command prompt in which it is running and press Ctrl+C on Windows or Q on Mac.

Packaging

At this point, the SWF file has been compiled for your AIR application and you have your descriptor file. You also ran a debug version through the FDB to ensure that everything is working correctly. Before proceeding with packaging your application, if you have created a debug SWF, you may want to run the `amxmlc` compiler again without the debug option to generate the SWF file to be included with the installer file.

Now all that is left is packaging the application into an AIR file for installation using the AIR Developer Tool (ADT). The ADT batch file can be found in the `/bin` folder or the AIR installation directory, and has commands to create and sign certificates and create AIR application installer files.

Preparing an AIR installer file is a multi-step process, as each application must be signed using a digital certificate. As mentioned earlier, you can sign your AIR application by linking a certificate from a certificate authority, such as VeriSign or Thawte, or you can use the ADT to create a self-signed certificate. Using a certificate from a recognized provider assures users that they are not installing a fraudulent copy of your AIR application. Using a self-signed certificate does not provide information verifying the identity of the signer, but for the purposes of the examples used throughout this book, the applications you build will use a self-signed certificate.

To create a certificate to be signed when packaging your application into an AIR installer file, open a command prompt and change the directory to that of the development folder for the Hello World project. Enter the following command and press Enter:

```
>adt -certificate -cn HelloWorld 1024-RSA certificate.pfx password
```

Included in this command to generate a certificate and associated private key is the common name, the key type to use for the certificate (either 1024-RSA or 2048-RSA), the path of the generated certificate file (using either the `.pfx` or `.p12` extension) and the password used when signing the AIR file. To learn about additional operational parameters you can add when using the ADT `-certificate` command (such as your organization name), refer to the AIR documentation or enter > **adt -help** at the command prompt.

The certificate and private key are stored in a PKCS12-type keystore file, which is used to sign and package an AIR file using the `-package` command of the ADT tool.

With a command prompt open and pointing to the HelloWorld project development folder, enter the following command and press Enter:

```
> adt -package -storetype pkcs12 -keystore certificate.pfx HelloWorld.air
   HelloWorld-app.xml HelloWorld.swf
```

The `-storetype` parameter value is the version of the PKCS (Public Key Cryptography Standards) standard for storing private keys and certificates. The container, or keystore — created in the previous `-certificate` command and provided here as `certificate.pfx` for the `-keystore` parameter — is encrypted with a password. When you run the `-package` command, you will be prompted to enter the password you supplied when creating the certificate.

Following the signing options of the `-package` command is the name of the generated AIR file, the application descriptor file, and the application SWF file. Along with the application SWF file, you can specify additional space-delimited files and directories to be added to your build. Additional values (delimited by spaces) could include any icon files or embedded media you may need packaged with your application. The files and directories, however, need to reside in the current directory or any subdirectories in which the ADT is run. As such, you can use the `-C` option to change the directory for included files.

For example, you may have the following folder structure within the HelloWorld project:

```
/assets
   /images
      HelloWorld.png
/bin
   HelloWorld.swf
```

By running the following command using the `-C` option, you can include those files in your build:

```
>adt - package -storetype pkcs12 -keystore certificate.pfx HelloWorld.air
   HelloWorld-app.xml -C ./bin HelloWorld.swf -C ./assets/images HelloWorld.png
```

Relative paths between the current directory and the specified files in the directory paths using the `-C` option will keep their structure within the installation folder. For example, specifying `-C modules/module.swf` will place the `module.swf` file in a subdirectory titled `modules` within the application directory. All files not specified using a relative path will be included at the root of the package directory. If you prefer to have a specific directory structure within the application directory, then the `-e` command-line option is available, which enables you to place a file in a specified directory.

Whether you choose to specify the package file using the `-C` option or not, if all goes well, after running this command and entering your password, an AIR installer file named `HelloWorld.air` will be generated in your project directory.

Installing and Uninstalling

Installing an AIR application is as easy as double-clicking the AIR installer file. This section will show you how to install the Hello World application and explain what happens at the system level when an application is installed. You will also learn how to uninstall an AIR application, and the results of doing so.

Installing the Hello World Application

Double click on the `HelloWorld.air` file you generated previously, found in its usual location. Upon double-clicking the AIR file, the dialog shown in Figure 1-1 will be presented.

In the installer window shown in Figure 1-1, you will see the application name you specified in the `filename` element of your `HelloWorld-app.xml` descriptor file. You are also presented with the options to Install or Cancel. By clicking the Install button, the dialog shown in Figure 1-2 is displayed.

The title and description displayed in Figure 1-2 are the values specified for the `name` element and the `description` element in the application descriptor file, respectively.

You are presented with the option to cancel the installation again, but roll the dice and click Continue. The window will briefly display a progress bar, indicating the installation process. If you have chosen to start the application after installation, then when the installation has finished you will see the screen shown in Figure 1-3.

Amazing, right?

Figure 1-1

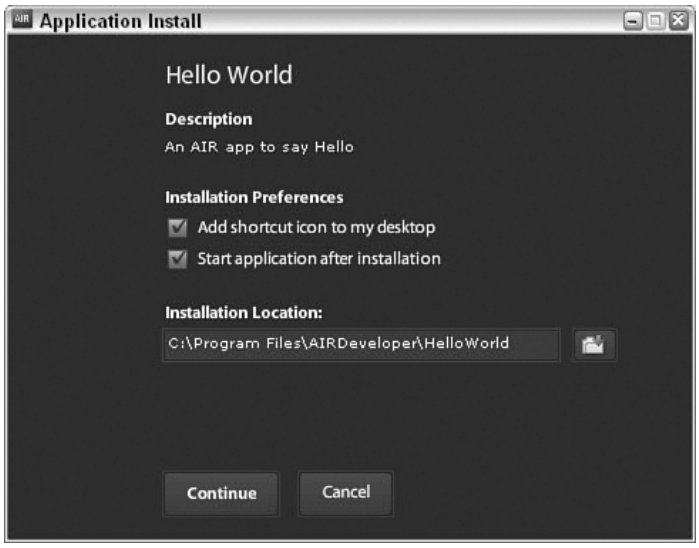

Figure 1-2

While waiting for the money to roll in, peruse the next sections, which cover what occurs during installation of an AIR application on your machine.

On Windows and Mac OS:

- Installs the application in the programs directory of your operating system. A directory is created using the path specified for the `installFolder`, and the application folder is titled after the `name` value specified in the application descriptor. This is known as the *application directory* and is

accessed using the `File` class from the AIR API, as you will see in the examples throughout this book. For the example in this chapter, you will find the application installation at `C:\Program Files\AIRCMR\HelloWorld\Hello World` on Windows and `HD/Applications/AIRCMR/HelloWorld` on Mac OS X.

- Creates a directory for application storage titled after the value for the `id` element in the application descriptor file. Created within this directory is a folder titled `Local Store`. This is known as the *application storage directory* and is accessed using the `File` class from the AIR API. For the example in this chapter, you will find the storage directory at `C:\Documents and Settings\<user>\Application Data\com.aircmr. HelloWorld` on Windows and `Users/<user>/Library/Preferences/com.aircmr. HelloWorld` on Mac OS.
- Adds a shortcut on your Desktop named after the `filename` element in the descriptor file if you opted to create one from the installation window.

Figure 1-3

On Windows only:

- Adds the application to the Start menu under `AIRCMR/HelloWorld` — the values supplied for the `programMenuFolder` and `filename`, respectively.
- Adds the application titled Hello World — the value supplied as the `name` in the descriptor file — in the Add or Remove Programs list available in the Control Panel.

Navigate to the installation folder. On a Mac, Ctrl+click on the Hello World application and select Show Package Contents from the menu. In the root of this directory are the files specified in the parameters for the ADT when you packaged the AIR file. Also within this directory is the application's executable file and the application descriptor file, which has been renamed to `application.xml` and placed within the `META_INF/AIR` subdirectory.

The *application directory* and *application storage directory* can be referenced using the `File` class of the AIR API. These directories each serve their own purpose in creating and accessing resources for your applications at runtime, a topic discussed later in this book.

Uninstalling the Hello World Application

Why uninstall the Hello World application? Because you live on the edge, and you want to save that disk space for something grander. Fortunately enough, if you want to reinstall it, the AIR file is still available in the project's development directory.

To uninstall an AIR application on Windows, follow these steps:

1. Click Start ➪ Control Panel ➪ Add or Remove Programs.
2. Within the entry list, scroll to the AIR application you wish to uninstall and click once to select it.
3. Click the Remove button. Doing so will remove the installation directory from Program Files and remove the Desktop and Start Menu shortcuts you may have opted to create during the installation process.

To uninstall an AIR application on a Mac, all you have to do is drag the installation folder to the Trash and then empty the Trash.

Uninstalling the application removes the application directory and any ancillary files you opted to create through the installation window. Uninstalling with these processes does not, however, remove the application storage directory or any files created and associated with the application after installation.

Good-bye, Hello World! You are now armed with the tools and knowledge to start building and distributing AIR applications. You also have a structured development environment to make that process run smoothly.

The next section addresses adding improvements to the development process by introducing an Apache ANT build scheme and a simple debug panel available from the code examples for this chapter on the accompanying website.

Extra Tools

Some developers are just too busy to type multiple options into a command line. To automate the processes of deploying, debugging, and packaging discussed previously in this chapter, consider using Apache ANT, a popular tool for that purpose and a perfect fit for your build process, as it is cross-platform and requires Java, which is already on your machine. If you are more comfortable using another tool for automating the build process, you are encouraged to do so. However, if you are interested in setting up a build process through ANT, the following section will walk you through that process.

You may also find that running the Flash Debugger each time you launch an application to debug code can present a time-consuming obstacle within your development. As such, the concept of a simple debug panel that communicates to an AIR application using the `LocalConnection` class of the API is discussed. The code for the simple debug panel and the ANT build file can be found in the code examples for this chapter on the accompanying website.

ANT Build

A few chapters could easily be devoted to ANT alone, but in keeping with AIR application development, this section provides a brief overview of ANT build markup and not necessarily the intricate workings of the tool. Although this solution is presented to you for ease of development, this might not be your cup of tea. Thus, all examples within this book have been run using the command-line tools described previously in this chapter.

Apache ANT (from here on referred to as ANT) is a Java tool that uses XML markup to describe tasks and dependencies in automating the software build process. If you currently do not have ANT installed on your computer, then you will need to install it. Download the latest distribution from `http://ant.apache.org/bindownload.cgi`.

Install ANT in a directory that makes sense with your workflow — for example, `C:/ant-` on Windows. You will also need to add a path to your ANT installation within the Environment Variables.

> *If you work on a Mac, then you may already have ANT installed. Open a Terminal and type* > ***ant -version****. If you are presented with a ''command not found'' message, then either you don't have ANT installed on your machine or you don't have a path to the installation* `/bin` *directory set. If this is the case, then install ANT in the* `/usr/local` *directory and perform the following steps.*

To add a path to your ANT installation on Windows, follow these steps:

1. Click Start ➪ Control Panel ➪ System.
2. Select the Advanced tab from the menu.
3. Click on Environment Variables.
4. Within the System variables grid, navigate to and double-click on Path.
5. In the Variable Value field, add a semicolon (;) and then the path to the `/bin` folder within your ANT installation directory. For the purposes of this example, that is `;C:\ant\bin`.

Follow these steps to add a path to your ANT installation on Mac:

1. Open the terminal and type **open -e .profile**.
2. Add a colon (:) and the path to the `/bin` folder within your ANT installation directory. For the purposes of this example, that is `/usr/local/ant/bin`.

Running ANT from the command line requires a build file within the directory in which it is called. The ANT build file is an XML markup describing tasks and dependencies for the build process. Property values can also be set in the build file, enabling you to refer to the value of a property using the `${propertyname}` construct within tasks. A property file can also be set in the build file to separate out the property-value pairs from the build markup.

First create the build file as it relates to the Hello World application you built previously in this chapter:

1. Enter the following markup in any text editor:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- ====================================================================== -->
<!-- AIR Application Build File (Ant)                                       -->
<!-- ====================================================================== -->

<project name="AIR Application" default="main" basedir=".">

    <property name="app.name" value="HelloWorld" />
    <property name="desc.name" value="HelloWorld-app" />
```

```
<property name="store.type" value="pkcs12" />
<property name="cert.type" value="1024-RSA" />
<property name="cert.name" value="certificate.pfx" />
<property name="cert.pass" value="password" />

<property name="src.dir" value="." />
<property name="deploy.dir" value="." />

<target name="main" depends="init,taskInput,compile,launch,package" />

<target name="init" description="Sets properties based on OS.">
    <condition property="amxmlc.exec" value="amxmlc.bat">
        <os family="windows" />
    </condition>
    <condition property="amxmlc.exec" value="amxmlc">
        <os family="mac" />
    </condition>
    <condition property="adt.jar" value="C:/flex_sdk_3/lib/adt.jar">
        <os family="windows" />
    </condition>
    <condition property="adt.jar" value="/Applications/flex_sdk_3/lib/adt.jar">
        <os family="mac" />
    </condition>
</target>

<target name="taskInput" description="Presents task options.">
    <input message="Please select a task..." validargs="compile,launch,package"
        addproperty="task.action" />
    <condition property="do.compile" value="true">
        <or>
            <equals arg1="${task.action}" arg2="compile" />
            <equals arg1="${task.action}" arg2="launch" />
            <equals arg1="${task.action}" arg2="package" />
        </or>
    </condition>
    <condition property="do.launch" value="true">
        <equals arg1="${task.action}" arg2="launch" />
    </condition>
    <condition property="do.package" value="true">
        <equals arg1="${task.action}" arg2="package" />
    </condition>
</target>

<target name="compile" depends="taskInput" description="Generates SWF.">
    <exec executable="${amxmlc.exec}">
        <arg line="-output ${deploy.dir}/${app.name}.swf" />
        <arg line="${src.dir}/${app.name}.mxml" />
    </exec>
</target>

<target name="launch" if="do.launch" depends="compile"
    description="Launmches application.">
    <exec executable="adl">
        <arg line="${deploy.dir}/${desc.name}.xml"/>
```

```
        </exec>
    </target>

    <target name="package" if="do.package" depends="compile"
        description="Packages AIR application.">
        <antcall target="create.certificate" />
        <java jar="${adt.jar}" fork="true" failonerror="true"
            inputstring="${cert.pass}">
            <arg value="-package" />
            <arg value="-storetype" />
            <arg value="${store.type}" />
            <arg value="-keystore" />
            <arg value="${cert.name}" />
            <arg value="${app.name}.air" />
            <arg value="${desc.name}.xml" />
            <arg value="${app.name}.swf" />
        </java>
    </target>

    <target name="create.certificate" if="do.package"
        description="Creates self signed certificate.">
        <java jar="${adt.jar}" fork="true" failonerror="true">
            <arg value="-certificate" />
            <arg value="-cn" />
            <arg value="${app.name}" />
            <arg value="${cert.type}" />
            <arg value="${cert.name}" />
            <arg value="${cert.pass}" />
        </java>
    </target>

</project>
```

2. Save the file as `build.xml` in the development directory of the Hello World project.

Three main execution tasks are associated with the ANT build file: `compile`, `launch`, and `package`. The type of task to run depends on the user input of the `taskInput` task. The `compile` task is always called, as the success of the `launch` and `package` tasks depends on the existence of a generated SWF file.

Along with these three main tasks that can be run based upon input is a task to create a self-signed certificate prior to packaging, and a task to set property values based on the operating system. The `init` task is run in order to set the OS-specific properties needed for running the command-line tools of the Flex 3 SDK. To run a `.bat` file within the `exec` ANT task on Windows, the extension needs to be included. On Mac OS X, the name of the file is sufficient. In order to run the ADT using the `java` directive, the path to the `.jar` file is needed. As such, a `condition` task is run to evaluate the correct path based on your operating system.

To run this build, open a command prompt, navigate to the development directory, and enter the following command:

```
>ant
```

After running this command, you will be prompted to input the task you wish to perform — `compile`, `launch`, or `package`. Each task will generate a SWF file. If you choose the `launch` task, the AIR application will then be run using the AIR Debug Launcher (ADL). If you choose to package your AIR application, then after the `package` task is complete, you will find a `certificat.pfx` file and a `HelloWorld.air` file within the development directory.

The property settings in this build file are pretty basic and may not suit larger projects. Play around with your own build file as it suits your needs per project. These files can be moved from one development directory to the next without affecting the automation process, though it is recommended that you change any application-specific property values.

Simple Debug Panel

If your debugging tasks are simply logging- or trace-related and don't rely on stepping through breakpoints or handling faults, then you may find it difficult to find the time to open a separate command prompt to launch the Flash Debugger. Creating a simple debug panel to handle all your `trace` statements would be an excellent example of an AIR application that is quick to create and extremely handy when it comes time to debug any of your other AIR applications.

Take what you have learned from this chapter and make your own simple debug panel — in fact, one is available in the code examples for this chapter on the accompanying website. The source is included and you may modify the code as you see fit. One possible addition is a write-to-log feature if you are feeling adventurous. The SimpleDebugPanel application does not employ anything specific to the AIR API and uses `LocalConnection` to pass logging statements from your application to the debug panel running under the AIR runtime. The `README` file included in the directory outlines correct usage in your applications, as you need to add some class files to your projects in order for the debug panel to receive log statements.

Summary

In this chapter you built your first AIR project, a simple Hello World application, and learned about the command-line tools provided in the AIR SDK installation. Also included in this chapter were some time-saving solutions for building and debugging applications. In the following chapters you will build applications that are practical in real-world scenarios — and, it is hoped, have some fun while doing so.