# Chapter 1

# About Flex 3

Flex 3 is the most recent version of a platform for developing and deploying software applications that run on top of the Adobe Flash Player. While such tools have existed for many years, the most recent toolkit from Adobe Systems allows programmers with object-oriented backgrounds to become productive very quickly using the skills they already have learned in other programming languages and platforms.

Since the release of Flex 2, the Flex development environment has encouraged a development workflow similar to that used in other desktop development environments such as Visual Studio, Delphi, and JBuilder. The developer writes source code and compiles an application locally and then uploads the finished application to a Web server for access by the user. That isn't how Flex started, however.

Flex was originally released by Macromedia as a server-based application deployment and hosting platform. In the early versions of the Flex product line, an MXML/ActionScript compiler was included in a Java-based Web application hosted on a Java 2 Enterprise Edition (J2EE) server. Application source code was stored on the server. When a user made a request to the server, the application was compiled "on request" and delivered to the user's browser, and hosted by the Flash Player.

This server-based compilation and application deployment model is still available in the most recent version of the server software now known as LiveCycle Data Services ES. But the version of the compiler that's delivered in LiveCycle Data Services isn't necessarily the same as the one that's available in both the Flex 3 Software Developers Kit (SDK) and Flex Builder 3. And most developers find it simpler to use the primary "local compilation" development model.

## IN THIS CHAPTER

In this chapter, I describe the nature of Flex applications, the relationship between Flex applications and the Flash Player, and how Flex leverages the nearly ubiquitous distribution of Flash Player on multiple operating systems. I also describe how Flex applications can be packaged for deployment as desktop applications using the Adobe Integrated Runtime (AIR), formerly known as Apollo.

## Learning the Fundamentals of Flex

The Flex product line allows developers to deploy applications that run on the Flash Player as Web applications and on the Adobe Integrated Runtime (AIR) as desktop applications. The compiled applications that you create with Flex are the same as those produced by the Adobe Flash authoring environment (such as Adobe Flash CS3), but the process of creating the applications is very different.

### Getting to know Flex applications

A Flex application is software that you create using the various pieces of the Adobe Flex 3 product line, which includes the following:

- The Flex 3 Software Developers Kit (SDK)
- Flex Builder 3

One major difference between the SDK and Flex Builder is that the SDK is free, while Flex Builder is available only through a license that you purchase from Adobe Systems. But in addition to the Flex SDK that's at the core of Flex Builder, the complete development environment includes many tools that will make your application development more productive and less error-prone than working with the SDK and another editing environment.

Flex Builder 3 Professional (the more complete and expensive of the available Flex Builder licenses) also includes a set of components known as the Data Visualization Toolkit that aren't included in the SDK. The Data Visualization Toolkit includes the Flex Charting components for presenting data as interactive visual charts and a new component called the `AdvancedDataGrid` that presents relational data with groups, summaries, multi-column sorting, and other advanced features.

NEW FEATURE **The Flex Charting Controls were available as a separately licensed product in the Flex 2 product line. With Flex 3, the Charting Controls, the `AdvancedDataGrid` component, and other advanced controls are now available only as part of a Flex Builder 3 Professional license.**

#### Flex programming languages

Flex 3 applications are written using two programming languages — ActionScript 3 and MXML:

- **ActionScript 3** is the most recent version of the ActionScript language to evolve in the Flash authoring environment over the lifetime of the product. A complete object-oriented language, ActionScript 3 is based on the ECMAScript edition 4 draft language specification. It includes most of the elements of object-oriented languages, including class definition syntax, class package structuring, strong data typing of variables, and class inheritance.

## Flex as Open Source

In April 2007, Adobe Systems announced its intention to migrate the Flex SDK to an open-source project, to be licensed under the Mozilla Public License (MPL). This license allows developers to modify and extend source code, and to distribute components of the code (or the entire SDK). Any changes that developers make to the ActionScript files that make up the Flex SDK must in turn be made available to other developers. This does not affect the developer's own proprietary code. You still own the MXML and ActionScript code you write for your own applications.

Not all components in the Flex SDK are available in the open-source package. Some components, such as the Flex Charting Components and `AdvancedDataGrid`, are available only through commercial licenses. Also, Flex Builder is available only through a license that you purchase from Adobe.

The open-source Flex SDK is managed through the `http://opensource.adobe.com/wiki/display/flexsdk/` Web site. Additional information and ongoing discussion of the Flex open-source project is available at these Web sites:

- `http://groups.google.com/group/flex-open-source`
- `http://flex.org/`

To get a copy of the Mozilla Public License, visit `www.mozilla.org/MPL/`.

- **MXML** is a pure XML-based markup language that is used to define a Flex application and many of its components. Most of the elements in MXML correspond to an ActionScript 3 class that's delivered as part of the Flex class library.

When you compile a Flex application, your MXML code is rewritten in the background into pure ActionScript 3. MXML can be described as a "convenience language" for ActionScript 3 that makes it easier and faster to write your applications than if you had to code completely in ActionScript.

NOTE **ActionScript 3 also is used in the Flash CS3 authoring environment for logical code, creating class definitions, and other programming tasks. Unlike Flex 3, which uses only version 3 of ActionScript, you can create Flash documents in Flash CS3 that use older versions of the language, such as ActionScript 2.**

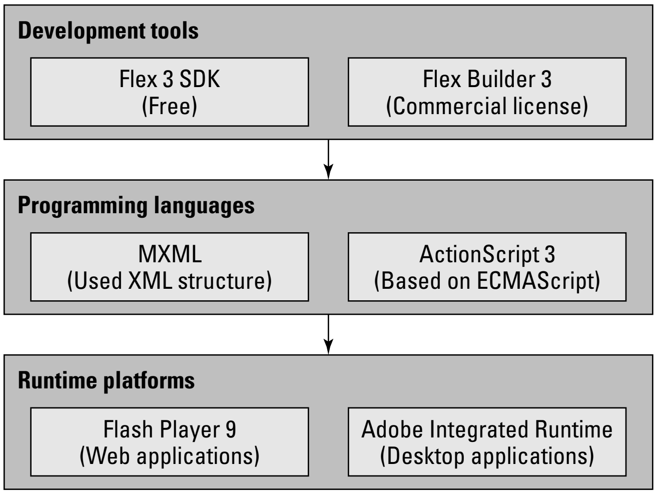
The diagram in Figure 1.1 describes the relationship between the Flex SDK's command-line compiler, Flex Builder, the MXML and ActionScript programming languages, and the Flash Player and Adobe Integrated Runtime.

### MXML versus ActionScript 3

MXML and ActionScript can be used interchangeably in many situations. MXML is commonly used to declare visual layout of an application and many objects, but it's usually your choice as a developer as to when to use each language.

FIGURE 1.1

The Flex SDK and Flex Builder both compile source code in MXML and ActionScript, producing executable applications that are hosted by the Flash Player on the Web or the Adobe Integrated Runtime ("AIR") on the desktop.


Development tools
Flex 3 SDK (Free)
Flex Builder 3 (Commercial license)
Programming languages
MXML (Used XML structure)
ActionScript 3 (Based on ECMAScript)
Runtime platforms
Flash Player 9 (Web applications)
Adobe Integrated Runtime (Desktop applications)

In these examples, I'm declaring an instance of an ActionScript class named `Label`. The `Label` class is part of the Flex class library that's included with both the Flex SDK and Flex Builder 3. Its purpose is to present a single line of text in a Flex application.

### *Declaring objects in MXML*

The `Label` class is represented in MXML as a tag named `<mx:Label/>`. To create an instance of the `Label` class using MXML and set its `text` property to a value of `Hello World`, declare the tag and set the property as an XML attribute:

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

This results in creating an instance of the `Label` class that is displayed in the application.

### *Declaring objects in ActionScript 3*

The `Label` class also can be instantiated using ActionScript 3. When using the ActionScript 3 coding model, you first create the object using the class's constructor method and then add the object to the application's display list so it becomes visible. You can set the `text` property anytime after creating the object:

```
import mx.controls.Label;
var myLabel:Label = new Label();
myLabel.text = "Hello World";
this.addChild(myLabel);
```

This ActionScript code accomplishes exactly the same steps as the MXML code in the first example. Notice that it takes four lines of ActionScript instead of the single line of MXML code. The amount of code needed to accomplish any particular task is a common difference and one of the reasons MXML exists. MXML can significantly reduce the amount of code in your application without compromising its features or performance.

NOTE **Assuming that the ActionScript code above is in a main application file, the prefix `this` in the method call `this.addChild()` would refer to the Application itself. If the same code were in an MXML component or ActionScript class, `this` would refer to the current instance of that component or class.**

## Flex versus Flash development

Developers tend to use Flex instead of Flash when they want to create software applications that have these characteristics:

- High level of interactivity with the user
- Use of dynamic data with application servers such as ColdFusion, ASP.NET, PHP, or J2EE
- Highly scaled applications in terms of the number of views, or screens, from which the user can select

In contrast, developers tend to use Flash when they are creating documents with these characteristics:

- Documents whose main purpose is to present visual animation
- Marketing presentations
- Hosting of Web-based video

Many applications that are built in Flash CS3 could be built in Flex, and vice versa. The selection of development environment, then, is frequently driven by a developer's background and existing skill set.

### Developing in Flash

As described above, developers who use Flash are frequently focused on presenting animation, hosting video, and the like. Flash is generally considered superior for animation work because of its use of a timeline to control presentations over a designated period of time. Flash supports a variety of animation techniques that make use of the timeline, including these:

- Frame by frame animation
- Motion tweening
- Shape tweening

Flash also allows you to create animations using pure ActionScript code, but that approach also can be used in Flex. Developers who come from a graphic design background and are used to thinking visually appreciate the precision and visual feedback that the Flash development environment provides.

One drawback that application developers encounter with Flash is that the primary source document used in Flash, the .fla file format, is binary. As a result, it doesn't work well with the source control systems that application developers commonly use to manage their development projects, because you can't easily "diff," or discover differences between, different versions of a binary file.

### Developing in Flex

Developers who use Flex to build their applications commonly have a background in some other programming language. Documents can be created and made useful in Flash without any programming, but a Flex application is almost entirely code-based. Animations are handled entirely through ActionScript, because Flex doesn't have a timeline as part of its development toolkit.

Flex also has superior tools for handling large-scale applications that have dozens or hundreds of views, or screens. Although Flash CS3 has a screen document feature, this feature hasn't received the development attention from Adobe that would make it a compelling architectural choice for these "enterprise" applications.

Finally, Flex applications are built in source code, which is stored in text files. These text files are easy to manage in source-code control applications such as CVS and Subversion. As a result, multi-developer teams who are dependent on these management tools find Flex development to be a natural fit to the way they already work.

The Flex Builder 3 design view feature has become more friendly and useful to graphic designers than in previous versions, but it isn't always intuitive to a designer who's used to "real" graphic design tools like Adobe's own Photoshop, Illustrator, and Fireworks.

Table 1.1 describes some of the core differences between Flex and Flash development.

**TABLE 1.1**

**Differences between Flex and Flash Development**

| Task | Flex | Flash |
|---|---|---|
| Animation | Flex uses ActionScript classes called Effects to define and play animations. There is no timeline. | The Flash timeline allows animation frame-by-frame or tweening, and also supports programmatic animation with ActionScript. |
| Working with data | Flex has multiple tools for working with data and application servers, including the RPC components (HTTPService, WebService, and RemoteObject). It is also a natural fit for use with LiveCycle Data Services. | Flash can communicate with the same RPC sources as Flex, but its programming tools aren't as intuitive or robust. |

| Task | Flex | Flash |
|---|---|---|
| Design | Flex has a design view for WYSIWYG ("What You See Is What You Get") application layout, but has no visual tools for creating graphic objects from scratch. | Flash has very good graphic design tools, although not as complete a toolkit as Illustrator. However, it has excellent tools for importing and using graphics created in Photoshop and Illustrator. |
| Programming languages | Flex supports ActionScript 3 and MXML. | Flash supports all versions of ActionScript (but only one version per Flash document) and does not support MXML. |
| Code management | Flex applications are created as source code in text files, which are completely compatible with source-code management systems. | Flash documents are binary, which presents problems when building applications in multi-developer environments that require source-code management tools. |

**NOTE** **Applications built for development in the Adobe Integrated Runtime (AIR) can be created in either Flex or Flash. AIR applications can be created from any compiled Flash document or from HTML-based content.**

## Flex and Object-Oriented Programming

Flex application development is especially compelling for developers who are already acquainted with object-oriented programming (OOP) methodologies. Object-oriented programming is a set of software development techniques that involve the use of software "objects" to control the behavior of a software application.

Object-oriented programming brings many benefits to software development projects, including these:

- Consistent structure in application architectures
- Enforcement of contracts between different modules in an application
- Easier detection and correction of software defects
- Tools that support separation of functionality in an application's various modules

You'll find no magic bullets in software development: You can create an application that's difficult to maintain and at risk of collapsing under its own weight in an OOP language just as easily as you can create one that primarily uses procedural programming. But a good understanding of OOP principles can contribute enormously to a successful software development project.

And because ActionScript 3 is a completely object-oriented language, it serves Flex developers well to understand the basic concepts of OOP and how they're implemented in Flex development.

Object-oriented programming is commonly supported by use techniques known as modularity, encapsulation, inheritance, and polymorphism.

## Modularity

*Modularity* means that an application should be built in small pieces, or modules. For example, an application that collects data from a user should be broken into modules, each of which has a particular purpose. The code that presents a data entry form, and the code that processes the data after it has been collected, should be stored in distinct and separate code modules. This results in highly maintainable and robust applications, where changes in one module don't automatically affect behavior in another module.

The opposite of modularity is *monolithic*. In monolithic applications such as the example in Listing 1.1, all the code and behavior of an application are defined in a single source-code file. These applications tend to be highly "brittle," meaning that changes in one section of the application run a high risk of breaking functionality in other areas. Such applications are sometimes referred to as *spaghetti code* because they tend to have code of very different purposes all wrapped around each other.

**LISTING 1.1**

**A monolithic Flex application**

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Model>
    ...data representation...
  </mx:Model>
  <mx:Script>
    ...ActionScript...
  </mx:Script>
  <mx:HBox>
    <mx:DataGrid>
     <mx:columns>
        <mx:DataGridColumn .../>
        <mx:DataGridColumn .../>
        <mx:DataGridColumn .../>
      </mx:columns>
    </mx:DataGrid>
    <mx:Form>
      <mx:FormItem label="First Name:">
        <TextInput id="fnameInput"/>
      </mx:FormItem>
      <mx:FormItem label="Last Name:">
        <TextInput id="lnameInput"/>
      </mx:FormItem>
      <mx:FormItem label="Address:">
        <TextInput id="addressInput"/>
      </mx:FormItem>
    </mx:Form>
  </mx:HBox>
</mx:Application>
```

In the above application, all the application's functionality is mixed together: data modeling, data collection, and logical scripting. Although the application might work, making changes without introducing bugs will be difficult, especially for a multi-developer team trying to work together on the application without constantly disrupting each other's work.

A modular application such as the version in Listing 1.2 breaks up functionality into modules that each handle one part of the application's requirements. This architecture is easier to maintain because the programmer knows immediately which module requires changes for any particular feature.

**LISTING 1.2**

**A modular Flex application**

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script source="scriptFunctions.as"/>
  <valueObjects:AValueObject id="vo"/>
  <views:ADataGrid id="grid"/>
  <forms:AForm id="form"/>
</mx:Application>
```

Flex implements modularity through the use of MXML components and ActionScript classes that together implement the bulk of an application's functionality.

## Encapsulation

*Encapsulation* means that a software object should hide as much of its internal implementation from the rest of the application as possible, and should expose its functionality only through publicly documented "members" of the object. A class definition that's properly encapsulated exposes and documents these object members to allow the application to set properties, call methods, handle events, and refer to constants. The documentation of the object members is known as the application programming interface (API) of the class.

In the Flex class library, class members include:

- **Properties:** Data stored within the object
- **Methods:** Functions you can call to execute certain actions of the object
- **Events:** Messages the object can send to the rest of the application to share information about the user's actions and/or data it wants to share
- **Constants:** Properties whose values never change

In Flex, encapsulation is fully implemented in ActionScript 3. Each member that you define in a class can be marked using an access modifier to indicate whether the particular method or property is `public`, `private`, `protected`, or `internal`. A `public` method, for example, allows the application to execute functionality that's encapsulated within the class, without the programmer who's calling the method having to know the details of how the action is actually executed.

For example, imagine a class that knows how to display a video in the Flash Player and allows the developer to start, stop, and pause the video, and control the video's audio volume. The code that executes these functions would have to know lots about how video is handled in Flash and the particular calls that would need to be made to make the audio louder or softer. The API of the class, however, could be extremely simple, including methods to execute each of these actions.

```
public class VideoPlayer()
{

  public function VideoPlayer(video:String):null
  { ... call video libraries to load a video ... }

  public function start()
  { ... call video libraries to play the video ... }

  public function stop()
  { ... call video libraries to stop the video ... }

  public function setVolume(volume:int):null
  { ... call video libraries to reset the volume ... }

}
```

The application that instantiates and uses the class wouldn't need to know any of the details; it just needs to know how to call the methods:

```
var myVideoPlayer:VideoPlayer = new VideoPlayer("myvideo.flv");
myVideoPlayer.start();
myVideoPlayer.setVolume(1);
```

We say, then, that the `VideoPlayer` class encapsulates complex behavior, hiding the details of the implementation from the rest of the application.

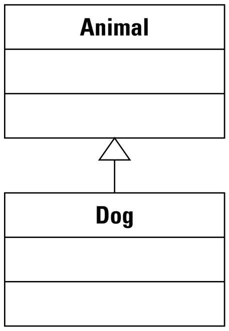
### Inheritance

Inheritance refers to the ability of any class to extend any other class and thereby inherit that class's properties, methods, and so on. An inheritance model allows the developer to define classes with certain members (properties, methods, and so on) and then to share those members with the classes that extend them.

In an inheritance relationship, the class that already has the capabilities you want to inherit is called the *superclass,* or *base class,* or *parent class.* The class that extends that class is known as the *subclass,* or *derived class,* or *child class.* Unified Modeling Language (UML) is a standardized visual language for visually describing class relationships and structures. In this book, I frequently use UML diagrams such as the example in Figure 1.2 to describe how a class is built or its relationship to other classes.

FIGURE 1.2

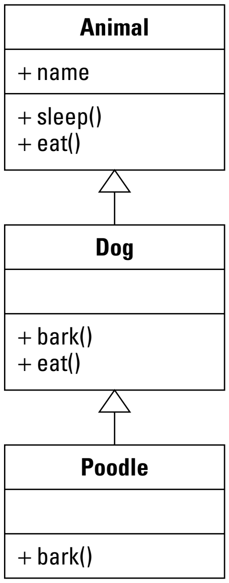
This is an example of a UML diagram that describes a relationship between a base and a derived class.



One class can extend a class that in turn extends another. UML diagrams can be extended to describe these relationships as well. The UML diagram in Figure 1.3 describes a three-tier inheritance relationship between a superclass named `Animal` and subclasses named `Dog` and `Poodle`.

FIGURE 1.3

This diagram describes a three-part inheritance relationship.



In Figure 1.2, methods of the superclass Animal are inherited by the subclass Dog. Dog has additional methods and properties that aren't shared with its superclass and that can override the superclass's existing methods with its own implementations. The same relationship exists between Dog and Poodle.

Because all versions of Animal sleep in the same way, calling Dog.sleep() or Poodle.sleep() actually calls the version of the method implemented in Animal. But because Dog has its own run() method, calling Dog.run() or Poodle.run() calls that version of the method. And finally, because all dogs bark in a different way, calling Poodle.bark() calls a unique version of the bark() method that's implemented in that particular class.

Inheritance allows you to grow an application over time, creating new subclasses as the need for differing functionality becomes apparent.

In Flex, the ActionScript inheritance model allows you to create extended versions of the components included in the Flex class library without modifying the original versions. Then, if an upgraded version of the original class is delivered by Adobe, a simple recompilation of the application that uses the extended class will automatically receive the upgraded features.

## Polymorphism

*Polymorphism* means that you can write methods that accept arguments, or parameters, data typed as instances of a superclass, but then pass an instance of a subclass to the same method. Because all subclasses that extend a particular superclass share the same set of methods, properties, and other object members, the method that expects an instance of the superclass also can accept instances of the subclass and know that those methods can be called safely.

Polymorphism also can be used with a programming model known as an *interface*. An interface is essentially an abstract class that can't be directly instantiated. Its purpose is to define a set of methods and other object members and to describe how those methods should be written. But in an interface such as the one described in Figure 1.4, the method isn't actually implemented; it only describes the arguments and return data types that any particular method should have.

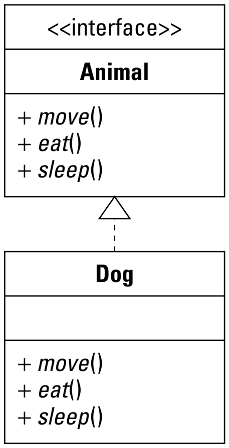
A class "implements" an interface by creating concrete versions of the interface's methods that actually do something. As with the relationship between super and subclasses, a method might be written that accepts an instance of the interface as an argument. At runtime, you actually pass an instance of the implementing class.

For example, you might decide that Animal should be abstract; that is, you would never create an instance of an Animal, only of a particular species. The following code describes the interface:

```
public interface Animal
{
  public function sleep()
  {}
}
```

FIGURE 1.4

This UML diagram describes the relationship between an interface and an implementing class.



The interface doesn't actually implement these methods. Its purpose is to define the method names and structures. A class that implements the interface might look like this:

```
public class Dog implements Animal
{
  public function sleep()
  { ... actual code to make the dog sleep ... }
  public function bark()
  { ... actual code to make the dog bark ... }
}
```

Notice that a class that implements an interface can add other methods that the interface doesn't require. This approach is sometimes known as *contract-based programming*. The interface constitutes a contract between the method that expects a particular set of methods and the object that implements those methods.

Flex supports polymorphism both through the relationship between superclasses and subclasses and through creation and implementation of interfaces in ActionScript 3.

## Understanding the Flash Player

Flex applications are executed at runtime by the Flash Player or the Adobe Integrated Runtime. In either case, they start as applications compiled to the .swf file format.

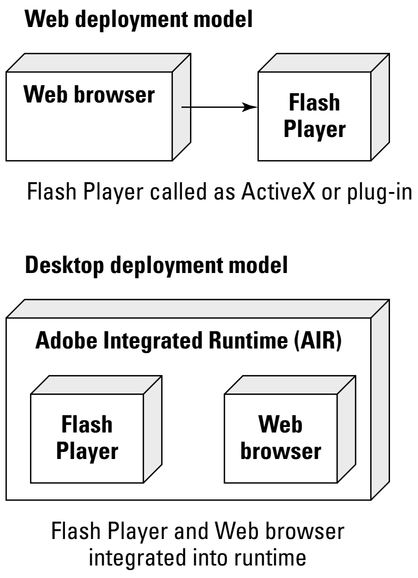
When you deploy a Flex application through the Web, it's downloaded from a Web server at runtime as a result of a request from a Web browser. The browser starts the Flash Player, which in turn runs the application.

The Adobe Integrated Runtime includes the Flash Player as one of its critical components. Other components include a Web browser kernel to execute HTML, CSS and JavaScript, and APIs for local file access and data storage. But the version of the Flash Player that's included with AIR is the same as the one that runs on users' systems as a Web browser plug-in or ActiveX control. As a result, any functionality that you include in a Flex application should work the same regardless of whether the application is deployed to the Web or the desktop.

The diagram in Figure 1.5 describes the architectural difference between the Flash Player's deployment in a Web browser versus the Adobe Integrated Runtime.

**FIGURE 1.5**

Flash Player installed with a Web browser versus the Adobe Integrated Runtime


Web deployment model
Web browser
Flash Player
Flash Player called as ActiveX or plug-in
Desktop deployment model
Adobe Integrated Runtime (AIR)
Flash Player
Web browser
Flash Player and Web browser integrated into runtime

## Learning a little history about the Flash Player

FutureWave Software originally created a product called Future Splash Animator, which in turn evolved from a product called SmartSketch. The player for the animations was Java-based and was the ancestor of the current Adobe Flash Player. After its purchase by Macromedia, the product was renamed and released in 1996 as Macromedia Flash 1.0.

The product went through a steady evolution, starting with basic Web animation and eventually becoming a full-featured programming environment with rich media (video and audio) hosting capabilities.

During its time with Macromedia, Flash (the IDE) was packaged as part the Studio bundle and was integrated with other Studio products such as Dreamweaver and Fireworks. Macromedia positioned Flash MX and MX 2004 as development environments for what the company began to call *rich internet applications* (RIAs). Although the development environment that was Flash never fully satisfied the requirements of application developers (see the discussion in the section "Flex versus Flash development" of issues that are commonly encountered in Flash when developing true applications), the Flash Player continued to grow in its ability to host the finished applications, however they were built.

After Adobe Systems purchased Macromedia, Flash became a part of the Adobe Creative Suite 3 (CS3) product bundles. Along with this rebundling came increased integration with other CS3 products such as Illustrator and Photoshop. Other Adobe products such as AfterEffects and Premiere received new export features that allow their video-based output files to be integrated into Flash-based presentations.

Table 1.2 describes the major milestones in the history of the Flash Player.

**TABLE 1.2**

**Flash Player History**

| Version | Year | New Features |
|---|---|---|
| Macromedia Flash Player 1 | 1996 | Basic Web animation |
| Macromedia Flash Player 2 | 1997 | Vector graphics, some bitmap support, some audio support; object library |
| Macromedia Flash Player 3 | 1998 | The movieclip element; alpha transparency, MP3 compression; standalone player; JavaScript plug-in integration |
| Macromedia Flash Player 4 | 1999 | Advanced ActionScript; internal variables; the input field object; streaming MP3 |
| Macromedia Flash Player 5 | 2000 | ActionScript 1.0; XML support; Smartclips (a component-based architecture); HTML 1.0 text formatting |
| Macromedia Flash Player 6 | 2002 | Flash remoting for integration with application servers; screen reader support; Sorenson Sparc video codec |
| Macromedia Flash Player 7 | 2003 | Streaming audio and video; ActionScript 2; first version associated with Flex |
| Macromedia Flash Player 8 | 2005 | GIF and PNG graphic loading; ON VP6 video codec; faster performance; visual filters including blur and drop shadow; file upload and download; improved text rendering; new security features |
| Adobe Flash Player 9 | 2006 | ActionScript 3; faster performance; E4X XML parsing; binary sockets; regular expressions |
| Adobe Flash Player 9 Update3 | 2007 | H.264 video; hardware-accelerated full-screen video playback |

Each new product bundling and relationship has increased the requirements for the Flash Player. As a result, the most recent version of the Player (version 9) has all the features I've described:

- Object-oriented programming with ActionScript 3
- Web-based animation
- Rich media hosting and delivery

**NOTE** **In addition to the Flash Player that's delivered for conventional computers, Macromedia and Adobe have released versions of Flash Lite for hosting Flash content on devices such as cell phones and PDAs. None of the current versions of Flash Lite support ActionScript 3, so Flex applications currently can't be deployed on those platforms. Undoubtedly, this is a goal of future development by Adobe.**

## Flash Player penetration statistics

One of the attractions of the Flash Player is its nearly ubiquitous penetration rate in the Web. Each new version of the Player has achieved a faster rate of installation growth than each version before it; version 9 is no different. As of December 2007 (according to statistics published on Adobe's Web site), the penetration rate for Flash Player 7 was 99% or greater, Flash Player 8 was at 98% or greater, and Flash Player 9 already had a penetration rate of 93% or greater. Of course, these rates change regularly; for the most recent information on Flash Player penetration rates, visit:

```
http://www.adobe.com/products/player_census/flashplayer/
```

Penetration rates are very important to organizations that are deciding whether to build applications in Flex, because the availability of Flash Player 9 (required to run both Flex applications and Flash documents built with ActionScript 3) determines whether a Flex application will open cleanly or require the user to install or upgrade the Player prior to running the application. If a user needs to install the Flash Player, however, many ways exist to get the job done.

## The Debug Flash Player

The Debug version of the Flash Player differs from the production version in a number of ways. As described in detail below, you can install the debug version of the Flash Player from installers that are provided with Flex Builder 3 and the Flex 3 SDK.

The Debug version of the Player includes these features:

- Integration with `fdb`, the command-line debugger that's included with the Flex 3 SDK
- Integration with Flex Builder debugging tools such as the `trace()` function and breakpoints
- Other debugging tools

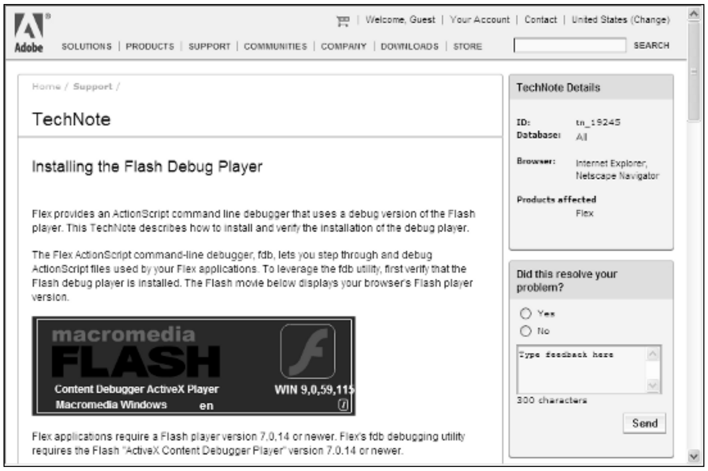
To ensure that you're running the Debug player, navigate to this Web page in any browser that you think has the Player installed:

```
http://kb.adobe.com/selfservice/viewContent.do?externalId=tn_19245
```

As shown in Figure 1.6, you should see a Flash document that tells you which version of the Player is currently installed. When you load this document with the Debug Player, it displays a message indicating that you have the Content Debugger Player. This tool also tells you whether you're running the ActiveX or plug-in Player and what version.

**FIGURE 1.6**

Discovering your Flash Player version


Welcome, Guest | Your Account | Contact | United States (Change)
Adobe
SOLUTIONS | PRODUCTS | SUPPORT | COMMUNITIES | COMPANY | DOWNLOADS | STORE
SEARCH
Home / Support /

TechNote

Installing the Flash Debug Player

Flex provides an ActionScript command line debugger that uses a debug version of the Flash player. This TechNote describes how to install and verify the installation of the debug player.

The Flex ActionScript command-line debugger, fdb, lets you step through and debug ActionScript files used by your Flex applications. To leverage the fdb utility, first verify that the Flash debug player is installed. The Flash movie below displays your browser's Flash player version.

macromedia FLASH
Content Debugger ActiveX Player — WIN 9,0,59,115
Macromedia Windows — en

Flex applications require a Flash player version 7.0.14 or newer. Flex's fdb debugging utility requires the Flash "ActiveX Content Debugger Player" version 7.0.14 or newer.

TechNote Details

ID: tn_19245
Database: All
Browser: Internet Explorer, Netscape Navigator
Products affected: Flex

Did this resolve your problem?

Yes
No

Type feedback here

300 characters

## Flash Player installation

As of this writing, Flash Player 9 is available for these operating systems:

- Windows
- Mac OS X
- Linux
- Solaris

For up-to-date information about current operating system support, including minimum browser and hardware requirements, visit this Web page:

```
http://www.adobe.com/products/flashplayer/productinfo/systemreqs/
```

The Flash Player can be installed on a user's computer system in a variety of ways:

- As an integrated Web browser plug-in
- As a standalone application
- As part of the Adobe Integrated Runtime

**NOTE** **Regardless of how you install the Flash Player, users who install the Flash Player must have administrative access to their computer. On Microsoft Windows, this means that you must be logged in as an administrator. On Mac OS X, you must have an administrator password available during the installation.**

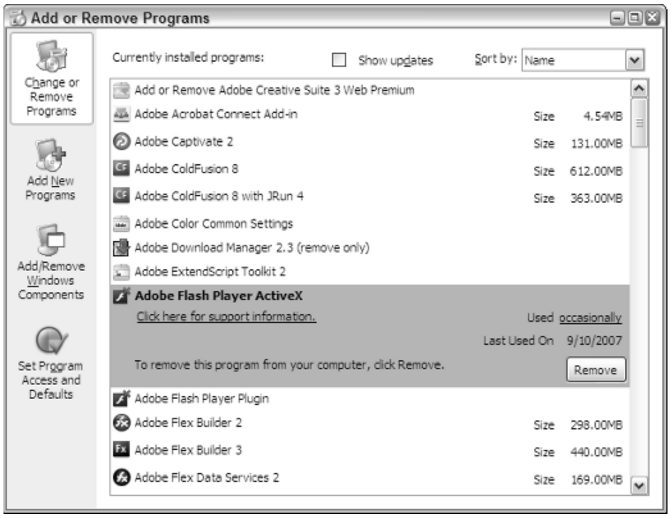
## Uninstalling the Flash Player

Before installing the Flash Player, make sure any existing installations have been removed. The process for uninstalling the Flash Player differs from one operating system to another, but in all cases you must close any browser windows before trying to uninstall the Player

On Windows XP, use the Control Panel's Add or Remove Programs feature, shown in Figure 1.7, and uninstall whatever versions of the Flash Player you find.

FIGURE 1.7

Windows XP's Add or Remove Programs feature, listing both the plug-in and ActiveX versions of the Flash Player


Add or Remove Programs
Change or Remove Programs
Add New Programs
Add/Remove Windows Components
Set Program Access and Defaults
Currently installed programs:
Show updates
Sort by: Name
Remove

| | | |
|---|---|---|
| Add or Remove Adobe Creative Suite 3 Web Premium | | |
| Adobe Acrobat Connect Add-in | Size | 4.54MB |
| Adobe Captivate 2 | Size | 131.00MB |
| Adobe ColdFusion 8 | Size | 612.00MB |
| Adobe ColdFusion 8 with JRun 4 | Size | 363.00MB |
| Adobe Color Common Settings | | |
| Adobe Download Manager 2.3 (remove only) | | |
| Adobe ExtendScript Toolkit 2 | | |
| **Adobe Flash Player ActiveX** | | |
| Click here for support information. | Used | occasionally |
| | Last Used On | 9/10/2007 |
| To remove this program from your computer, click Remove. | | |
| Adobe Flash Player Plugin | | |
| Adobe Flex Builder 2 | Size | 298.00MB |
| Adobe Flex Builder 3 | Size | 440.00MB |
| Adobe Flex Data Services 2 | Size | 169.00MB |

On Mac OS X, use the uninstaller application that's available for download from this Web page:

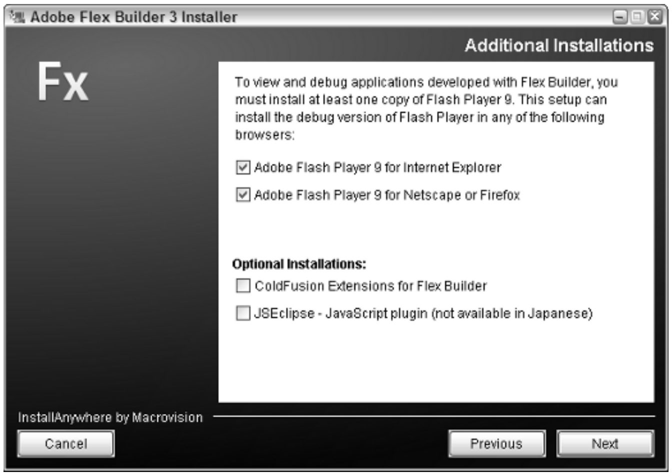
```
www.adobe.com/go/tn_14157
```

## Installation with Flex Builder

As shown in Figure 1.8, when you install Flex Builder 3, you're prompted to install the debug version of the Flash Player as one of the last steps in configuring the installation. You should always accept this part of the installation, because it ensures that your system is equipped with the most recent version of the Player that you need for building, debugging, and testing your Flex applications.

FIGURE 1.8

The Flex Builder installer prompts you to install the Flash Player plug-in or ActiveX control on currently installed browsers.


Adobe Flex Builder 3 Installer
Additional Installations
Fx
InstallAnywhere by Macrovision
Cancel
Previous
Next

To view and debug applications developed with Flex Builder, you must install at least one copy of Flash Player 9. This setup can install the debug version of Flash Player in any of the following browsers:

☑ Adobe Flash Player 9 for Internet Explorer

☑ Adobe Flash Player 9 for Netscape or Firefox

**Optional Installations:**

☐ ColdFusion Extensions for Flex Builder

☐ JSEclipse - JavaScript plugin (not available in Japanese)

Before installing Flex Builder, make sure that you've closed any browser windows. If the installation detects open browser windows, it prompts you to close those windows before continuing the installation process.

## Using Flex Builder installation files

If you need to reinstall the debug version of the Flash Player, you should use the version that's included with Flex Builder 3 or the Flex SDK. If you've installed Flex Builder, you can find the installation files in a subfolder within the Flex Builder installation folder. On Windows, this folder is named:

```
C:\Program Files\Adobe\Flex Builder 3\Player\Win
```

This folder has three files:

- **Install Flash Player 9 Plugin.exe:** The plug-in version for Firefox and Netscape
- **Install Flash Player 9 ActiveX.exe:** The ActiveX control for Internet Explorer
- **FlashPlayer.exe:** The standalone player (does not require installation — just run it!)

Before running any of the installers, be sure to close any open browser windows.

## Installing the Flash Player from the Web

You also can get the Flash Player from the Adobe Web site. Select a download location depending on whether you want the production or debug version of the Player.

### *Downloading the production Flash Player*

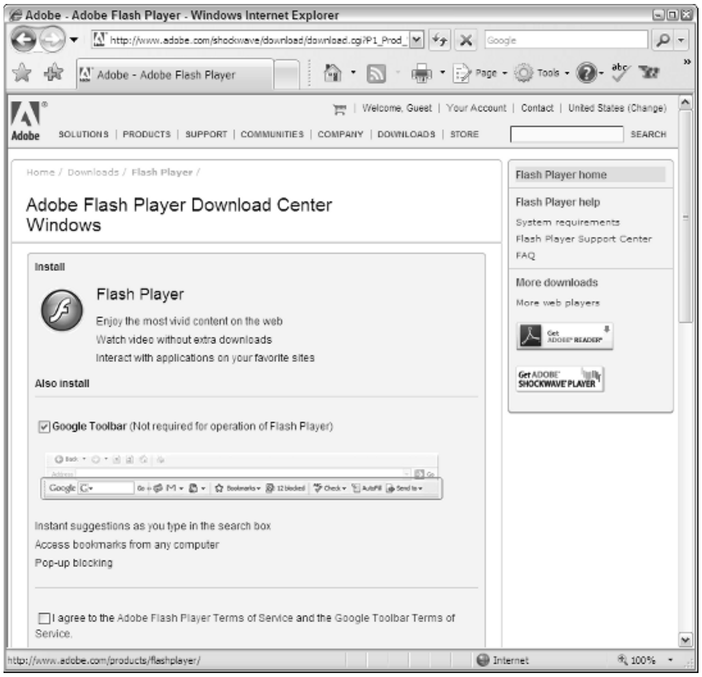
End users who want to run Flex applications and other Flash-based content can download the Flash Player installer from this Web page:

```
http://www.adobe.com/go/getflashplayer
```

When you see the page shown in Figure 1.9, you should see a link to download the Flash Player that's appropriate for your operating system and browser.

**FIGURE 1.9**

Downloading the Flash Player from Adobe.com



Adobe - Adobe Flash Player - Windows Internet Explorer

http://www.adobe.com/shockwave/download/download.cgi?P1_Prod_

Home / Downloads / Flash Player /

Adobe Flash Player Download Center
Windows

Install

Flash Player

Enjoy the most vivid content on the web
Watch video without extra downloads
Interact with applications on your favorite sites

Also install

☑ Google Toolbar (Not required for operation of Flash Player)

Instant suggestions as you type in the search box
Access bookmarks from any computer
Pop-up blocking

☐ I agree to the Adobe Flash Player Terms of Service and the Google Toolbar Terms of Service.

Flash Player home
Flash Player help
System requirements
Flash Player Support Center
FAQ
More downloads
More web players

http://www.adobe.com/products/flashplayer/

Welcome, Guest | Your Account | Contact | United States (Change)
SOLUTIONS | PRODUCTS | SUPPORT | COMMUNITIES | COMPANY | DOWNLOADS | STORE
SEARCH
Get ADOBE READER
Get ADOBE SHOCKWAVE PLAYER
Internet
100%

**CAUTION** **The Flash Player that you download from this page is the production version, rather than the debug version. If you have the production version installed, you can test your applications, but you can't take advantage of debugging tools such as tracing, breakpoints, and expressions evaluation.**

**TIP** **The Flash Player Download Center may include a link to download the Google toolbar or other content. You do not have to download and install this unrelated content in order to get all the features of the Flash Player.**

### *Downloading the debug Flash Player*

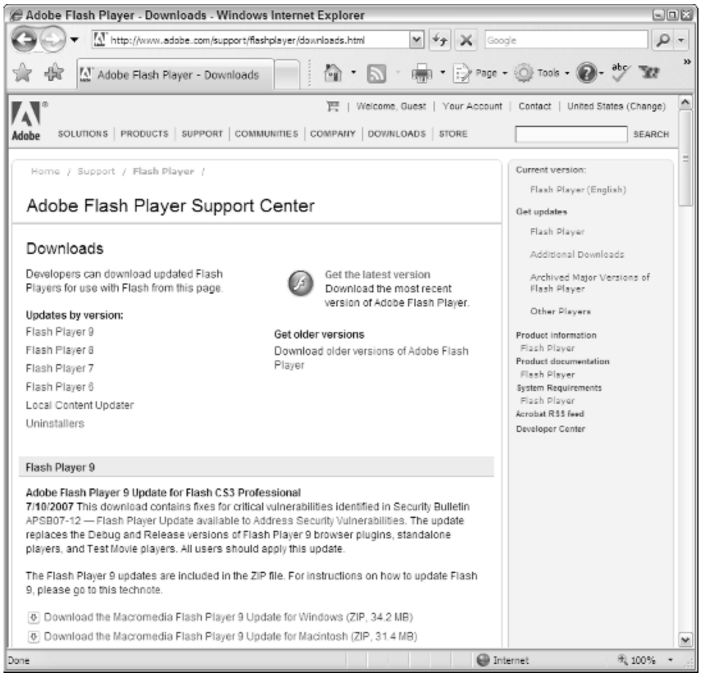
To download the debug version of the Flash Player, visit this Web page:

```
http://www.adobe.com/support/flashplayer/downloads.html
```

As shown in Figure 1.10, you should see links for all versions of the Player, including both debug and production versions for a variety of operating systems and browsers.

**FIGURE 1.10**

This is the Adobe Flash Player Support Center.



Adobe Flash Player - Downloads - Windows Internet Explorer
http://www.adobe.com/support/flashplayer/downloads.html
Google
Adobe Flash Player - Downloads
Page
Tools
Welcome, Guest | Your Account | Contact | United States (Change)
Adobe
SOLUTIONS | PRODUCTS | SUPPORT | COMMUNITIES | COMPANY | DOWNLOADS | STORE
SEARCH
Home / Support / Flash Player /

Adobe Flash Player Support Center

Downloads

Developers can download updated Flash Players for use with Flash from this page.

Get the latest version
Download the most recent version of Adobe Flash Player.

Updates by version:
Flash Player 9
Flash Player 8
Flash Player 7
Flash Player 6
Local Content Updater
Uninstallers

Get older versions
Download older versions of Adobe Flash Player

Current version:
Flash Player (English)
Get updates
Flash Player
Additional Downloads
Archived Major Versions of Flash Player
Other Players
Product information
Flash Player
Product documentation
Flash Player
System Requirements
Flash Player
Acrobat RSS feed
Developer Center

Flash Player 9

Adobe Flash Player 9 Update for Flash CS3 Professional
7/10/2007 This download contains fixes for critical vulnerabilities identified in Security Bulletin APSB07-12 — Flash Player Update available to Address Security Vulnerabilities. The update replaces the Debug and Release versions of Flash Player 9 browser plugins, standalone players, and Test Movie players. All users should apply this update.

The Flash Player 9 updates are included in the ZIP file. For instructions on how to update Flash 9, please go to this technote.

Download the Macromedia Flash Player 9 Update for Windows (ZIP, 34.2 MB)
Download the Macromedia Flash Player 9 Update for Macintosh (ZIP, 31.4 MB)

**TIP** **You might find an even more recent version of the Flash Player on the Adobe Labs Web page at `http://labs.adobe.com`. Adobe Labs hosts projects that are still in development, but that are far enough along that Adobe is sharing the current code with the community.**

## Flex 3 development tools

Flex developers have two sets of development tools: Flex Builder 3 and the Flex 3 SDK.

### Flex Builder 3

Flex Builder 3 is an *integrated development environment* (IDE) for building Flex applications. This is the tool that most developers use to build Flex applications. I describe Flex Builder 3 in detail in Chapter 2.

## The Flex Software Developers Kit (SDK)

The Flex class library and command-line tools you need to build Flex applications are completely free. As long as you don't need to use Flex Builder or certain components that require a license, you can download the Flex SDK from Adobe and build and deploy as many applications as you want. The obvious benefit is the cost. The drawback to this approach is that you'll have to select a text editor such Eclipse that doesn't have the specific support for Flex application development that you get with Flex Builder.

If you decide to use the Flex 3 SDK, download the most recent version from Adobe at `www.adobe.com/go/flex`. The SDK is delivered in a zipped archive file that can be extracted to any platform.

The SDK includes most of the class library you use to build Flex applications. The following components, however, require a license for deployment:

- Flex Charting components
- `AdvancedDataGrid` component
- Application profiling tools

As shown in Figure 1.11, if you decide to use these features without a license, any instances of the charting components or `AdvancedDataGrid` component are displayed in your application with a watermark indicating that you are using an evaluation version of the component.

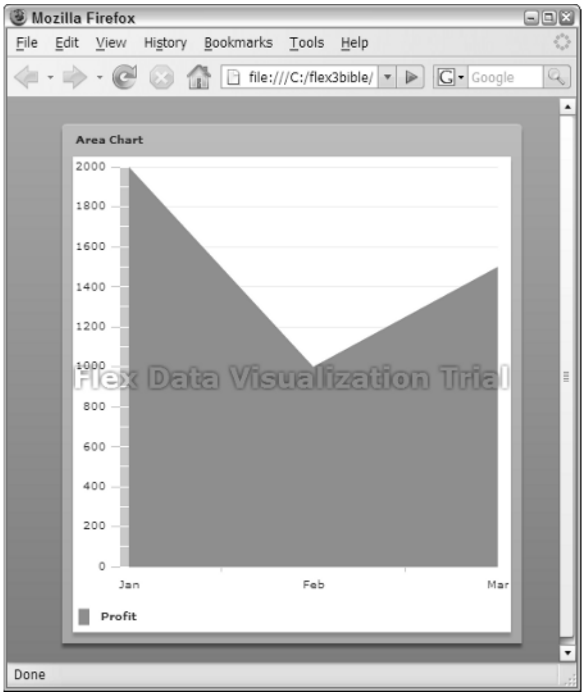
In addition to the Flex class library, the Flex 3 SDK includes these command-line tools:

- `mxmlc`: A compiler for building Flex applications
- `compc`: A compiler for building component libraries, Runtime Shared Libraries (RSLs), and theme files
- `fdb`: A debugger to debug applications
- `fcsh`: The Flex Compiler Shell, which you can use to execute multiple compilation tasks without the overhead of having to launch a new Java Virtual Machine (JVM) for each task
- `amxmlc`: The AIR application compiler
- `acompc`: The AIR component compiler
- `adl`: The AIR debug application launcher
- `optimizer`: A tool for reducing ActionScript compiled file size and creating a "release version" of an application, component, or RSL

Detailed information about how to use each of these command-line tools is available in the Adobe publication *Building and Deploying Flex Applications.*

FIGURE 1.11

A watermarked charting component



## Using MXMLC, the command-line compiler

To compile a Flex application with `mxmlc`, the command-line compiler, it's a good idea to add the location of the Flex 3 SDK bin directory to your system's path. This allows you to run the compiler and other tools from any folder without having to include the entire path in each command. Figure 1.12 shows the command-line compiler.

**TIP** **When you install Flex Builder 3 on Microsoft Windows, the installer provides a menu choice that opens a command window and adds all directories containing Flex 3 components to the current path. To use this tool, select All Programs ➪ Adobe ➪ Adobe Flex 3 SDK Command Prompt from the Windows Start menu.**

To compile an application from the command line, switch to the folder that contains your main application file. If you want to try this using the exercise files that are available for download with this book, switch to the `chapter01` directory:

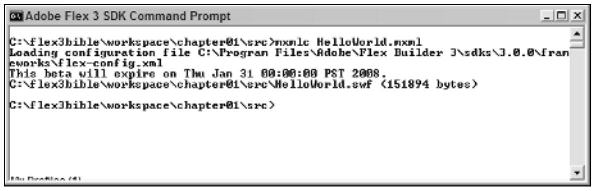
```
cd /flex3bible/chapter01
```

This directory contains a file called `HelloWorld.mxml`, a simple Flex application. To compile the application, run this command:

```
mxmlc HelloWorld.mxml
```

**FIGURE 1.12**

The command-line compiler at work



```
C:\flex3bible\workspace\chapter01\src>mxmlc HelloWorld.mxml
Loading configuration file C:\Program Files\Adobe\Flex Builder 3\sdks\3.0.0\fran
eworks\flex-config.xml
This beta will expire on Thu Jan 31 00:00:00 PST 2008.
C:\flex3bible\workspace\chapter01\src\HelloWorld.swf (151894 bytes)

C:\flex3bible\workspace\chapter01\src>
```

After the compilation is complete, your directory will contain a new file called `HelloWorld.swf`. This is the compiled application that you deploy to your Web server.

**TIP** **The command-line compiler has many options for tuning your application. For complete details on how to use the compiler, see the Adobe publication *Building and Deploying Flex Applications.***

## Getting Help

Documentation for Flex 3 is available from the Adobe Web site at:

```
www.adobe.com/support/documentation/en/flex/
```

The documentation is available in a variety of formats, including Acrobat PDF, HTML Help, and ASDocs HTML files.

The documentation includes these publications, among others:

- *Developing Flex Applications* contains extensive documentation on the functional tools that are available in the Flex framework.
- *Building and Deploying Flex Applications* focuses on application architecture, compiler tools, and deployment strategies.
- *ActionScript 3.0 Language and Components Reference* contains generated documentation of the Flex class library, including each class's properties, methods, and so on. This documentation also includes extensive code samples.

The documentation also is delivered in indexed, searchable format with Flex Builder 3. I describe how to explore and use this version of the documentation in Chapter 2.

## Summary

In this chapter, I gave an introduction to the world of application development with Adobe Flex. You learned the following:

- Flex applications are built as source code and compiled into Flash documents.
- Flex applications can be run as Web applications with the Flash Player, delivered through a Web browser.
- Flex applications also can be run as desktop applications, hosted by the Adobe Integrated Runtime (AIR).
- The Flex Software Developers Kit (SDK) is completely free and available as an open-source project that's managed by Adobe Systems.
- Flex Builder 3 is a commercial integrated development environment for building Flex applications.
- Flex developers tend to have a background in object-oriented software development, but anyone who's willing to invest the time can become proficient in Flex application development.