CHAPTER 1

About Flex 4

Flex 4 is the most recent version of a platform for developing and deploying software applications that run on top of Adobe Flash Player for the Web and Adobe AIR for the desktop. While such tools have existed for many years, the most recent set from Adobe Systems enables 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 desktop development environments such as Visual Studio, Delphi, and JBuilder. The developer writes source code and compiles an application locally and then, for applications designed for deployment from the Web, 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 Enterprise Edition (JEE) 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 a component now known as the Flex Web Tier Compiler. But the version of the compiler that's delivered in the Web Tier Compiler isn't always the same as the one that's included in both the Flex 4 Software Developers Kit (SDK) and the newly renamed Flash Builder 4. 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 Adobe 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 Adobe AIR.

Learning the Fundamentals of Flex

The Flex product line enables developers to deploy applications that run on Flash Player as Web applications and on Adobe AIR as native desktop applications. The compiled applications that you create with Flex are in the same format as those produced by the Adobe Flash authoring environment (such as Adobe Flash CS4), but the process of creating the applications is very different.

Getting to know Flex applications

A Flex application is built as a Flash-based software presentation that you create with the Flex 4 SDK. Most Flex developers create their applications using the Flash Builder 4 integrated development environment product line (formerly named Flex Builder). And a new product from Adobe, Flash Catalyst, helps to bridge the gap between developers who use Flash Builder and designers who use Photoshop or Illustrator to create application designs.

One major difference between the Flex SDK and Flash Builder is that the SDK on its own is free and mostly open source, while Flash 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 Flash 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.

Web Resource

The release version of the Flex SDK is bundled with Flash Builder 4, but you can download and use more recent builds of the SDK from Adobe's open-source Web site at `http://opensource.adobe.com/wiki/display/flexsdk/Flex+SDK`**.** ■

Flash Builder 4 Premium (the more complete and expensive of the available Flash Builder editions) also includes a set of components known as the Data Visualization components that aren't licensed in the open-source Flex SDK. The Data Visualization components include the charting components for presenting data as interactive visual charts and a couple of advanced interactive data-centric components called the `AdvancedDataGrid` and `OlapDataGrid` that present relational data with groups, summaries, multicolumn sorting, and other advanced features.

Note

The Flex Data Visualization components were available as a separately licensed product in the Flex 2 product line. With the release of Flex 3, they became available only as part of the Flex Builder 3 Professional license. The license model for the data visualization components has stayed the same in Flash Builder 4 Premium. ■

Flex as Open Source

In February 2008, Adobe Systems released the Flex SDK as an open-source project, licensed under the Mozilla Public License (MPL), version 1.1. This license enables you to modify and extend source code and to distribute components of the code (or the entire SDK). You must make any changes that you make to the ActionScript files that are part of the Flex SDK available to other developers. This does not affect your own proprietary code. You still own the MXML and ActionScript code you write for your own applications. To get a copy of the MPL, visit `www.mozilla.org/MPL/`.

As described previously, not all components in the Flex SDK are available in the open-source package. Some components, such as the Flex charting components and the advanced data presentation controls, are available only through commercial licenses. Also, Flash Builder is available only through a license that you purchase from Adobe.

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

- `http://flex.org/`
- `http://forums.adobe.com/community/opensource/flexsdk/general`

Flex programming languages

Flex applications are written using three programming languages — ActionScript 3, MXML, and FXG (Flash XML Graphics):

- **ActionScript 3.** 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.
- **MXML.** 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.
- **FXG.** A new XML-based language that enables you to represent graphic objects as XML markup. The new Adobe Flash Catalyst application generates projects that describe functional applications and their graphic presentations in a combination of MXML, FXG, and ActionScript. You can then develop these projects further in Flash Builder 4. MXML includes many vector graphic drawing tags that enable you to declare low-level graphic objects in your Flex applications. These tags are designed to follow the FXG markup language's syntax and element and attribute names. You can also treat complete FXG files as graphical images.

New Feature

FXG (Flash XML Graphics) was created by Adobe as a language that represents graphical objects in a Flex application. Its capabilities closely follow the rendering model of Adobe Flash Player 10. There are many similarities between FXG and SVG (Scalable Vector Graphics), another XML language that represents graphics that's been available for many years. In fact, the Adobe development team first considered using SVG, but decided to create a new language because the existing SVG didn't match how graphics are rendered in Flash Player. Many Adobe Creative Suite products are able to export graphics as FXG markup, including Photoshop, Illustrator, and Fireworks. ■

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

Beginning with Flash CS3 Professional, ActionScript 3 also is used in the Flash authoring environment for logical code, creating class definitions, and other programming tasks. Unlike Flex, which uses only version 3 of ActionScript, you can create Flash documents in Flash 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 AIR.

MXML versus ActionScript 3

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

In these examples, I'm declaring an instance of an ActionScript class named `Label` and setting some *properties* and *styles*. The `Label` class is part of the Flex 4 class library that's included with both the Flex SDK and Flash Builder 4. Its purpose is to present simple text in a Flex application.

New Feature

The `Label` control used in these examples is a member of a new component collection named the Spark components. (Controls and containers used in Flex 2 and 3 are now known as the MX components.) The Spark `Label` and two other new controls named `RichText` and `RichEditableText` are designed to replace the MX `Label` and `Text` controls. ■

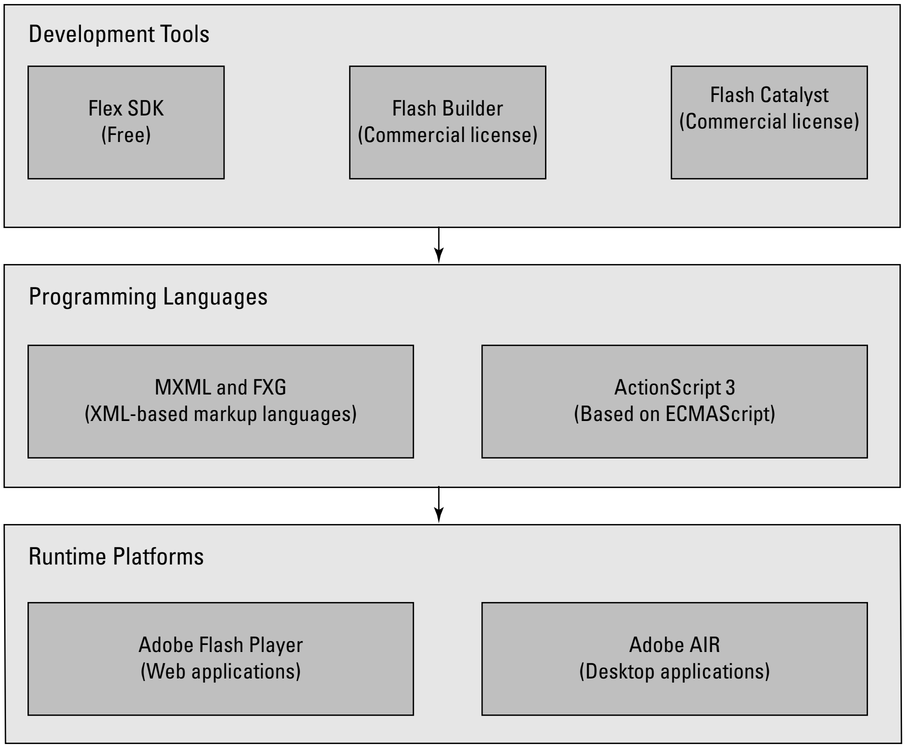
Declaring objects in MXML

The `Label` control is represented in MXML as a tag named `<s:Label/>`. To create an instance of the `Label` control using MXML and set its `text` property to a value of `Hello from MXML`, declare the tag and set the property as an EXtensible Markup Language (XML) attribute. The following example also sets the `fontWeight` and `fontSize` styles to affect the control's appearance.

```
<s:Label id="myMXMLText" text="Hello from MXML"
  fontSize="18" fontWeight="bold"/>
```

FIGURE 1.1

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


Development Tools
Flex SDK (Free)
Flash Builder (Commercial license)
Flash Catalyst (Commercial license)
Programming Languages
MXML and FXG (XML-based markup languages)
ActionScript 3 (Based on ECMAScript)
Runtime Platforms
Adobe Flash Player (Web applications)
Adobe AIR (Desktop applications)

Cross-Reference

The XML namespace prefix `s:` refers to the new Spark namespace that's declared at the top of all new Flex 4 applications:

```
xmlns:s="library://ns.adobe.com/flex/spark"
```

I describe this and other new Flex 4 namespaces in Chapter 4. ■

Declaring objects in ActionScript 3

You can instantiate and add `Label` and other controls to the application's layout using ActionScript 3. When using this coding model, you first declare the object as a variable. If you want the reference to the object to persist, you declare it outside of any functions. You then instantiate the object using the class's constructor method and set its properties and styles, and add the object to the application's content group so it becomes visible.

You can set the control's properties and styles anytime after creating the object:

```
import mx.events.FlexEvent;
import spark.components.Label;
protected var myActionScriptText:Label;
protected function creationCompleteHandler(event:FlexEvent):void
{
  myActionScriptText = new Label();
  myActionScriptText.text = "Hello from ActionScript";
  myActionScriptText.setStyle("fontSize", 18);
  myActionScriptText.setStyle("fontWeight", "bold");
  this.contentGroup.addElement(myActionScriptText);
}
```

The preceding ActionScript code accomplishes exactly the same steps as the MXML code in the first example. Notice that it takes many lines of ActionScript, some inside a custom function, to replace the MXML declaration. This difference in the amount of code needed to accomplish any particular task is one of the reasons MXML exists. MXML can significantly reduce the amount of code in your application without compromising its features or performance.

New Feature
The new Flex 4 `Application` container referenced with the `<s:Application>` tag places its child objects in a `Group`, which is addressed by the container's `contentGroup` property. This new `Application` container is also a member of the new Spark component collection. ■

Note
Assuming that the previous ActionScript code is in a main application file, the prefix `this` in the method call `this.contentGroup.addElement()` refers 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

The line separating the terms *Flex* and *Flash* has changed over the years. As I described previously, Flex originally referred to the entire product line: the class library, compilers, development tool, and server environment. The original Flex server is now named LiveCycle Data Services, and the development tool formerly known as Flex Builder is now named Flash Builder, because it's used to create and edit ActionScript code by both Flash and Flex developers. In this discussion, I use the term Flash to refer to the visual authoring environment known as Flash Professional, and not the recently renamed Flash Builder. And I use the term Flex to refer primarily to the Flex 4 SDK.

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

- Projects built by multi-developer teams
- High level of interactivity with the user

- Use of dynamic data with application servers such as ColdFusion, PHP, or JEE
- 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 can be built in Flex, and vice versa. Your choice of development tools is frequently driven by your background and existing skill set.

Developing in Flash

As I described earlier, 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:

- Frame by frame animation
- Motion tweening
- Shape tweening
- Inverse kinematics

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

The format of the primary source document used in Flash, the FLA file, is binary rather than text-based. As a result, it doesn't work well in multi-developer environments, where source-code management systems are commonly use to manage code. You can't easily *diff*, or discover differences between, different versions of a binary file. In these environments, it's common to move as much ActionScript code to external text-based files as possible, even when a project's primary format is built in Flash. In Flash CS5 Professional and Flash Builder 4, Adobe has now made it much easier to move between the products. And starting with Creative Suite 5, Flash Builder 4 is now included with the Web Premium software bundle that also includes Dreamweaver, Fireworks, and other Web-based development tools.

Developing in Flex

Developers who use Flex to build applications frequently have a background in some other programming language. Presentations can be created and made useful in Flash without any programming, but a Flex application is almost entirely code-based. Animations are created entirely through ActionScript, because neither the Flex SDK nor Flash Builder has a timeline as part of their development toolkits.

Flex also has superior tools for handling large-scale applications that have dozens or hundreds of views, or screens. Flash CS3 had a screen document feature, but the feature didn't receive the development attention from Adobe that would have been required to make it a compelling architectural choice for these "enterprise" applications. The feature was removed in Flash CS4.

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 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.

Flash Builder's design view feature has become increasingly friendly and useful to graphic designers, but it isn't always intuitive to a designer who's used to "real" graphic design tools like Adobe's Photoshop, Illustrator, and Fireworks. The introduction of Adobe Flash Catalyst, a new graphic design application that supports creation of graphically rich compositions for Flex applications, now enables graphic designers to participate as full partners in Flex application development.

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 application servers as Flex, but its programming tools aren't as intuitive or robust.
Design	Flash Builder has a design view for WYSIWYG (What You See Is What You Get) application layout but doesn't have visual tools for creating graphic objects from scratch. The new Adobe Flash Catalyst enables designers to import compositions from PhotoShop and Illustrator and transform them into Flex applications that can be developed further in Flash Builder.	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.

Task	Flex	Flash
Programming languages	Flex 4 and Flash Builder 4 support ActionScript 3 and MXML for component definition and instantiation, and FXG to declare low-level graphics.	Flash Professional CS4 supports all versions of ActionScript (but only one version per Flash document) but 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

Flash-based applications built for desktop deployment with Adobe AIR can be created in either Flash Builder with the Flex SDK or in Flash Professional. 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.

OOP brings many benefits to software development projects, including:

- 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.

OOP 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 wrapped around each other.

LISTING 1.1

A monolithic Flex application

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/mx">
  <fx:Script>
    <![CDATA[
      import mx.collections.ArrayCollection;
      [Bindable]
      private var myData:ArrayCollection;
      ...additional ActionScript code...
    ]]>
  </fx:Script>
  <s:VGroup>
    <mx:DataGrid dataProvider="{myData}">
      <mx:columns>
        <mx:DataGridColumn/>
        <mx:DataGridColumn/>
        <mx:DataGridColumn/>
      </mx:columns>
    </mx:DataGrid>
    <mx:Form>
      <mx:FormItem label="First Name:">
        <s:TextInput id="fnameInput"/>
      </mx:FormItem>
      <mx:FormItem label="Last Name:">
        <s:TextInput id="lnameInput"/>
      </mx:FormItem>
      <mx:FormItem label="Address:">
```

```
        <s:TextInput id="addressInput"/>
      </mx:FormItem>
    </mx:Form>
  </s:VGroup>
</s:Application>
```

In the previous 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, each of which handles 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"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:valueObjects="valueObjects.*"
  xmlns:views="views.*"
  xmlns:forms="forms.*">
  <fx:Script source="scriptFunctions.as"/>
  <valueObjects:AValueObject id="vo"/>
  <views:ADataGrid id="grid"/>
  <forms:AForm id="form"/>
</s: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 enable 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:

- **Constants.** Properties whose values never change.
- **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.
- **Methods.** Functions you can call to execute certain actions of the object.
- **Properties.** Data stored within the object.
- **Skin Parts.** A part of a Spark component that displays a part of the component and can be modified in a custom skin.
- **Skin States.** A view state that a component reacts to by displaying, hiding, or changing parts of the component's visual presentation.
- **Styles.** Visual characteristics of an object that determine its appearance.

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, enables any part of 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 will have to know a lot about how video is handled in Flash and the particular calls that will 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 with very simple calls from the main application, like this:

```
public class VideoPlayer()
{
  public function VideoPlayer(videoFile:String):void
  { ... call video libraries to load a video ... }
  public function start():void
  { ... call video libraries to play the video ... }
  public function stop():void
  { ... call video libraries to stop the video ... }
  public function setVolume(volume:int):void
  { ... 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.

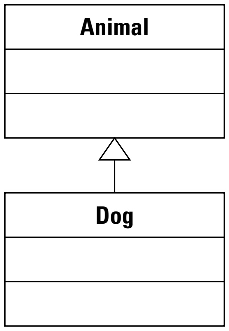
Inheritance

Inheritance refers to the capability of any class to extend any other class and thereby inherit that class's properties, methods, and so on. An inheritance model enables you 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, base class,* or *parent class.* The class that extends that class is known as the *subclass, 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 shown 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.



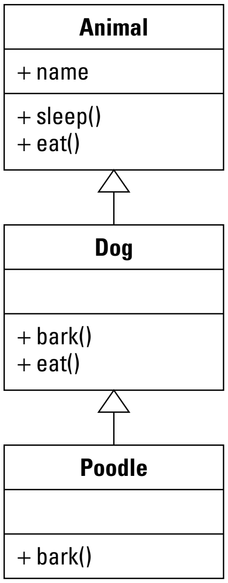
One class can extend a class that in turn extends another. UML diagrams can be extended to describe these relationships as well. The UML diagram shown in Figure 1.3 describes a three-tier inheritance relationship between a superclass named `Animal` and subclasses named `Dog` and `Poodle`.

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 `eat()` method, calling `Dog.eat()` or `Poodle.eat()` 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.

FIGURE 1.3

This diagram describes a three-part inheritance relationship.



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

In Flex, the ActionScript inheritance model enables 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.

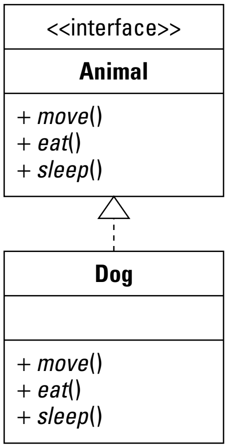
Polymorphism

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 shown 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.

FIGURE 1.4

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



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()
  {}
}
```

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 Adobe Flash Player

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

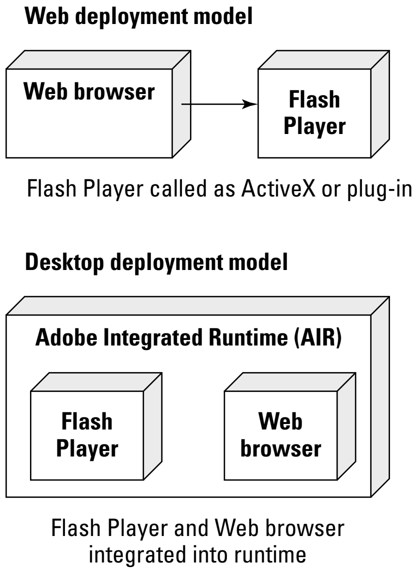
When you deploy a Flex application through the Web, it's downloaded from a Web server at runtime as a result of a request from a Web browser. The browser starts Adobe Flash Player, which in turn runs the application.

Adobe AIR includes Flash Player as one of its critical components. Other components include a Web browser kernel to execute HTML (Hypertext Markup Language), CSS (Cascading Style Sheets), and JavaScript, and APIs for local file access and persistent data storage. The version of 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 shown in Figure 1.5 describes the architectural difference between Flash Player's deployment in a Web browser versus AIR.

FIGURE 1.5

Flash Player installed with a Web browser versus AIR


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 Adobe Flash Player history

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 authoring tool) 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 of issues that are commonly encountered in Flash when developing true applications in the section on Flex versus Flash development in this chapter), Flash Player continued to grow in its capability 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 Creative Suite products such as Illustrator and Photoshop. Other Adobe products such as After Effects and Premiere include new export features that enable their video-based output files to be integrated into Flash-based presentations. First introduced with Flash Professional CS4 in 2008, Flash Player 10 offers many new features, along with improved runtime performance.

Table 1.2 describes the major milestones in the history of Adobe 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; stand-alone 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

continued

TABLE 1.2 *(continued)*

Version	Year	New Features
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	Graphical user interface (GIF) and portable network graphic (PNG) 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 Update 3 (version 9.0.28)	2007	H.264 video; hardware-accelerated full-screen video playback
Adobe Flash Player 10	2008	3D effects; custom filters and effects; advanced text rendering; dynamic sound generation; vector data type; dynamic streaming; Speex audio codec; enhanced file upload and download APIs; color correction
Adobe Flash Player 10.1	2010	The first release as part of the Open Screen Project. First version on cell phones to support ActionScript 3. HTTP Streaming, advanced video delivery.

Each new product bundling and relationship has increased the capabilities of Flash Player. As a result, the most recent version of Flash Player as of this writing (version 10.1) has all the features I've described:

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

Note

In addition to the version of Flash Player that's delivered for conventional computers, Macromedia and Adobe have released Flash Lite for hosting Flash content on devices such as cell phones and PDAs (Personal Digital Assistants). Beginning with Flash CS5 Professional, Adobe Systems Flash Player 10.1 will work on most small devices such as cell phones and support ActionScript 3. In addition, Flash CS5 will enable you to compile Flash presentations as native iPhone applications.

The Flex development team isn't far behind. The Flex Mobile SDK will enable you to create applications for mobile deployment using the Flex application architecture. For more information on this effort, visit the Flex Mobile Web page: `http://labs.adobe.com/technologies/flex/mobile/`. ■

Understanding Flash Player penetration statistics

One of the attractions of Flash Player is its nearly ubiquitous penetration rate in the Web. Each new version of Flash Player has achieved a faster rate of installation growth than each version before it; version 9 is no different. As of December 2009 (according to statistics published on Adobe's Web site), the penetration rate for Flash Player versions 7, 8, and 9 was 98 percent or greater (including in emerging markets), and Flash Player 10 already had a penetration rate of 93 percent or greater. Of course, these rates change periodically; for the most recent Flash Player penetration rates, visit:

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

Penetration rates are important to organizations that are deciding whether to build applications in Flex, because the availability of Flash Player 10 (required to run the most recently published Flex applications and Flash documents) 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, there are many ways to get the job done.

Using the debug version of Flash Player

The *debug* version of Flash Player differs from the production version in a number of ways. As I describe in the following section, you can install the debug version of the Flash Player from installers that are provided with Flex Builder 4 and the Flex 4 SDK.

The debug version of Flash Player includes these features:

- Integration with `fdb`, the command-line debugger that's included with the Flex SDK
- Capability to process and report logging messages issued with the `trace()` function
- Integration with Flash Builder debugging tools such as breakpoints
- Other debugging tools

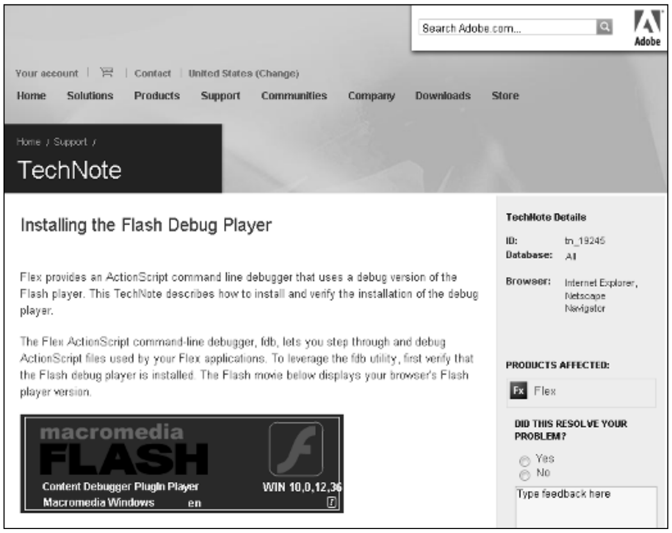
To ensure that you're running the Flash Debug Player, navigate to this Web page in any browser that you think has the Flash Player installed:

```
www.adobe.com/go/tn_19245
```

As shown in Figure 1.6, you should see a Flash document that tells you which version of the Flash Player is currently installed. When you load this document with the Flash 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 Plugin Player and which version.

FIGURE 1.6

Discovering your Flash Player version


Search Adobe.com...
Adobe
Your account | | Contact | United States (Change)
Home Solutions Products Support Communities Company Downloads Store
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 PlugIn Player WIN 10,0,12,36
Macromedia Windows en

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

Flash Player installation

As of this writing, Flash Player 10 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:

```
www.adobe.com/products/flashplayer/systemreqs/
```

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 stand-alone application
- As part of Adobe AIR

Note

Regardless of how you install Flash Player, users who install 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 Flash Player

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

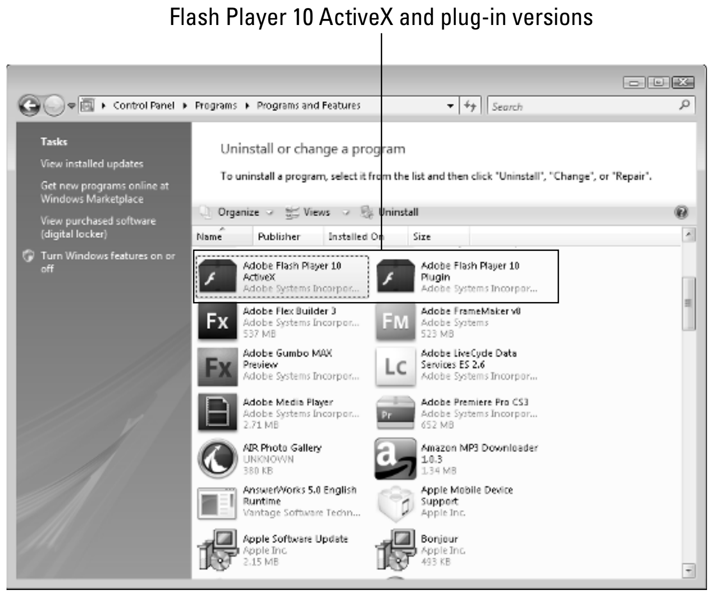
On Windows XP, use operating system's standard tools for uninstalling any software: the Control Panel's Add or Remove Programs feature on Windows XP or Windows Vista's Uninstall or change a program screen (shown in Figure 1.7).

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

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

FIGURE 1.7

Windows Vista's Uninstall or change a program feature, listing both the plug-in and ActiveX versions of the Flash Player


Flash Player 10 ActiveX and plug-in versions
Control Panel ▸ Programs ▸ Programs and Features
Search
Tasks
View installed updates
Get new programs online at Windows Marketplace
View purchased software (digital locker)
Turn Windows features on or off
Organize
Views
Uninstall
Name
Publisher
Installed On
Size

Uninstall or change a program

To uninstall a program, select it from the list and then click "Uninstall", "Change", or "Repair".

Adobe Flash Player 10 ActiveX — Adobe Systems Incorpor...
Adobe Flash Player 10 Plugin — Adobe Systems Incorpor...
Adobe Flex Builder 3 — Adobe Systems Incorpor... — 537 MB
Adobe FrameMaker v8 — Adobe Systems — 523 MB
Adobe Gumbo MAX Preview — Adobe Systems Incorpor...
Adobe LiveCycle Data Services ES 2.6 — Adobe Systems Incorpor...
Adobe Media Player — Adobe Systems Incorpor... — 2.71 MB
Adobe Premiere Pro CS3 — Adobe Systems Incorpor... — 652 MB
AIR Photo Gallery — UNKNOWN — 380 KB
Amazon MP3 Downloader 1.0.3 — 1.34 MB
AnswerWorks 5.0 English Runtime — Vantage Software Techn...
Apple Mobile Device Support — Apple Inc.
Apple Software Update — Apple Inc. — 2.15 MB
Bonjour — Apple Inc. — 493 KB

Installation with Flash Builder

When you install Flash Builder 4, the debug version of Flash Player is installed automatically. To ensure that this part of the installation succeeds, make sure that you've closed any browser windows before you start the installation. If the installation detects open browser windows, it prompts you to close those windows before continuing the installation process.

Using Flash Builder installation files

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

```
C:\Program Files\Adobe\Flash Builder 4\Player\Win
```

This folder has three files:

- **Install Flash Player 10 Plugin.exe.** The plug-in version for Firefox, Safari, and other browsers.
- **Install Flash Player 10ActiveX.exe.** The ActiveX control for Internet Explorer.
- **FlashPlayer.exe.** The stand-alone player (does not require installation — just run it!).

Installing 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://get.adobe.com/flashplayer/
```

When you see the page shown in Figure 1.8, you should see a link to download the Flash Player that's appropriate for your operating system and browser.

Caution

The version of 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 might include a link to download the Google toolbar or other content. You do not have to download and install this unrelated content to get all the features of the Flash Player. ■

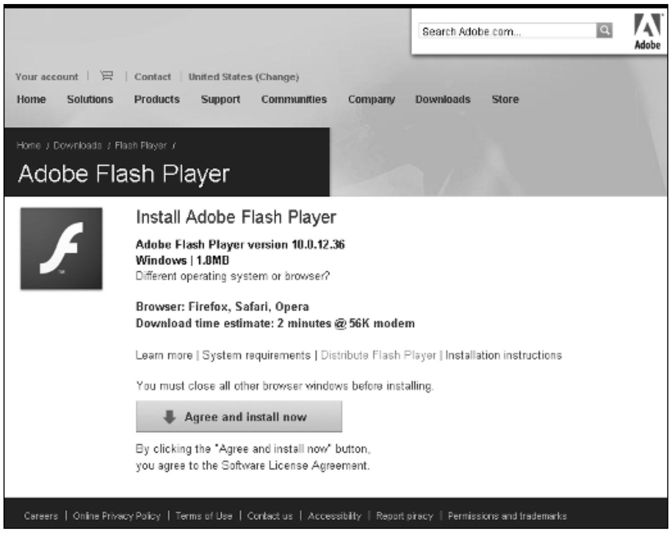


FIGURE 1.8

Downloading Flash Player from Adobe.com

Search Adobe.com...
Adobe
Your account | | Contact | United States (Change)
Home Solutions Products Support Communities Company Downloads Store
Home / Downloads / Flash Player /
Agree and install now
Careers | Online Privacy Policy | Terms of Use | Contact us | Accessibility | Report piracy | Permissions and trademarks

Adobe Flash Player

Install Adobe Flash Player

Adobe Flash Player version 10.0.12.36
Windows | 1.8MB
Different operating system or browser?

Browser: Firefox, Safari, Opera
Download time estimate: 2 minutes @ 56K modem

Learn more | System requirements | Distribute Flash Player | Installation instructions

You must close all other browser windows before installing.

By clicking the "Agree and install now" button, you agree to the Software License Agreement.

Downloading the debug version of Flash Player

You can download the debug version of Flash Player from this Web page:

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

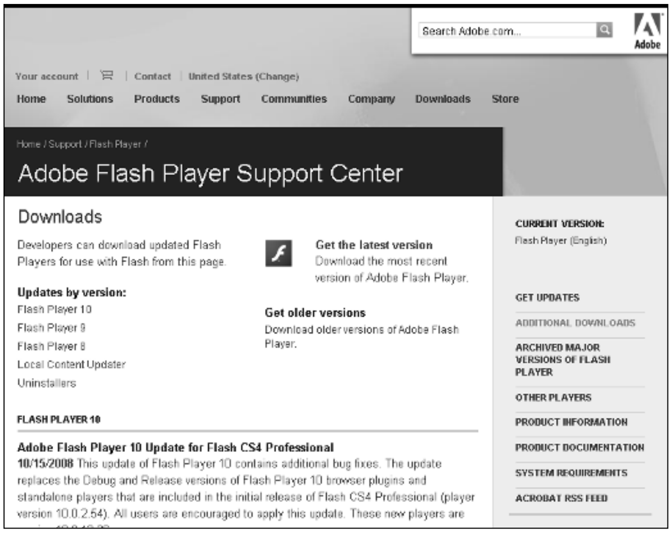
As shown in Figure 1.9, you should see links for all versions of Flash Player, including both debug and production versions, for a variety of operating systems and browsers.

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. ■**

FIGURE 1.9

The Adobe Flash Player Support Center


Search Adobe.com...
Adobe
Your account | Contact | United States (Change)
Home Solutions Products Support Communities Company Downloads Store
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 10
Flash Player 9
Flash Player 8
Local Content Updater
Uninstallers

Get older versions
Download older versions of Adobe Flash Player.

CURRENT VERSION:
Flash Player (English)

GET UPDATES
ADDITIONAL DOWNLOADS
ARCHIVED MAJOR VERSIONS OF FLASH PLAYER
OTHER PLAYERS
PRODUCT INFORMATION
PRODUCT DOCUMENTATION
SYSTEM REQUIREMENTS
ACROBAT RSS FEED

FLASH PLAYER 10

Adobe Flash Player 10 Update for Flash CS4 Professional
10/15/2008 This update of Flash Player 10 contains additional bug fixes. The update replaces the Debug and Release versions of Flash Player 10 browser plugins and standalone players that are included in the initial release of Flash CS4 Professional (player version 10.0.2.54). All users are encouraged to apply this update. These new players are

Flex 4 Development Tools

Flex developers have many development tools to choose from: Flash Builder 4, the Flex 4 SDK, and Flash Catalyst.

Understanding Flash Builder 4

Flash Builder 4, formerly known as Flex Builder, is an *integrated development environment* (IDE) for building Flex applications. This is the tool that most developers use to build Flex applications. I describe Flash Builder 4 in detail in Chapter 2.

Using the Flex 4 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 Flash 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 as Emacs or a version of Eclipse without the Flash Builder plug-in that doesn't have the specific support for Flex application development that you get with Flash Builder.

You can download the most recent version of the Flex 4 SDK from this Web page:

```
http://opensource.adobe.com/wiki/display/flexsdk/Download+Flex+4
```

The SDK is delivered in a zipped archive file that can be extracted to any platform.

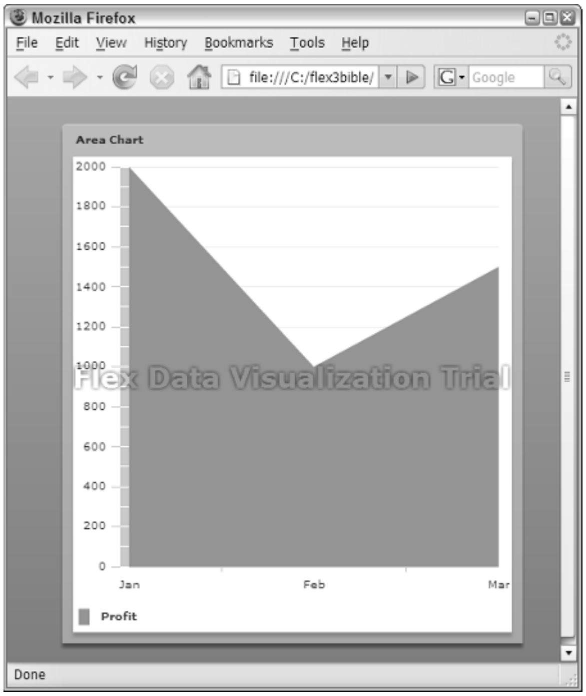
The SDK includes most of the class library you use to build Flex applications. The following components, however, require a license for deployment:

- Flex Data Visualization components, including charting and other advanced visual controls
- Application profiling tools

As shown in Figure 1.10, if you decide to use the Data Visualization components without a license, any instances of the components are displayed in your application with a watermark, indicating that you are using an evaluation version of the component.

FIGURE 1.10

A watermarked charting component



In addition to the Flex class library, the Flex 4 SDK includes these command-line tools:

- **adl.** The AIR debug application launcher.
- **adt.** The AIR developer tool.

- **acompc.** The AIR component compiler.
- **amxmlc.** The AIR application compiler.
- **asdoc.** A tool to extract documentation from ActionScript classes and generate HTML file sets known as *ASDocs*.
- **compc.** A compiler for building component libraries, Runtime Shared Libraries (RSLs), and theme files.
- **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.
- **fdb.** A debugger to debug applications.
- **mxmlc.** A compiler for building Flex applications.
- **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 Adobe's documentation.

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 4 SDK bin directory to your system's path. This enables you to run the compiler and other tools from any folder without having to include the entire path in each command.

Tip

When you install Flash Builder 4 on Microsoft Windows, the installer provides a menu choice that opens a command window and adds all directories containing Flex 4 components to the current path. To use this tool, choose All Programs ➪ Adobe ➪ Adobe Flex 4 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:

```
cd /flex4bible/myfiles
```

Assuming this directory contained a file called `HelloWorld.mxml`, to compile the application, you would run this command:

```
mxmlc HelloWorld.mxml
```

After 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 documentation. ■

Getting Help

Documentation for Flash Builder 4 and Flex 4 is available from the Adobe Web site at:

```
http://help.adobe.com/en_US/Flex/4.0/UsingFlashBuilder/index.html
```

The most current version of the ActionScript 3.0 Language Reference in for the Flex 4 SDK is available at:

```
http://help.adobe.com/en_US/Flex/4.0/langref/
```

The documentation also is delivered in the new Adobe Community Help application with Flash Builder 4. 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 are built in three programming languages: MXML, FXG, and ActionScript.
- Flex applications can be run as Web applications with Adobe Flash Player, delivered through a Web browser.
- Flex applications also can be run as cross-operating system native desktop applications, hosted by the Adobe AIR.
- The Flex 4 SDK is free and available as an open-source project that's managed by Adobe Systems.
- Flash Builder 4 is a commercial integrated development environment for building Flex applications.
- Flash Catalyst is a new application that enables graphic designers to create working prototypes of graphically rich Flex applications and define graphical skins for Flex components.
- 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.