Part I

ActionScript 3.0 Language Basics

IN THIS PART

CHAPTER 1

Introducing ActionScript 3.0

In this chapter you'll look at what ActionScript is, where you can use it, and what you can do with it. You'll learn where ActionScript fits in the grand scheme of the Flash Platform, and you'll take a complete tour of the tools and technologies involved therein.

What Is ActionScript 3.0?

You may well already know the answer to this question, because you had enough interest in it to buy this book! ActionScript 3.0 is the language used to program interactive Flash content. Where this content goes and how you can build it is the subject of the following section.

ActionScript 3.0 is a well-organized, mature language that shares much of its syntax and methodologies with other object oriented, strongly typed languages, so an experienced programmer can readily pick it up. Don't fear, though, for this book introduces ActionScript from the bottom up and starts gently.

If you've used Flash before but never ActionScript, you might know that you can build content for Flash Player without ActionScript — but without ActionScript, Flash is just an animation tool (though, admittedly, a good one). ActionScript is necessary when you want to create Flash content that is highly dynamic, responsive, reusable, and customizable. Here's just a short list of the many things you can accomplish using ActionScript:

- Loading images
- Playing audio and video
- Drawing programmatically
- Loading data such as XML files
- Responding to user events such as mouse clicks

To get the most out of the Flash Platform, you're gonna need ActionScript 3.0. And as you learn what it can do, you'll be amazed at its power. But let's look at how ActionScript fits into the Flash universe.

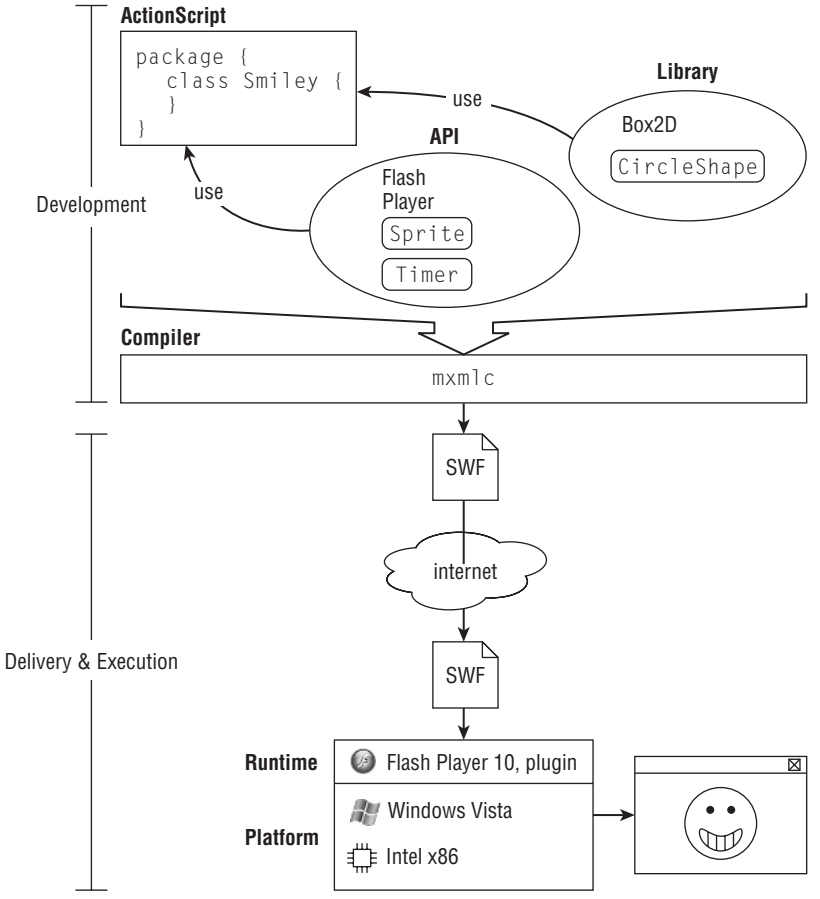
Exploring the Flash Platform

The wild world of Flash is by no means small, and wrapping your head around ActionScript and all the technologies related to it is challenging. Even if you don't know all the latest technologies in the peripheries, it's essential that you know major parts of the *Flash Platform* and how they work together. Defining that term and what it encompasses will be the goal of this section.

You'll examine the different parts of the Flash Platform from the perspectives of you building content and the user getting and running content. I'll define and discuss the tools, languages, platforms, and runtimes involved in the process. Figure 1-1 shows a bird's-eye view of the Flash Platform.

FIGURE 1-1

A high-level overview of the Flash Platform



You, the programmer, write in a computer programming *language*, ActionScript 3.0. You use *tools*, like Flash Builder, and specifically a *compiler*, to convert your code into an *executable*, in this case a *SWF*.

The SWF is delivered to the end user on her *platform* and executes inside a *runtime*, usually Flash Player, of which there are many versions. Let's break this down.

A Programmer's Perspective

For all the ways that the Flash Platform is unique, it shares the same basic steps as most programming environments. You write code in a language using some kind of editor, tool, or integrated development environment, and you use a compiler to convert that code into a file that can be run on the target environment.

Language

A computer programming *language* defines the grammar and lexicon that you'll be working in to create beautiful code. ActionScript 3.0, Python, Java, Lua, and C# are different languages. They all look different and have different rules for what you type where, what words are reserved, how to loop and how to write comments, and even where you can and can't put spaces. ActionScript 3.0 is not the same language as ActionScript 2.0, and neither is ActionScript 1.0.

ActionScript 3.0 has some features in common with modern JavaScript, because both are designed to adhere to specifications of a family of languages called ECMAScript. At the time of writing, this fact is little more than a curiosity, because most JavaScript in use is written to a baseline standard far behind the kind of JavaScript that starts to look like ActionScript 3.0. Furthermore, adherence to ECMAScript standards has provided little visible benefit, and progress marches on. In general, ActionScript 3.0 looks most like Java or C#. Coming to ActionScript 3.0 from either of these languages, or ActionScript 2.0, should be a fairly smooth ride. In this second edition, I've removed any emphasis put on transitioning from ActionScript 2.0 to ActionScript 3.0.

I'll describe some features of the language here. This might help some of you who have several languages under your belts already and can benefit from a description of ActionScript 3.0. If you don't understand any of these terms, please don't fret! The rest of this Part exists to investigate these qualities of the language in depth.

ActionScript 3.0 can use both dynamic and strong typing, but the compiler, the language, this book, and most of the world want you to use it with strong typing. If you want to live in a dynamic world, you can do so by turning off strict mode in your compiler or development environment. ActionScript 3.0 is an object oriented language that makes heavy use of namespaces. It has facilities for reflection. It embodies some elements of functional programming.

Maybe it goes without saying, but ActionScript 3.0 is the primary language used in the Flash universe. However, it's not the only one.

Depending on the platform and runtime you're targeting — that is to say, where, on what device, and on what software you want your content to run — you might use another language that this book is not about. I'll get into this more once you look at the platforms and runtimes that exist in the Flash universe. But rest assured that ActionScript 3.0 is the way to go for the Flash Platform right now. Most other options are for older technology.

There's another language that's a big part of the Flash Platform, and that's MXML. MXML is a declarative XML language used to program Flex. The interesting thing is that MXML compiles into ActionScript 3.0 during building. It's also used interchangeably with ActionScript 3.0. I won't discuss MXML or Flex in this book, but because Flex is a superset of Flash, this book provides an excellent, maybe prerequisite, background for any Flex developer.

Mixed into ActionScript 3.0 are several microlanguages, tailor-made for solving specific kinds of problems more efficiently than the grammar of ActionScript 3.0 would allow. Technically, these are part of

the ActionScript 3.0 language specification, but when you use them you can tell instantly that another dialect is being spoken. These are E4X, a language for manipulating XML; regular expressions, a language for searching for and manipulating patterns of text; and XML itself, a way to store hierarchical information. Furthermore, there are closely associated languages, like Pixel Bender language, used to write Pixel Bender shaders. Although this language can't be written directly into ActionScript 3.0 code like E4X, it is necessary to use some of Flash Player's features. All of these languages are important parts of Flash Platform development and cannot be ignored by an ActionScript 3.0 developer.

There are two major points of confusion when speaking about the ActionScript 3.0 language. First is the fact that ActionScript 3.0 can change, and has changed, without changes to that little "3.0" number sitting in its title. It's a living language, and it's being developed even as you read this book. (Hello readers from the future! Have we learned how to speak to dolphins yet or what?) So how can ActionScript 3.0 change? Well, there are two ends that have to agree: the compiler has to make something out of your code, and the runtime has to be able to run what the compiler makes. But the fact is, all it takes is for the compiler to change to support a change in the language. Both can be modified at the same time, or the compiler only could change, to allow for alterations in the language. The compiler is the most important because it's the only part of the entire Flash Platform that sees actual ActionScript 3.0 source. This kind of change has already happened. The compiler shipped in Flash CS4, Flash Builder, and in newer builds of the Flex SDK supports syntax for parameterized types that you'll see in Chapter 9, "Vectors." This syntax, `TypeA.<TypeB>`, looks like utter rubbish to an older ActionScript 3.0 compiler like that in Flash CS3, yet it is (now) part of ActionScript 3.0. We simply have to be careful. The second point of confusion when speaking about ActionScript 3.0 is the difference between ActionScript 3.0 and the Flash Player API.

API

Where a language determines keywords (like `for`, `class`, and `is`), syntax (like where to put curly braces), and grammar (like how subexpressions are evaluated and what can and can't go on the left side of an assignment), it's really the *Application Programming Interface*, or API, that gets most of the work done. It's easy to confuse these two, so let's untangle them once and for all.

The language by itself can't do much of anything. Without an API it's little more than a glorified calculator. You can do operations like creating variables, assigning values, summing up things, and concatenating strings; you can even create classes and functions. That said, there's a whole lot to learn about the language itself, and you could in theory make it through the end of Part II, "Core ActionScript 3.0 Data Types," before using any of the Flash Player API, if you keep your eyes closed strategically. It's the runtime (Flash Player or AIR) that provides most of the exciting stuff the Flash Platform has to offer: graphics, sound, animation, networking, video, and so on. None of this is built into the language.

You can draw an analogy between programming languages and spoken or written languages. It's necessary to understand the grammar and pronunciation of a language, but that alone is not sufficient for communication. You need a rich vocabulary, and that's what the API provides.

If you don't mind skipping ahead to some topics discussed in Chapter 4, "Object Oriented Programming," an easy way to determine what's part of the core language and what's part of the API is to look at how it's namespaced. Any classes and functions in the default package are part of the language, like `Error`, `XML`, `int`, and `Number`. Anything in the `flash.*` package and its subpackages is part of the Flash Player API, like `flash.display.Sprite` and `flash.geom.Matrix3D`. The Flash Player API is a library of classes and functions that get real stuff done.

Each runtime you target when building a program has its own API associated with it. In this book, I'll only cover the Flash Player runtime (specifically only versions 9 and up) and the Flash Player API.

There is one more runtime I could potentially target: the AIR runtime. The good thing about the AIR API is that it's a superset of the Flash Player API. In other words, if you want to build AIR apps, everything you learn in this book will be applicable, and much will be necessary.

The runtime and the API it supports are fundamentally linked. Adobe engineers add new features into new versions of the Flash Player (the runtime) and simultaneously expose those features to programmers through additions to the Flash Player API. Changes to the API are far more frequent than changes to the language. That's why, in this book, I note when topics I discussed are particular to a certain version of the API.

Libraries

The Flash Player API is a staggeringly large collection of classes and methods that help you make interesting things happen on the screen, as the weight of this book attests to. But the brilliant thing about code is its extensibility. For every ability that the Flash Player API enables, there are dozens of ways that people have taken advantage of it to build something more complicated or make some task easier. And many of these people have been so kind as to share their hard-written code with the world (although a few do expect compensation).

When you build applications for the Flash platform, in addition to using the Flash Player API — a given — you can use any number of other libraries, leveraging their capabilities to pull off some impressive feats without breaking a sweat yourself. My favorite libraries include tweening libraries that let you animate things about the screen with a single line of code; physics engines that let you simulate collisions, friction, gravity, and other forces; 3D engines that let you present 3D objects and scenes; loading libraries that let you streamline the slightly annoying process of getting several types of assets loaded into a larger application; and data structure libraries that provide optimized storage for specific purposes.

A library is simply a collection of code that you can use. Sometimes you have the actual ActionScript 3.0 code that makes it up, and sometimes you use a precompiled binary (a SWC). In either case, you use libraries like you use the API. You create and access classes that they contain to get the job done. Libraries extend your programming toolbox with new, usually job-specific tools. In fact, the API is itself a library; what makes it special is that it's built into the runtime.

Because there are as many libraries as there are stars in the sky, I only mention them when there's a specific task that developers overwhelmingly use a library to accomplish, and in this case I'll just give a quick description of the library and let you know where you can find more information.

Compilers, Tools, and IDEs

There's one critical piece in this puzzle I haven't covered yet. ActionScript 3.0 code is text. You write it down in plaintext files. But to get from plaintext files to something you can actually run — in this case a SWF — you have to *compile* it. To put it simply, compiling translates the text of a program — the source code — into a simpler language that the runtime can run directly. We speak and understand human languages like English and French. Your computer speaks a certain instruction set that depends on what processor is inside it. It would be quite painful to program directly in the computer's language. (Some people come close by programming in assembly languages.) And it would be quite difficult and imprecise to have the computer interpret meanings from a regular, spoken language. So we have computer languages to bridge this gap — and compilers to translate. Unlike translating spoken languages, there can be no ambiguity in the compiler's translation. If the compiler doesn't understand the code you've written or believes it's incorrect, it fails to compile. You'll learn to recognize and deal with compiler errors in Chapter 5, "Validating Your Program."

So how do you use these compilers, and where do you obtain them? This all depends on the development environment you set up for yourself. In the Flash universe, most of the available versions of the ActionScript 3.0 compiler are integrated tightly with some tool.

The tools and Integrated Development Environments, or IDEs, you're likely to come across are

- Adobe Flash Builder
- Adobe Flash Professional
- FlashDevelop
- FDT

In addition, there are free and open-source versions of the Flex SDK available for download from Adobe. These toolchains are not IDEs, but they contain a compiler. All the tools in the previous list are either integrated with their own compiler or integrate tightly with a copy of the Flex SDK that you provide or that comes bundled with it. Because of the tight integration between these tools and the compiler, you may not even be aware of the compiler. For example, when you choose Publish or Test Movie in Flash, you invoke Flash's ActionScript compiler. With the default settings in Flash Builder, the compiler is invoked every time you save a file!

For every tool in this list but Flash, the compilers used are part of Adobe's Flex SDK, which comes in both free and open-source flavors. You can get the latest and greatest versions of these at no cost at `http://opensource.adobe.com/`. Specifically, these come with five major compilers: `mxmlc` to compile MXML and ActionScript 3.0; `asc` to compile ActionScript 3.0; `fsch`, a shell for repeated compilation; `compc` to compile SWCs instead of SWFs; and `adl` to compile and package AIR applications.

Source code is just text, so you can write it with any text editor you like. But because it's structured code, you can use tools to help you write this code faster and with fewer errors. IDEs, like Flash Builder, give you intelligent tools for writing ActionScript 3.0 code; searching it; discovering relationships between parts of the code; auto-completing your typing; exploring files; searching through projects; renaming classes, methods, and variables; and more. Possibly the most powerful feature of a good IDE, however, is its integration of an interactive debugger, which you can see how to use in Chapter 25, "Using the AVM2 Debugger and Profiler." Using an IDE makes a programmer's life much better, but you can always fall back on using a text editor to write ActionScript 3.0 source code and running `mxmlc` yourself. It's entirely up to you to decide what tools to use in your development environment. Personally, I recommend that you use Flash Builder if you're going to follow along with the book. I recommend against using Flash Professional for any serious ActionScript 3.0 programming.

Once you've compiled your ActionScript 3.0 program, you'll end up with a SWF file.

SWFs

A SWF file, or simply "a SWF," is an efficient, compressed binary file that can contain graphics, animation, text, bitmaps, sounds, video, and even arbitrary data. Most importantly for us, it also contains compiled ActionScript. The main purpose of a SWF is to get the stuff into the world (possibly across the great expanses of the internet, and onto the screens of your users) that we, as programmers, create. The end consumer of a SWF file is the Flash Player runtime.

I mentioned one other runtime: the AIR runtime. When compiling AIR apps, an `.air` file is generated, but even this is a package that contains SWFs for its executable ActionScript code.

Flex

I've tiptoed around Flex up until this point. Many newcomers to the Flash Platform are confused by Flex versus Flash, often with respect to the naming of certain products. Let's clear the air here.

Flex is two things. Primarily, it's a big, well-designed library for developing Rich Internet Applications, or RIAs. RIA is something of a vague buzzword encompassing programs that live on the internet and have some of the features once reserved for desktop applications: widgets like scrollbars to scroll, buttons and tabs to click, flippers to flip, and so on; multiple screens and transitions; display of tables and data. The Flex framework is a library that contains all these widgets, the ability to skin them, lots of code for easily connecting to web services, and more. As a library, it adds on to the capabilities of the Flash Player API, not replaces them. Furthermore, the end product of a program that uses Flex is a SWF, just like any other SWF, and it runs in Flash Player just like any other SWF. The only difference is that when you use the Flex framework, that SWF either contains or references the Flex framework library.

The second major component of Flex is a declarative XML language called MXML. This is a separate language from ActionScript 3.0, although when you compile a project with a Flex compiler, it is converted into ActionScript 3.0 code — and it coexists happily with ActionScript 3.0 code.

I won't cover either the Flex API or MXML in this book. However, it's important to know that Flex is still based in ActionScript and can be built targeting the same Flash Player runtimes. Because the Flex framework is a library built on top of ActionScript 3.0 and the Flash Player API, what you will learn in this book is indispensable for Flex development.

Note

A word about the use of the word "Flex" in product names: Flash Builder 4 is the successor to Flex Builder 3. Both of these tools can be equally well used to build either Flash or Flex applications, which the change of name is meant to emphasize. You'll also see the term "Flex" in the Flex SDK. The truth is that the Flex SDK bundles all you'll need to compile and package Flash, Flex, and AIR applications. ■

In Short

Let's quickly put back together the programmer's experience with the Flash Platform, in a typical example. You open your IDE, Flash Builder. You start writing code in the ActionScript 3.0 language and use classes from the Flash Player API. You choose Run from the menu, and Flash Builder builds and runs your program, compiling all your ActionScript 3.0 code into a SWF and opening that SWF in Flash Player.

A User's Perspective

One of the biggest benefits of programming for the Flash Platform is that your content can be easily run in so many places for so many users. The two *runtimes*, Flash Player and AIR, are widely supported on multiple *platforms*.

Runtimes

A runtime is an environment in which a program executes. The runtime provides all the services necessary to do the things that the API promised were available. In the Flash Player API, you can create an instance of `Camera` to gain access to a connected webcam; the Flash Player runtime has to deal with the potentially complex task of finding the connected hardware and pulling a video stream from

it. In ActionScript 3.0 and the Flash Player API, you can programmatically draw graphics; it's up to the Flash Player runtime to render those graphics and work with the operating system to display them on-screen.

I've mentioned two runtimes: AIR and Flash Player. AIR, because it's a runtime made for desktop applications, contains additional capabilities, such as rendering web pages and spawning new windows in the user's operating system. Another runtime in the Flash universe, Flash Lite, is a somewhat dated environment for mobile and embedded devices. It doesn't support ActionScript 3.0, so I won't waste time on it. I'll only cover the Flash Player runtime here.

Platforms and Platform Independence

A platform is considered a combination of the hardware and operating system in use. More specifically, it is the instruction set of the CPU that matters to the platform, although these are mostly standardized. Users who want to run Flash content (stuff made with ActionScript 3.0 and the Flash Player API) must be able to run Flash Player on their platform.

At the time of writing, Flash Player 10 is supported on PCs with x86 processors running Windows 98 and up; Macs with PowerPC (G3, G4) or Intel processors running OS X 10.4 and up; PCs with x86 processors running several flavors of Linux including Red Hat, SUSE, and Ubuntu; and systems with x86 or SPARC processors running Solaris 10. Notably, at the time of writing, 64-bit processors are not supported, but commercially used 64-bit processors have no problem running in 32-bit mode, so I can still use Flash Player on my desktop's screaming Core i7 processor (which may be stone age by the time you read this). Also, at the time of writing, Flash Player 10.1 is planned to roll out on multiple mobile devices, including the latest Android OS and Palm webOS. In addition, there are other exotic Flash Player runtimes for platforms. For example, a PlayStation 3 with updated firmware has a Flash Player runtime on par with Flash Player 9.

In any case, that whole paragraph just means this: almost everyone can get access to the Flash Player runtime. One of the great things about the Flash Platform is that it's *platform independent*. It doesn't matter what kind of computer you use to compile your program: the resulting SWF file is completely indifferent to where it was born and where it's going. It's simply a binary file, and anyone who knows how to read and interpret it may do so, just like, say, a JPG image file. The other component of platform independence is that it doesn't matter what platform your user is on; users on different platforms are able to run the same SWF and see the same outcome.

This model of platform independence is just the same as Java's. Once Java code is compiled, it can be sent anywhere and run on any platform — that is, any platform where a Java Runtime Environment is available. Contrast this with traditional software development or game development, where a product is made for one platform and must be significantly reprogrammed or, at minimum, recompiled to be available on another platform. And considering that Flash content is widely distributed on the internet, the benefit of platform independence is clear. As the programmer, you don't want to have to make one SWF for every platform, and the users don't want to have to choose their platform every time they view a web page with Flash content.

The Flash Player Zoo

You've already seen that the Flash Player is available on different platforms. It also comes in several flavors and a plethora of versions.

There is a *standalone* Flash Player used to run SWF files from a user's computer. This can be bundled with a SWF to generate an executable that launches Flash Player and the bundled content at the same time. This is a Flash Player *projector*. Most common, there are the *plug-in* Flash Players that operate inside your browser to host Flash content in a web page. (On PCs these come in both ActiveX plug-in and Netscape plug-in varieties.)

Furthermore, all these flavors of Flash Player are available in debugger versions. You can learn more about the debugger versions in Chapter 24, "Errors and Exceptions."

Finally, there are lots of versions of Flash Player in existence. The major version of Flash Player is most important (for example, Flash Player 9 or Flash Player 10). The minor versions and revision numbers appear after the major version number, such as in Flash Player 10.0.22.87. In general, major versions introduce suites of new features, and minor revisions are mostly bug fixes and performance enhancements, although some feature changes creep in.

When you compile ActionScript 3.0 code, you can target a specific major version of Flash Player, because as I mentioned earlier, every version of Flash Player is tied to the Flash Player API for that version. All ActionScript 3.0 code is compiled to target Flash Player 9 at a minimum.

The good news is that you don't need to worry about the differences between all the flavors of Flash Player. The same SWF will work in Flash Player of the same version of every platform, no matter if it's the standalone version or the plug-in version, and no matter what browser the plug-in version is being hosted by.

In Short

Here you see a typical user's interaction with the Flash Platform. The user downloads or upgrades to Flash Player 10. She opens Firefox on her Mac and navigates to a page with Flash content. Behind the scenes, her browser downloads the SWF file from the internet and runs the plug-in version of Flash Player 10, and the Flash Player runs the content of the SWF. End result: she sees the Flash content hosted in the web page.

From ActionScript 2.0 to ActionScript 3.0

If you've programmed in ActionScript 2.0 before, ActionScript 3.0 has a whole lot of new features for you to explore. Here is an overview of the key new features. You may skip this section if you are coming to ActionScript 3.0 from a different language.

Display List

In ActionScript 2.0, there were three basic types of objects that could be displayed: movie clips, buttons, and text fields. These types didn't inherit from a common source, meaning polymorphism didn't work for these display types. Furthermore, instances of these display types always had a fixed, parent-child relationship with other instances. For example, to create a movie clip, you had to create that movie clip as a child of an existing movie clip. It was not possible to move a movie clip from one parent to another.

In ActionScript 3.0 there are many new display types. In addition to the familiar types such as movie clips, buttons, and text fields, you'll now find new types such as shapes, sprites, loaders, bitmaps, and more. All display types in ActionScript 3.0 inherit from `flash.display.DisplayObject`, allowing you to use them interchangeably in many cases. Furthermore, display objects in ActionScript 3.0 can be constructed independent of any other display object, and these objects can be associated as children of other display objects and even moved from one parent container to another. In other words, you can create a text field in ActionScript 3.0 simply by calling the constructor, and that text field will exist independent of any parent container object.

```
var text:TextField = new TextField();
```

You can then add the text field to a parent container at any time. The following example illustrates this with a display object called `container`, which could be a sprite or any other display object container type:

```
container.addChild(text);
```

Note

In the preceding example, `container` is used as a generic variable name that would presumably refer to an object created elsewhere in the code. ■

The hierarchy of parent containers and their children is known as the *display list* in ActionScript 3.0.

Runtime Errors

ActionScript 3.0 provides many new runtime errors. This is an important new feature because it allows you to diagnose problems much more quickly. In ActionScript 2.0, when an error occurred at runtime, it would frequently occur silently, and it would be difficult for you as the developer to determine what the problem was. With improved runtime errors and error reporting in the debug player, it is now much easier to debug ActionScript 3.0 applications than it was with ActionScript 2.0.

Runtime Data Types

Strict typing in ActionScript 2.0 was only used by the compiler, not at runtime. At runtime, all ActionScript 2.0 types are dynamic. However, in ActionScript 3.0, strict typing is preserved at runtime as well. The advantage is that now runtime data mismatches are reported as errors, and application performance and memory management are improved as a result of preserved typing at runtime.

Method Closures

In ActionScript 3.0 all methods have proper method closures, which means that a reference to a method always includes the object from which the method was originally referenced. This is important for event handling, and it stands in stark contrast to method closures in ActionScript 2.0. In ActionScript 2.0, when you reference a method, the object from which the method is referenced does not persist. This causes problems, most notably when adding event listeners. In ActionScript 2.0, a delegate is often used as a solution. However, in ActionScript 3.0, delegates are not necessary.

Intrinsic Event Model

In ActionScript 3.0, the event model is built into the core language. Many native ActionScript classes, including all the display object types, inherit from the `flash.events.EventDispatcher` class. This means that there is one standard way to dispatch and handle events in ActionScript 3.0.

Regular Expressions

Regular expressions are a powerful way to find substrings that match patterns. ActionScript 3.0 includes an intrinsic `RegExp` class, which allows you to run regular expressions natively in Flash Player.

E4X

E4X is short for ECMAScript for XML, and it is a new way to work with XML data in ActionScript. Although you can still work with XML as you did in ActionScript 2.0 by traversing the DOM, E4X allows you to work with XML in a much more speedy and intuitive manner.

Summary

- AS3 is used to program interactive content for the Flash Platform.
- AS3 is object oriented and usually strongly typed, and it supports dynamic types.
- The Flash Platform encompasses languages, APIs, tools and IDEs, compilers, and the Flash Player and AIR runtimes.
- AS3 is the principal language used for Flash; MXML turns into AS3.
- Without the Flash Player API, you can't do much with AS3.
- The API is tightly tied to the runtime. This book is about ActionScript 3.0 and the Flash Player API.
- There are many tools for building Flash content. Most rely on the ActionScript compilers in the Flex SDK.
- You compile your ActionScript code to produce a SWF.
- SWFs are portable, compressed, and platform independent. They contain compiled code and other resources.
- SWFs are consumed by Flash Player.
- Flash Player comes on many platforms in many different flavors.
- All a user needs to run your Flash content is a compatible version of Flash Player.