

Part I: Introduction to Flex 3

Chapter 1: Why Flex?

Chapter 2: The Flex 3 Ecosystem

Chapter 3: ActionScript 3.0 Fundamentals

Chapter 4: Using ActionScript 3.0

Chapter 5: Introduction to Flex Builder 3

1

Why Flex?

It seems like nearly every programming book I have ever read starts off with the obligatory “Introduction to [whatever]” chapter, which basically regurgitates the essentials that everyone knows, and hence is the one chapter nearly every reader skips. In this first chapter of the book, I thought we’d start things off with a different approach, one that takes you, the reader, through the reasons that you might want to use Flex, what Flex is for, bursting some myths about Flex and the Flash platform, and, finally, ending up with a “Top 10” of Flex’s strong points. Enjoy!

What Is Flex?

Often this is either the first presciently asked question for someone new to the Adobe Flex ecosystem and the Flash Platform, or it is the last question asked after many hours of wrestling with how all of this “stuff” fits together in the scheme of things. Read on: the first two chapters in this book will make it all crystal clear.

In a nutshell, Flex is a rich Internet application (RIA) development toolkit based on the ActionScript 3.0 and MXML languages that can be deployed for the Web using the Flash Player plug-in, or to the desktop using the Adobe Integrated Runtime (AIR).

As Adobe Evangelist James Ward puts it in his article “How I Overcame My Fear of Flash” (www.jamesward.com/wordpress/2007/02/21/how-i-overcame-my-fear-of-flash/),

“[the] Flash Player is a ubiquitous, cross-browser, cross-OS virtual machine enabling next generation web experiences. Flex is a simple tool for developers to build applications that execute in the Flash virtual machine.”

What Is an RIA?

No introductory discussion of Flex would be complete without at least a cursory examination of the rich Internet application, or RIA.

Although the concept of RIAs had been around for quite some time, it wasn't until 2002 when Macromedia (now Adobe Systems Inc.) coined the term from a Flash MX white paper, that the idea began catching on in web development. You can read the white paper, "Macromedia Flash MX — A next-generation rich client," at www.adobe.com/devnet/flash/whitepapers/richclient.pdf.

The RIA's predecessor could be called a "Web 1.0 application," which is based on a linear, hypertext-centric architecture written in HTML, JavaScript, and CSS, which for the purposes of this text we will call a "hypertext application." An RIA, which has also come to be known as a "Web 2.0 application," is unique and differs from a hypertext application in several important ways:

- ❑ **Rich media** — Although it could be argued that a hypertext application can have and use what is known as "rich media," it is not its defining characteristic. An RIA, on the other hand, is typified by the presence-rich content. This usually involves custom designed interfaces in a highly branded and/or expressive visual environment, which may use sounds, video, graphics, and motion that cannot be reproduced in a hypertext environment. One could say that a hypertext application is defined by a textual paradigm, whereas an RIA is defined by a visual paradigm.
- ❑ **Resides on the client** — More importantly, a hypertext application follows a page-oriented architecture, where an application interface is created on the server and downloaded to the client in "pages." An RIA, on the other hand, is a self-contained solid-state machine that runs on the client, usually with the help of a dedicated runtime, such as a browser plug-in.
- ❑ **Asynchronous communication and data persistence** — The browser page makes an HTTP `GET` request to the server each and every time the application must refresh its functional state, which uses a *synchronous* method of communication. The state and logic of the application must be tracked on the server, as each page cannot easily manage application logic across other pages. So, it is left up to the server scripting, typically coded in a language such as PHP, ASP, or JSP, to manage the application state for each and every client connected to the server.

Whereas a hypertext application communicates or "downloads" each page to the browser as the user traverses the application, the RIA need only be downloaded once, since it manages the creation of its own UI and internal logic. Because the client maintains full control over application state and functionality, and can communicate with the server independently of user interaction with the application, it is said to use *asynchronous* server communication.

- ❑ **Lower network consumption** — And because an RIA does not need to communicate its state to the server, it is free to communicate to the server only if and when the online functionality is required, such as when retrieving or uploading a file, saving or retrieving data to and from the user's account, or performing some task the client is unable to process on its own. This has the potential of considerably reducing bandwidth and server load, often translating into a huge financial savings.
- ❑ Developers must take care not to overuse the pre-fetching of assets and/or data, which can eliminate any bandwidth or server load savings that might have been gained. Some RIAs actually consume a lot more network resources, but these are typically applications that make heavy use of streaming media, which would simply not be possible in a hypertext application.

- ❑ **Acts like software** — Most RIAs distinguish themselves from simple embedded rich content by the presence of a standard user interface and a certain level of complexity that categorizes it as an application and not simply a rich media animation or video delivery medium. For instance, a video player on YouTube is not an RIA, but an online video editing application could be considered one.

Since most RIAs are developed according to object-oriented principles and not a page state model, their architecture can be more flexible; states can be set asynchronously by different parts of the application as a whole. This allows the application to act more like traditional software than a website. In other words, what determines where the user is in the application can be fluid and prone to many different variables, depending on the task being performed at any given moment. Therefore, RIAs have the advantage of increased responsiveness, as there is no waiting for a page refresh, which increases user productivity in accomplishing the task at hand.

- ❑ **Increased security protocols** — However, that flexibility and fluidity comes at a price. Maintaining the application on the client means increased vigilance in communications security must be taken to avoid abuse, and some RIA technologies have developed some very specific and sophisticated security “sandboxes” that allow data to be communicated to and from the application only in very specific ways, which is mostly a good thing.
- ❑ **Browser-opaque** — One of the typical drawbacks to RIAs over traditional hypertext applications is that because they act as a closed system (i.e., their application states are managed internally), they are “browser-opaque.” That is, they cannot be navigated with browser controls (i.e., history states, the Back button), are not search engine indexable, and are not visible to accessible devices. This is a failing of all RIAs by any technology, including AJAX, Flash, AIR, Silverlight, WPF, and JavaFX.

This means that just as hypertext applications are not very good at rich interactive content, RIAs are not very good at large volumes of search engine-accessible hypertext content, which is why many applications such as text-heavy websites, blogs, and social networks still use the traditional hypertext application paradigm. Some RIA technologies have made great inroads to mitigate this disadvantage, but at the moment these kinds of solutions require conscious implementation on the part of the developer to bridge the browser divide. No RIA technology is currently browser-transparent, and probably will not be until browsers start integrating an RIA runtime as part of their native rendering model and search engines change their indexing schemes to include certain state-dependant standards, which given the current rate of browser evolution may be a long time in coming. The irony is that even as RIAs are redefining the way we use the Web, they are accused of “breaking the Web paradigm.” Perhaps it is a matter of finding the right balance.

See Chapter 40 for more information on deeplinking in Flex applications.

- ❑ **Can be desktop-enabled** — Until recently, the term “RIA” implied that the application would be run from within a browser context, which typically limits access and interaction with the operating system. But now, certain technologies, such as Adobe AIR, allow for the creation of RIAs that break out of the browser paradigm and onto the desktop. An RIA that resides on the desktop is differentiated from a traditional software application in that it still requires a runtime sandbox to execute the application, is usually built on web technologies and their derivatives, and has direct access to the Internet. A desktop-enabled RIA may function similarly to both an Internet browser and an installed application. At the moment, no RIA technology has the same level of system interaction with the operating system that typifies installed software, but this line is becoming ever more indistinct as the technologies driving RIAs continue to evolve.

- ❑ **“Sometimes connected” applications** — Since the RIA can often choose when it is connected to the server, certain applications — whether in a browser or on the desktop — can operate in a “sometimes connected” mode unique to web applications, although the term is mostly used when referring to desktop RIAs. This can allow users to work offline for a period, and reconnect to the server if and when they need to save their work or update the application. A “sometimes connected” application is not necessarily independent of the server, as would be traditional network-enabled desktop software, but it allows a degree of flexibility in application usage that was impossible to accomplish just a few years ago.
- ❑ **Enables new business models** — Since RIAs are redefining the way we use the Web, enabling applications that act more like software than web pages, new business models are possible, such as the software as a service (SaaS) model. This more than anything is the current thrust driving the Web 2.0 revolution, which is seen as the promises of the dot-com years finally coming home to roost.

For instance, Adobe has launched Acrobat.com, a rich service portal built into Flex that allows users to convert documents to PDF, upload documents and files to their account through Adobe Share, create real-time online meetings through the ConnectNow service (an online version of Adobe Connect), and author, save and download documents using Buzzword, a full-featured word processor application. Adobe has also released an online version of Photoshop that allows users to process images with a subset of the features of the original application, as well as a video delivery portal called Adobe TV, which features video tutorials on myriad Adobe technologies. Gmail and Google Docs, built with an AJAX development model, are also immensely powerful and popular applications. Other sites, too numerous to mention here, leverage the RIA advantage to offer users everything from personal scheduling and organization, to social networking and media creation tools. Using RIA technologies such as Adobe Flash and AIR in tandem, *some applications can even leverage simultaneous browser and desktop-enabled versions of their online service*, or create a product developed exclusively for desktop deployment.

Clarifying the Competition

Before we get into the specifics of Flex and the Flash platform, let us enumerate some of the current RIA technologies available. As of this writing, primary contenders in the current market are considered to be AJAX, Adobe Flash and Adobe AIR. Secondary contenders are considered to be Microsoft Silverlight, Sun JavaFX, OpenLaszlo, Mozilla Prism, Google Gears, Curl, and Adobe Flash Lite.

Since there is some confusion in tech reporting circles and the blogosphere as to competing equivalencies between certain of these technologies, without getting into too much detail on specific capabilities, some clarifications are in order.

- ❑ AJAX, Adobe Flash, Microsoft Silverlight, OpenLaszlo, and Curl run in a browser environment, so they can be considered to be comparable competitors.
- ❑ Adobe AIR, Google Gears, and Mozilla Prism are desktop-enabled RIA runtime environments, so they can be considered to be comparable competitors. Although Microsoft WPF is often compared with Adobe AIR, some do not consider it an RIA technology because it is not intended to be connected to an Internet server, but to be used as a high-level Windows development platform.

To clarify, Silverlight does not compete with AIR, and WPF does not compete with Flash, because they are deployed in completely different environments.

- ❑ Flash Lite and Silverlight applications can be deployed in a mobile environment, so they can be said to be comparable competitors. Many mobile RIAs are developed for the native device operating system instead of through a mobile browser, so there is some ambiguity as to whether certain mobile technologies can be considered to be RIA-centric. For instance, Java is currently the language used by the majority of mobile applications, and by itself is not considered an RIA-specific technology. Google Android is a mobile operating system, which is sometimes confused as an RIA platform.
- ❑ JavaFX competes with all of these, since it can be deployed to the browser, on the desktop, or in a mobile environment.

For the Love of Flex

Without getting into a blow-by-blow comparison between the previously mentioned technologies, let's examine some of the misconceptions about Flex and the Flash platform, and some compelling reasons why Flex, Flash, and AIR are extremely powerful solutions for building engaging, next-generation RIAs. Since you're reading this book, like me you're probably already convinced. But if you still have concerns, or you're still on the fence, let us be of help.

Bursting Myths about Flash and Flex

The majority of misconceptions and concerns about Flash and Flex fall into the following categories:

- ❑ **Flash is proprietary** — Many people immersed in traditional open-source development environments using Linux, PHP, Java have the following concerns:
 - ❑ *Can I author a SWF application without Adobe software?*

Yes. To write software you need three things: an editor, a compiler, and a runtime. The SWF format is open, and there are third-party utilities that enable you to compile ActionScript into SWF bytecode. The ActionScript API itself is built into the Flash player, and the Flash Player and AIR runtimes are not open source. The Flex compiler is an open-source, free command-line utility that can be used with any third-party editor. The Flex framework and many of the Adobe specifications and tools are open source. Some of Adobe's other tools (such as BlazeDS, a communications server for the AMF binary protocol), are open source, and for those that are not there are usually open-source versions (such as Red5, an open-source version of Flash Media Server). Some of the elements in the Flex SDK are not open source, such as the Flash Player, the font encoding libraries, and the Data Visualization components. More information on what aspects of the Flex SDK are open and which are free are available here:

<http://opensource.adobe.com/wiki/display/flexsdk/Downloads>. There are also some great proprietary tools for Flex development, such as Adobe Flex Builder, but if you need to go completely open source with your tools as well as your framework, there is the option.

- ❑ *Can I deconstruct a SWF application without Adobe software?*

Yes. There are third-party utilities that enable the deconstruction of SWF files. The resulting ActionScript is not in its originally written format, but reverse-interpreted from the SWF bytecode, so markup code such as compiler metadata, MXML, and CSS will be shown as their

ActionScript equivalents. For those people for which this is a negative, who wish to protect their distributed software, there are obfuscation and encryption schemes that make reverse-engineering SWFs difficult-to-impossible.

- *Do I have to use an Adobe runtime to run my software?*

Yes. The Flash Player itself is not open source, and even though the SWF format is open, Adobe controls the SWF specification. This means that although third parties can build software that can create SWF files, no one else but Adobe can create a new Flash Player. This is actually one of its strengths. The Flash Player's ubiquity at over 97.7% market distribution across popularly used browsers is because there has only ever been one Flash Player (albeit different versions). The Flash Player is able to achieve such a small download footprint, and the incredible number-crunching speeds of 100–1000% the execution speed of JavaScript primarily due to the presence of the JIT compiler, which converts ActionScript bytecode directly into system-level machine code at runtime. This level of precision engineering would be very difficult if not impossible without the dedicated Adobe Flash Player engineers who have done nothing but advance and perfect the Flash virtual machine for more than 12 years. If users had to choose between competing Flash Players for their browser space, the ubiquity of the Flash Platform would be diluted and meaningless. So, open sourcing the Flash Player would actually be incredibly damaging to the stability of the platform. However, the ActionScript virtual machine within the Flash Player, which is called Tamarin, has been open sourced to Mozilla, which opens the door for future runtime solutions.

- **It's used to create annoying content** — One could say that about nearly any Web technology. As Aral Balkan, Flash Platform developer extraordinaire, puts it in his article "Bare-naked Flash: Dispelling Myths and Building Bridges" (<http://aralbalkan.com/1305>), "Usability is not an inherent property of a platform ... is HTML '99% bad' because of MySpace?" (Uh, no, in case you were wondering, although it is a tempting thought.) And we've come a long way since the "skip intro" days. Through YouTube and other video portals, Flash revolutionized video on the Internet in 2003, and one has only to examine the plethora of RIAs present and on the market because of Adobe Flash, such as Buzzword, Adobe TV, Blist, Aviary, or SlideRocket, to name but a few, to know that Flash, used to its potential, is far from just a media plug-in for quirky content.
- **It's not browser-integrated or SEO-capable** — This is true of all RIAs, due to the nature of the architecture, and is not unique to Flash applications. See "What Is an RIA?" above for details.

However, Google can index SWF files and introspect their metadata, which can be set at compilation, even if state introspection must be manually included. Adobe has made some great advances in search engine indexing in cooperation with Google and Yahoo! and is in ongoing discussions with these two search engine companies to improve SEO capability and the overall search experience.

Flash natively allows for ActionScript-to-JavaScript communication, and the Flex framework brings application deep linking and full browser integration to Flash applications. This allows the browser URL to change as the user navigates through the application, which enables browser history states, bookmarking and the back button. See Chapter 40, "Deep Linking," for details. And these same tools allow Flash applications to be fully accessible to screen readers.

AIR, Adobe's desktop RIA solution, allows seamless script bridging between JavaScript and ActionScript, with the ability to treat Flash objects as native JavaScript objects, and vice versa. This means that AIR can develop solutions that leverage both DHTML and Flash seamlessly in the same application, which is simply not possible in a browser environment.

So, yes, Flash is mostly browser-opaque natively, but the tools exist to develop solutions that enable full browser- and accessibility-enabled Flash solutions.

- ❑ **It's a browser plug-in** — This is actually a good thing, as it allows the technology to advance independently of the snail's pace of browser languages. Consider that the Flash Player adoption has moved through versions 3 to 10, Flex through versions 1 to 3, and ActionScript through versions 1.0, 2.0 and now 3.0. Although some browsers have adopted more up-to-date standards, as far as overall browser adoption is concerned, JavaScript and CSS have barely moved one version ahead after nearly a decade, and updates to HTML 4 have been incredibly slow in coming.
- ❑ **It's not secure** — *This is perhaps one of the greatest misconceptions of Flash-based applications, and is patently and unequivocally not true.* Adobe continues to improve security sandboxing specifications multiple times over the course of any Flash Player version and is quick to issue an upgrade when a critical vulnerability has been found, to keep pace with new use cases, security scenarios and capability. In fact, a common complaint nowadays in the community is that the Flash Player security sandbox requirements are often too strict! In fact they are a judicious balance between robust security and strong capability. The security protocols prevent everything from cross-domain or illegal security domain communication, script injection and other common issues. The Flash Player is in fact the most secure, capable, and powerful RIA runtime on the market today. See Chapter 65, "The Security Model," for details.

For example, the security vulnerability known as "clickjacking" was identified in late September 2008 as a security flaw in all browsers, and within weeks a security advisory was issued for the Flash Player, and a security patch for Flash Player 10 issued a few weeks after in mid-October. Although Microsoft has issued a patch for IE 8 RC1 beta, to this date IE 7 continues to be vulnerable, and Firefox, Safari, and Chrome continue to be affected. This is perhaps an extreme example, but in general, security updates to the Flash Player occur as fast if not faster than all the leading browsers.

- ❑ **The Flex Framework bloats my application filesize** — This is also a valid concern. Flex is an incredible toolset for creating component-based RIAs (the rest of this book is testament to this fact). But that toolset comes at a price. Flex adds between approximately 120 to 600 KB of filesize to your SWF file upon compilation, which means that the user will need to download at least that much on top of your own development footprint to view your application.

One way to mitigate this added footprint is to use persistent framework caching, covered in Chapter 66, "Modular Applications Development." This allows the application to be compiled with only your code additions. Upon running your application in the browser, the Flash Player downloads a signed Runtime Shared Library (RSL) file containing the Flex Framework from an Adobe server, which is cached on the client. So your user only ever has to download it once, reducing the total download in subsequent sessions.

RSLs are compiled code libraries that can be used in many scenarios. They are typically common groups of logic that can be reused in multiple projects, and they can be cached on the client side to help make your applications smaller in compiled file size.

If you're looking to develop something very lightweight, as is the case for mobile or interactive ad development, then using the Flex framework may not be the solution for you. You can still author a bare-bones ActionScript 3.0 project in Flex Builder, using either the Flash IDE or Flex as the compiler. See Chapter 26, "Flash Integration," for use case scenarios.

- ❑ **Doesn't easily integrate with my CMS** — The fact that a Flash or Flex application may or may not integrate well with a preexisting CMS has more to do with the lack of a consumable web service or an ActionScript API on the part of the CMS than it has to do with an incompatibility of the technology. A Flex RIA can be made to do nearly anything with regard to organizing and consuming web content. The Flash Player has the ability to launch a system dialog allowing the user to browse the local filesystem for the purposes of uploading a file to the server, or download

Part I: Introduction to Flex 3

a manipulated vector or bitmap as an image file. All that remains is to build the client side of the CMS interface. There exist third-party solutions that facilitate this process, offering a convenient applications framework for building Flash-based CMS solutions.

- ❑ **Flex is hard to learn and to use** — Flex is in fact very easy to learn and use. But seeing is believing, so try this experiment:
 1. Install and launch Flex Builder, which is the Eclipse-based Flex-integrated development environment (IDE) (we'll get into that later in this book).
 2. Create a new workspace, and a new Flex project in that workspace.
 3. In the default MXML application for that project, add the code in Listing 1-1:

Listing 1-1: A simple Flex RSS reader

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
  creationComplete="RSSFeed.send();" >
  <mx:HTTPService id="RSSFeed" url="http://www.joeflash.ca/blog/feed/" />
  <mx:List id="postTitles" labelField="title" left="20" right="20" top="20"
    dataProvider="{RSSFeed.lastResult.rss.channel.item}" />
  <mx:TextArea htmlText="{postTitles.selectedItem.description}" left="20"
    right="20"
    top="190" height="200" />
  <mx:Button label="Go to page" click="navigateToURL(new URLRequest
    (postTitles.selectedItem.link));" left="20" top="400" />
</mx:Application>
```

4. Run the application.

In less than a dozen lines of code, using only four tags, you have created a fully functional RSS reader application that can grab an RSS feed, parse the XML, display the blog titles and their descriptions, and navigate to the page URL at the touch of a button. Easy, wasn't it? That is the power of Flex. We'll get more into building Flex applications throughout this book.

- ❑ **Requires the Flex Player** — There is no such thing. Flex is a development toolkit with a component framework for developing RIAs aimed at the Flash Player or AIR.
- ❑ **Requires a special server which is very expensive, and it's slow** — You may be thinking of Flex 1.0 or 1.5, which was a Macromedia product (now owned by Adobe) that existed between 2004 and 2005. This first iteration of Flex needed to be compiled on the proprietary Flex Server, which had a considerable financial barrier, and used the Dreamweaver engine as the IDE, which was considerably slower. It was a component architecture built in ActionScript 2.0 aimed at Flash Player 7, an earlier version of the language and the runtime, which did not have the capabilities nor the speed of the current version. Adobe has since reinvented the ActionScript language, the Flash runtime, and the Flex 2 SDK in 2006. Flex 3 is based on ActionScript 3.0, which is a much more robust, JIT-compiled language that runs up to 100 times faster than ActionScript 2.0. The Flex SDK, which includes the framework and the compiler, is open source and does not require a server model for compilation (although that option is available through another Adobe product), and the Flex IDE is based on Eclipse.
- ❑ **Flex does not have a mature development community** — That depends on your definition of a "mature development community," but I and many others believe this statement to be patently untrue. The recent iteration of the Flex community, started mostly by Flash developers migrating

to the (then) new ActionScript 3.0 language and the Flex 2.0 beta, began developing and sharing open-source code back in early 2005. And since ActionScript 3.0 is, according to many developers, at least as powerful as other languages such as C# and Java, developers coming to Flex from other languages add their experience in advanced programming architectures to the Flex community, which can be seen by the active discussions on Design Patterns and J2EE architectures that have been evolving on lists such as Flexcoders since 2004 (<http://tech.groups.yahoo.com/group/flexcoders/>).

The development community for the Flash Platform, upon which Flex is based, has an even longer history. Adobe (then Macromedia) coined the very term RIA back in 2002 in a white paper on Flash MX development. Since Flex is built on top of the ActionScript language, one could argue that the Flex development community is built upon a community at least that mature. The ActionScript development community has a history of open-source collaboration going back to the late 1990s (OSFlash, Prototype). As Aral Balkan, Flash platform developer extraordinaire, puts it,

“The vibrant and eclectic open-source community is one of the major legitimizing factors of the Flash Platform as a mature platform and thriving ecosystem and not some marketing construct.”

The “maturity” of a development community is not just about how long its been around, despite the fact that application development in Flash and Flex predates the very term “rich Internet application” by several years. It’s also about the diversity and vibrancy of that community. And there is no question that Flash and Flex developers are a passionate and open-minded lot sharing and participating in the ongoing evolution of the technology on a daily basis. Therefore, many would say that Flex has a very mature development community indeed.

- ❑ **Flex is not geared towards enterprise development** — *This is also untrue.* Flex is built upon a robust, ECMAScript-compliant, strongly typed OOP language, using an IDE based on Eclipse, a proven and tested enterprise development tool. Flex can interface with any Web-based service, protocol, or language, and Adobe has a dedicated Data Services server product that uses the AMF protocol, allowing the asynchronous or real-time passing of native ActionScript objects to the server with full Java interoperability. Verify for yourself the speed of this messaging protocol, by taking a look at Census, an application built by Adobe Evangelist James Ward that benchmarks about a dozen different protocols and communications formats in Flex (www.jamesward.org/census).

Other third-party products on the market implement .NET or PHP data services for Flex front-ends, and Adobe has several server products that enable scalable, real-time streaming media solutions for video encoding and playback. The Flex ecosystem, which will be covered in the next chapter, also features Adobe Cairngorm, a J2EE-compliant, enterprise-level applications framework that has become the gold standard for deploying enterprise-level Flex applications and development teams. See Part X, “Using Cairngorm,” for more information.

Adobe was voted “One of the 12 companies that will matter the most to the intelligent enterprise in 2008” by the Intelligent Enterprise 2008 Editors’ Choice Awards (www.intelligententerprise.com/channels/performance_management/showArticle.jhtml?articleID=205207028&pgno=2). And in December 2007, NATO adopted Adobe Flex and LiveCycle Data Services to deliver an enterprise-capable, highly data-intensive RIA for its Mission Support System (MSS), built to “help improve NATO’s operational readiness, by reducing the time to prepare for missions, and improve the delivery of information to flight crew members” (www.adobe.com/aboutadobe/pressroom/pressreleases/200712/121107adobenato.html). See Part IX, “Data Services,” for more information on LiveCycle Data Services.

10 Reasons to Love Flex

Now that you may be convinced of a few things that are and are not true about Flash and Flex, let's take a look at the top 10 reasons to love to develop RIAs in Flex.

1. **Flash is everywhere** — The Adobe Flash 9 Player has about 98% market penetration across browsers. This means that when you launch an application on the Web, you are virtually guaranteed that anyone will be able to use it. Compared to the distribution statistics of other RIA runtimes, there is simply no contest.

Flash is cross-OS, or “platform-agnostic”: it runs in Windows, Mac and Linux. It is also installed in every type of browser, making it the only RIA technology that can truly claim to be a “write once, run everywhere” platform.

As Kevin Merritt put it when considering AJAX for the development of blist (<http://blog.blist.com/2008/02/23/why-blist-chose-flex-and-flash/>):

I met these guys for coffee, described it, and asked if AJAX was up to the task. They confirmed my suspicion. One of them commented something along the lines of “You could probably get 80% to 90% of that functionality with AJAX, but not 100%. More important, in order to make this work across browsers and OS's, you're going to have a gargantuan JavaScript payload with all sorts of browser-specific hacks. The end result is you're going to have a buggy, fragile mess.”

Flash does not have this problem.

2. **Flex = Flash on steroids** — Flex is Flash and Flash is ActionScript 3.0 (or more specifically, Flex runs on the Flash Player, and the Flash Player's language is ActionScript 3.0.)

In ActionScript 3.0 you have all the aspects of a mature, robust programming language: strong runtime typing, class inheritance, interfaces, error handling, a built-in event model, sealed classes, method closures, custom namespace accessors, regular expressions, E4X. The Flash Player incorporates the Tamarin ActionScript 3.0 Virtual Machine with the power of a Just in Time (JIT) compiler, which interprets SWF application bytecode into machine-level instructions so that your code runs as fast as the processor can handle. Warp speed, Scotty!

And since Flex is built on top of Flash, you have the full power of the Flash APIs to draw in real time with lines, gradients and fills and manipulate and animate vectors, bitmap data, and visual assets, complete with matrix transformations, programmatic Photoshop-like filters, and blends. Flash allows communication using a dizzying array of data formats, but if you don't find the format that suits your needs, create your own using a binary socket and custom interpreter! You also get high-definition, full-screen, hardware-accelerated video capabilities, supported by enterprise-capable video encoding and streaming server products.

Flex 3 adds another layer of power onto the Flash runtime: a visual markup language called MXML, and a full-featured compiler that includes compiler metadata and data binding. The Flex framework adds class libraries for natively managing HTTP requests, RPC services, and Web Services, a dizzying array of visual UI components, a deep linking framework for browser integration, an API for AIR applications that can interact natively with the desktop, logging and unit testing frameworks, and much much more, all of which will be covered in this book. Not to mention an IDE based on Eclipse, and a whole suite of server tools for your application to communicate with the backend using native ActionScript class objects.

3. **Flex is open source** — The Flex Software Development Kit (SDK), which comprises the compiler, the component framework, and several other tools, is a free, open-source development platform. Although Flex Builder is neither free nor open source, it is built upon Eclipse, and can be installed standalone (with Eclipse built in) or as an Eclipse plug-in alongside other Eclipse development environments. In fact, Adobe Flex Builder has been voted Best Open Source Developer Tool for RIAs by *InfoWorld's* Best of Open Source Software Awards (www.infoworld.com/slideshow/2008/08/166-best_of_open_so-7.html).
4. **Interoperability** — ActionScript 3 has XML baked into it as a native format with E4X parsing capability, facilitating JSON and XML data transfer. The Flash Player is able to interface directly with JavaScript in the browser through a native JavaScript communications API, and is able to recognize SWF filename query name-value pairs. Two great examples of Flash-AJAX interoperability are Google Finance and Yahoo! Maps Canada, both of which combine the versatility of an enhanced hypertext application with the interactive power of the Flash runtime.

The Flash Player also includes XML and binary sockets capability, including traditional GET and POST HTTP requests.

The Flash Player natively supports image file formats GIF, JPG, PNG, media file formats MP3, FLV, F4P, F4A, and F4V, including the AAC, MP4, M4V, M4A, 3GP, and MOV multimedia container formats, encoded in Sorenson Spark, On2VP6, H.264, MPEG-4, MP3, or AAC.

The Flash Platform also enables some unique RIA communications protocols. Connected to the Flash Media Server, the Flash Player is also able to stream video and audio using the RTMP protocol, and its rights-encrypted cousin RTMPE, as well as the new RTMFP format in Flash 10. Using the Adobe AMF protocol, the Flash Player is able to communicate complex ActionScript objects directly with data services applications on the server. Adobe LiveCycle Data Services and its open-source cousin BlazeDS enable bidirectional ActionScript-to-Java object transfer, and the ColdFusion server allows for ActionScript-to-CFC object transfer. Third-party data service implementations such as WebOrb, AMFphp, and Zend enable native ActionScript object communication with the .NET and PHP server languages. Other third-party data services solutions exist for languages such as Ruby and Python.

The Flex framework contains APIs that allow very easy deployment for HTTP requests, Web Services, and RPC services data transfers. So needless to say, a Flex application can communicate a wide variety of web technologies, allowing for a plethora of server integration options with an impressive array of interoperability with many data formats, languages, and protocols.

5. **The community** — Conceived with the same spirit as the Flash community, the Flex community is passionate, generous, and vibrant, always coming up with new ways to alert each other of the latest quirks, share their discoveries, their tips, their components, and their experiments, helping Adobe make a better product. And Adobe does an incredible job of feeding that fire: it actually listens to the community, and rewards them with changes to product development that directly reflect what designers and developers has been asking for, going so far as to organize events and surveys that elicit feedback for the sole purpose of making Flash and Flex better and better. One has only to look at all the new features in Flash Player 10 and those planned for Flex 4 to know that Adobe has listened to and cares about its community. Many Flash and Flex designers and developers, including the writers of this book, really enjoy what they do for a living, and thrive on that energy. And this level of passion and commitment to each other and to the evolution of the platform shows through in every Flash and Flex conference, every technical blog, every community forum and list this author has ever visited.

6. **Creative Suite integration** — When Adobe purchased Macromedia in 2005, it brought to the Flash and Flex development scene the possibility of the full force of its Creative Suite of applications. As of Creative Suite 3, we can now experience the fruits of that promise, as we now have seamless integration between Flash and Illustrator, Fireworks and After Effects. This opens up a vast sea of creative possibilities for Flex development, enabling even “richer media” applications than ever before. See Chapters 26 and 29, which cover Flash and Flex workflow integration in greater detail.
7. **Flex is easy to learn** — *for Flash developers* — For Flash developers with experience with ActionScript 3.0 and OOP principles, Flex can be considered merely another component framework with some new authoring tools. Most ActionScript developers already use a third-party coding editor such as FlashDevelop or FDT, so migrating to Flex Builder is easy, especially for those already using Eclipse for JavaScript, PHP, Java, or ColdFusion development. And for those comfortable with their existing tools, both FlashDevelop and FDT work well with the open-source Flex SDK (framework + compiler). Many Flash developers also use Flex Builder for coding ActionScript projects compiled with the Flash CS3 compiler, so there is much cross-workflow potential (see Chapters 26 and 29 for more detail on Flash-Flex integration). Flex has better GUI components, a better debugger, a memory profiler, a more versatile compiler, and there’s no timeline to complicate your project structure or code execution. And that’s just for starters.

And, contrary to the popularly held belief among some Flash developers, Flex does not make Flash and pure ActionScript development obsolete. Far from it. In fact, the sheer explosion in RIA development in Flex in recent years has only increased the need for rich asset creation only a Flash developer can provide, and the deep knowledge of ActionScript that is common to experienced Flash developers. As Adobe Evangelist Ely Greenfield puts it in her article “Why the Flex in FlexBook, or ... Why a Flash Author should care (a lot!) about Flex” (www.quietlyscheming.com/blog/2007/03/14/why-the-flex-in-flexbook-orwhy-a-flash-author-should-care-a-lot-about-flex),

“There’s a lot of incredibly talented individuals out there who know how to create amazing visual effects with ActionScript and the timeline. There’s a whole lot more incredibly talented developers out there who really do not have the skill-set to create those effects, but would really like to use them in their Flex applications. ... I believe talented Flash dev-igners could make good money selling their services into the Flex community. ... if the Flash community considers bringing some of that creativity to bear on the growing Flex ecosystem.”

for Java developers — I’ve heard it said how uncanny is the resemblance between ActionScript and Java. To such an extent that after a brief learning curve (and I do mean very brief), many Java developers find themselves building fully functional Java-enabled Flex apps in no time at all. Flex Builder is an Eclipse plug-in, which fits right at home in a traditional Java workflow. In addition, Adobe’s LiveCycle Data Services and BlazeDS are both Java-based server solutions, enabling a high degree of interoperability between ActionScript and Java objects. Adobe also has a developer resource page dedicated to Flash-Java integration articles, tutorials, and white papers. See Parts VIII, “Server Integration,” and IX, “Data Services,” for an in-depth look at Flex, Java, and data services. Appendix A, “ActionScript 3 Language Comparison,” also provides a comparison table between ActionScript and Java.

and for anyone else used to the Eclipse-based environment or enterprise-level applications development.

8. **Flex enables modular, rapid application development (RAD)** — Flex natively encourages a Model-View-Controller (MVC) separation of application parts through the use of MXML, which is an XML-based visual layout markup language that facilitates rapid prototyping and development (see Listing 1-1 for details). In the simplest Flex MVC pattern, the *View* or interface layout is typically coded in CSS and MXML, and the *Controller* or application logic is typically coded in ActionScript, though one is not constrained to these norms. This will be covered in greater detail in Chapter 60, “MVC Frameworks.” Flex Builder also has a *Design View*, which allows for the visualization of MXML layouts, and a *Properties View*, which allows for the setting of inspectable component properties directly in the IDE (see Chapter 5, “Introduction to Flex Builder 3,” for details).

Flex also enables a variety of modular compilation and deployment methods. You can precompile application modules or class libraries to increase compilation efficiency, code distribution, or asset management. By compiling all your fonts into one module, and all your skins into another module, for example, you cut down overall application compilation time and increase asset management efficiency. You can compile SWFs to be compiled in the main application as embedded assets, which can be easier to use than Flex modules, or you can opt to load SWF application subcomponents at runtime, decreasing the initial application filesize.

The application footprint can also be mitigated by the use of *persistent framework caching*, where you can compile your application sans the Flex framework, allowing the Flash Player runtime to preload and cache the framework as a separate class library on the client. This means that after the Flex framework has been downloaded once, it is cached on the client, and your application will load much quicker for the user. See Chapter 66, “Modular Application Development,” for details.

9. **Adobe AIR** — AIR brings the power of Flash and Flex to a whole other level. With AIR, AJAX applications can be deployed to the desktop, or a Flash/Flex application can be deployed to the desktop, in either the PC, Mac or Linux OSs, with access to system windowing and local file system interactivity. Or even more powerfully, both AJAX *and* Flash can be leveraged together in the same application, in ways that could never be done in a conventional browser environment. You can apply Flash bitmap filters and tweens directly onto an HTML object, or have a JavaScript object communicate directly with an ActionScript object without the intermediary of a communications format. This enables the development of flexible, “sometimes connected,” desktop solutions that have not previously been possible in the realm of RIAs.
10. **Seeing is believing** — But don’t take our word for it. Check out some Flex and AIR applications yourself. Have a look at some of the showcased applications, and you’ll see exactly what we mean:

www.flex.org/showcase
www.adobe.com/products/air/showcase

Summary

Hopefully throughout this chapter, you have gained an increased understanding of what Flex is and where it sits in the overall scheme of RIA development. In this chapter, we also looked at clearing up several misconceptions you might have about the Flash Platform and Flex, including some very compelling reasons for using Adobe Flex for all your RIA development requirements. The delight is in the details, of course, as the rest of this book will show you.

