

# FLEX4 IN ACTION

Tariq Ahmed  
John C. Bland II.  
Dan Orlando

MEAP

 MANNING





**MEAP Edition**  
**Manning Early Access Program**

Copyright 2009 Manning Publications

For more information on this and other Manning titles go to  
[www.manning.com](http://www.manning.com)

## TABLE OF CONTENTS

### PART ONE: APPLICATION BASICS

CHAPTER ONE: INTRODUCTION OF FLEX

CHAPTER TWO: GETTING STARTED

CHAPTER THREE: WORKING WITH ACTIONSCRIPT

CHAPTER FOUR: LAYOUT AND CONTAINERS

CHAPTER FIVE: DISPLAYING FORMS AND CAPTURING USER INPUT

CHAPTER SIX: VALIDATING USER INPUT

CHAPTER SEVEN: FORMATTING DATA

CHAPTER EIGHT: HALO DATAGRIDS, LISTS AND TREES

CHAPTER NINE: SPARK LISTS

CHAPTER TEN: LIST CUSTOMIZATION

### PART TWO: APPLICATION FLOW AND STRUCTURE

CHAPTER ELEVEN: EVENTS

CHAPTER TWELVE: APPLICATION NAVIGATION

CHAPTER THIRTEEN: INTRODUCTION TO POP-UPS

CHAPTER FOURTEEN: VIEW STATES

CHAPTER FIFTEEN: WORKING WITH DATA SERVICES

CHAPTER SIXTEEN: OBJECTS AND CLASSES

CHAPTER SEVENTEEN: CUSTOM COMPONENTS

CHAPTER EIGHTEEN: ADVANCED REUSABILITY IN FLEX

### PART THREE: THE FINISHING TOUCHES

CHAPTER NINETEEN: CUSTOMIZING THE EXPERIENCE

CHAPTER TWENTY: WORKING WITH EFFECTS

CHAPTER TWENTY-ONE: DRAG AND DROP

CHAPTER TWENTY-TWO: CHARTING

CHAPTER TWENTY-THREE: DEBUGGING AND TESTING

CHAPTER TWENTY-FOUR: WRAPPING UP THE PROJECT

# 1 Making the case

*This chapter covers:*

- Solving problems with Flex
- Using RIAs and RWAs
- Comparing Flex to the competition
- Learning the Flex ecosystem

This chapter makes the case why Flex is a great addition to your personal skill set or your organization. With buzzwords flying all over, a nonstop stream of websites with missing vowels in their names, and the Web 2.0 space on fire, a hodgepodge of technologies leaves the common developer caught in the middle. It is vital to be able to defend the decision to move forward with Flex to both customers and management.

In this chapter, we'll talk about challenges that a web developer faces and how to solve them using Flex by Adobe. We'll also get into the mechanics of a Flex application and discuss the ecosystem as a whole.

## **1.1 Why are web applications so prolific?**

Web applications are so prolific because the strength of the web is also its weakness. The original intent of the web was to be a lightweight information distribution system: a simple and platform neutral way (any OS, any hardware) to post documents on a server and retrieve them just as easily.

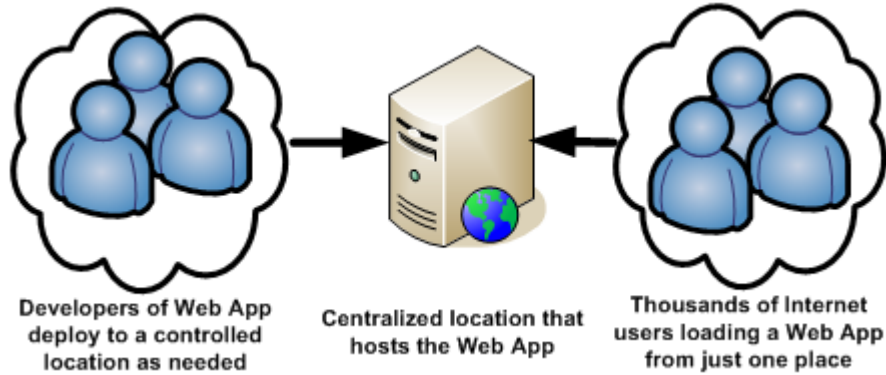


Figure 1.1 The great web advantage—centralized deployment

This advantage of centralized deployment (figure 1.1) inherited by dynamic pages (i.e. web applications) provides such a strategic value in both a business (e.g. ROI) and development perspective that it makes the thought of developing old school desktop applications difficult to justify.

Yes, desktop applications are rich and robust; you can do anything the OS permits. But their deployment model is a nightmare. The logistical complications of trying to get thousands, if not hundreds of thousands, of clients to run the precise version of your software at the exact same time are immense.

#### NOTE

This may all feel obvious to some of you, but part of being successful with Flex is being able to articulate the business case to management and teammates. At the same time, knowing the problems that Flex solves helps better understand the technology.

Of course, with the web, you can release enhancements and fixes as fast as you can code them. Now all your users can take advantage of the latest and greatest updates transparently.

Seems like a no-brainer right? As you know, technologies quickly become obsolete, yet the centralized deployment model of web applications is so effective that we've continued to use its HTML 4 language since 1999 (HTML 5 is supposedly due by 2010).

During all this time, one critical element was overlooked: the user experience (figure 1.2). Users willingly gave up usability for the ubiquity of web applications. Ultimately, we trained ourselves and the users to accept it.

True, there was an emphasis in the web development community prior to the AJAX revolution to utilize JavaScript to improve usability by employing a more progressive approach, but the core of the usability problem lies in the historical roots of the web—its structure has been built around what was intended to be a documentation-distribution mechanism.

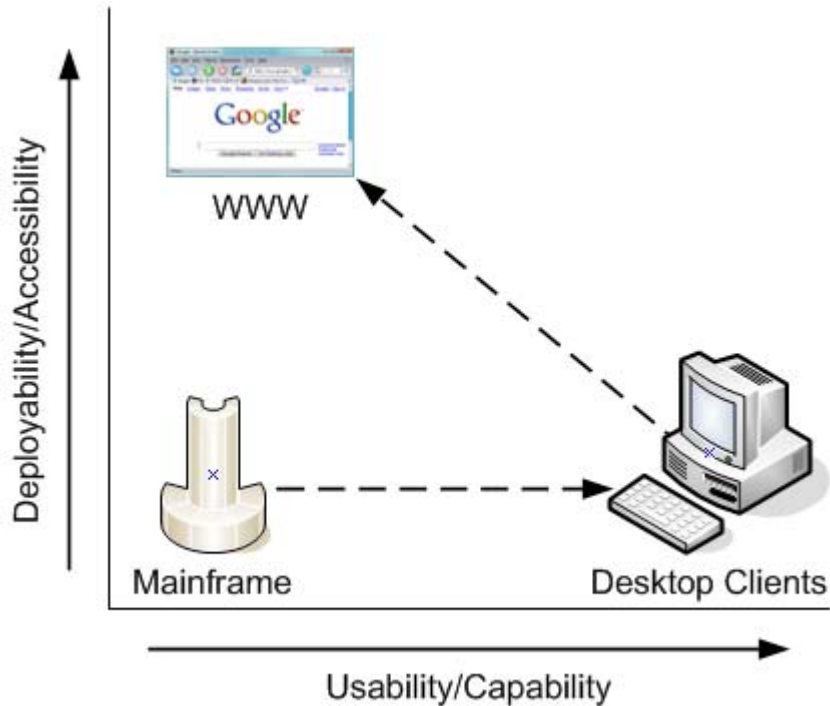


Figure 1.2 We took a great step backward in terms of usability for the sake of deployability.

As a developer, you exert a significant amount of effort to restore some semblance of usability by transferring as much application logic as possible to the front-end (client side) to mimic a client (desktop-like) experience.

The web was supposed to be platform-agnostic, yet ironically the more you push logic to the client side, the more you struggle with browser incompatibilities. This is where rich internet applications (RIAs) come into play.

## 1.2 The RIA solution

In this data-centric society, users and businesses depend on being able to work with information efficiently. Users simply want information quickly and easily. Businesses, from a customer-retention perspective, want to provide a better user experience than the competition, and need the technology to ensure the workforce is functioning productively.

In a sense, you now have a paradox: users wanting a pleasant experience and businesses trying to achieve high-feature velocity and operational efficiency. This is the case with traditional technologies and the divide upon which RIAs capitalize.

### 1.2.1 They all want it all

Users want to be able to access their data from whatever computer they're on. They also want to be able to do such trivial things as dragging and dropping. They want a rich, fluid graphical experience that incorporates sound and video. But they *don't* want to be constantly nagged to download the latest version.

Developers and software teams want it all too. Time-to-market is of high strategic value, whereas software development and maintenance is enormously expensive. Developers want to create software quickly and not worry about how to make it work on various platforms. They want the process of pushing out updates to be easy and fast.

Figure 1.3 summarizes who wins—and who loses—in each scenario.

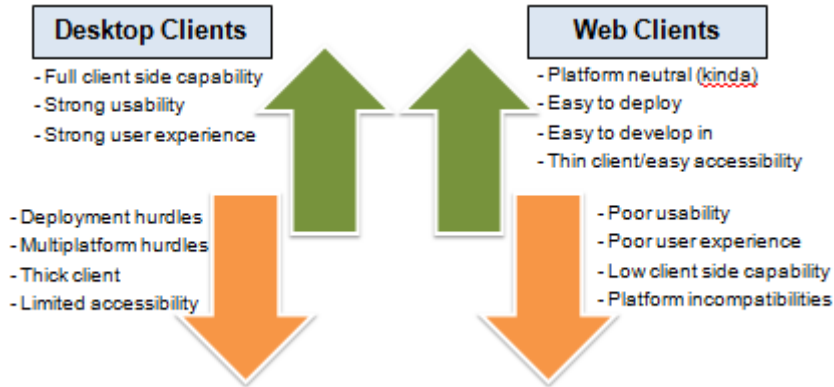


Figure 1.3 Pros and cons of desktop and web clients—choose the lesser of the evils.

But with problems or challenges come opportunity, and this is the opportunity that RIAs seize by providing the best of both worlds.

### 1.2.2 RIAs to the rescue

RIAs solve this problem by incorporating the best of both worlds. RIAs are a technology that gives businesses feature velocity and rapid deployment through the centralized internet deployment model, while providing users a desktop-like experience (figure 1.4).

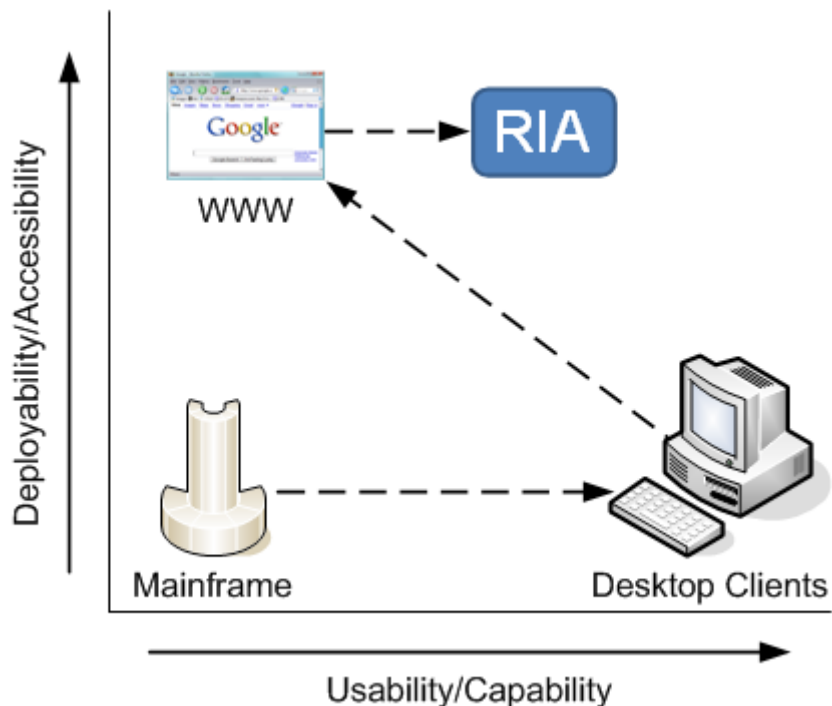


Figure 1.4 RIAs add the best of both worlds

RIAs bring back usability by enabling developers to give their users a compelling and fluid experience with that feeling of a live application (versus completely reloading a page every time you click something). That's the core ingredient to providing users a sense of engagement.

At the same time, the deployment and accessibility model remains the same—users can load these applications from any machine and all be running the same version. The best part is true platform neutrality; the same application yields the same look and feel regardless of environment.

### **1.2.3 How RIAs do it**

RIAs are able to accomplish this by simply not being an interactive document, and thus have none of the restrictions of one. They accomplish this via the use of a browser plug-in that acts as a local runtime engine. With a runtime engine available for various browsers and operating systems, you're able to achieve platform neutrality.

Since it is a plug-in it can piggyback onto the browser. Using the browser as a delivery mechanism gives the plug-in the high degree of deployability that web applications enjoy.

## **1.3 The RIA contenders**

The RIA space is hot right now, and contenders are standing in their respective corners. In one corner, you have the front-runner, Flex by Adobe who faces Microsoft's Silverlight and Sun's JavaFX. From a technical perspective you could argue that AJAX is more of a rich web application technology, than a rich internet application technology; but due to the big advancements in the AJAX toolkit arena it's worth adding to the list.

Here's a brief summary of the major RIA contenders.

### **1.3.1 Flex by Adobe**

First out of the gate, Adobe has maintained a fierce pace in expanding this platform. With Flex 4, Adobe made the framework open source; the software development kit (SDK) has been free since Flex 2, and the price point for the optional IDE is attractive.

Flex applications truly are rich *internet* applications; they're platform-agnostic, internet-deployed thin clients. Flex supports multiple transfer protocols such as text/XML, web services, RTMP/Messaging, and the binary format known as Action Message Format (AMF). It also has a robust charting engine, can stream video natively, and do much more.

Flex has the following things going for it:

- It leverages the nearly ubiquitous Flash Player, which has a 98% penetration level.
- The huge Flash ecosystem (existing forums, community, and knowledge).
- Tight integration with other Adobe products from designer (Photoshop, Fireworks, and so on), to developer, to server (ColdFusion, media streaming, and so forth).
- A four-year head start.
- Open source framework and SDK.

On the downside:

- Although Flex's printing abilities are satisfactory, there is a lot of room for improvement (particularly with respect to report-style printing).
- Because the technology is still relatively new, the size of the community is relatively small compared to that of .NET and Java.

Although heated arguments in discussion forums erupt whenever technologies are compared against each other we'll next venture onto Flex's main threat, Silverlight by Microsoft.

### **1.3.2 Silverlight by Microsoft**

Microsoft isn't well known for being an early innovator, seemingly preferring to invest enormous amounts of capital to dominate only after others have invented market spaces. But once they set their minds on something, they're willing to do what it takes to win.

Microsoft released Silverlight 1.0 in September 2007, 2.0 in October 2008, and Silverlight 3 Beta was launched March 2009. So they're moving at a blistering pace to capture as much market share as possible.

Comparing features of the Silverlight 3 technology vs. Flex 4 wouldn't be of much value as they roughly are on par with each other. The key differentiators are their respective ecosystems that each have their own distinct advantages.

Adobe dominates the design industry with their Creative Suite tools (Photoshop, Illustrator, Fireworks, etc...), and they have another tool coming called Catalyst that can convert a Creative Suite image into a fully functioning Flex application.

Microsoft on the other hand has a software suite for designers called Expression, which goes head to head with similar tools from Adobe. But you'd be hard pressed to find designers who've heard of Expression, let alone know it well. So in terms of the design to developer workflow, Adobe has the edge.

However, Microsoft has the formidable server software side of things. The .NET universe is a large one, and developing on the Silverlight platform allows you to tap into all things .NET (IIS.NET, Sharepoint, etc...).

Adobe does have a server aspect to their business with ColdFusion, LiveCycle Data Services, and their cloud computing initiatives. But compared to .NET, Microsoft has the edge.

From a platform perspective, Silverlight is available on Windows and Mac OS X, along with browser plug-ins for Internet Explorer, Firefox, and Safari. Opera is unofficially available, and the Chrome plug-in is only available on the Windows version.

Flex on the other hand runs wherever the Flash player can be found, and that's all combinations of major OS's and browsers.

The bottom line is if you're already a .NET developer you'll be able to leverage your existing code base and skill sets by going the Silverlight route.

### 1.3.3 JavaFX by Sun Microsystems

Sun had the opportunity to be the RIA leader with Java but failed to capitalize on it, rendering JavaFX pretty much a second attempt. JavaFX is also a late comer to the game with their first release having come out in March 2009, and as a result have a lot of catching up to do.

JavaFX does have the fortunate position of being able to leverage the commanding ecosystem of the enormous Java community, but it's still too early to tell if the technology will be able to capitalize on it.

The big question going forward is now that Oracle (who is a big adopter of Flex for the UI of various tools) owns Sun, what will they do with JavaFX? Time will tell!

### 1.3.4 AJAX—the last stand

Impressive results have been achieved using extremely advanced JavaScript and Dynamic HTML (DHTML) techniques in combination with the browser's XMLHttpRequest function. But in the end, it is a last great effort to squeeze every possible ounce of application usability from a document platform.

Some amazing things have been done despite this limitation with AJAX toolkits such as Adobe's own Spry, EXT, YUI, and the formidable jQuery. These JavaScript toolkits go to great efforts to enable you to develop highly interactive applications, and are a good option for making web applications more usable.

#### NATIVE ABILITIES VS. HACKS

But consider this, what if you want a three-state check box? Figure 1.5 shows how Microsoft Excel uses this form element to present three possible states, whereas HTML's checkbox only supports two (selected and unselected).

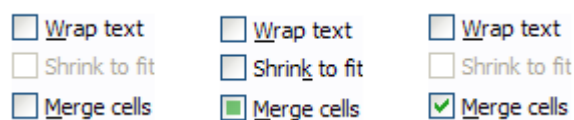


Figure 1.5 In Flex, you can make a three-state ComboBox just like that of Microsoft Excel. In HTML, the option to extend

components isn't available.

If you wanted to do this in HTML you can fake it using a graphic, JavaScript, and some CSS. In contrast, an RIA engine such as Flex or Silverlight, affords you the ability to extend the native checkbox to have three states.

Take that even further and ask yourself if JavaScript can:

- Play sound files natively?
- Decode a compressed video stream?
- Interact with devices such as a webcam?

As we mentioned earlier, impressive innovations have resulted from this last-hurrah squeeze, but they've come at a price that is incurred by platform inconsistencies.

#### CROSS-PLATFORM ISSUES

If you've worked with JavaScript, you know it is a massive headache to support multiple browsers. Every browser has variations, such that if you utilize advanced AJAX/JavaScript techniques, the result would require a significantly increased quality-assurance cycle (figure 1.6).

This deficiency alone severely impairs the ability of a business or customer to achieve the desired return on investment (ROI); the biggest single cost of systems development is developer time.



Figure 1.6 Don't forget to factor in the development costs of supporting multiple browsers when using AJAX.

#### AJAX COMMUNICATION LIMITS

AJAX supports just one thing: text over HTTP. Traditionally that text has been in the form of XML, and the more recent trend is to use JSON to reduce client-side processing. In either case both are inefficient for transferring large amounts of information due to the verbose amount of text.

Try transferring 5,000 records using XMLHttpRequest, then parsing them in JavaScript. That's exactly what Adobe's technical evangelist James Ward did, and you can see the results posted on his blog (<http://www.jamesward.org/wordpress/2007/04/30/ajax-and-flex-data-loading-benchmarks/>).

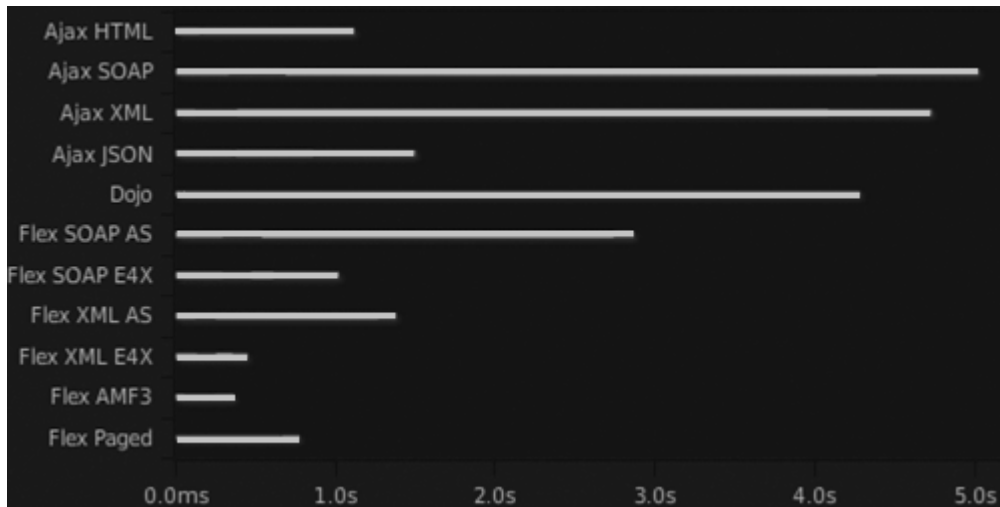


Figure 1.7 The speed of Flex's binary protocols versus the verbose approaches of AJAX

In figure 1.7, you can see how Flex—using its binary format, AMF3, versus XML or SOAP in AJAX—is up to 10 times faster. There are two reasons for this. First, as just mentioned, XML is verbose, which results in quite a bit of overhead in the message payload. Second, JavaScript is an interpreted language, whereas Flex is compiled to platform-neutral byte code.

Beyond speed, Flex's compact binary format offers other advantages. Less overhead means less demand on resources such as memory and network utilization.

#### NOTE

Keep in mind that Flex and Flash do support simple transfer of textual data over HTTP and SOAP web services.

AMF3 provides a clear advantage, particularly for large-scale applications or applications that need to exchange large volumes of data.

The fact that JavaScript and HTML haven't had to evolve for many years is a true testament to their longevity. They have stood the test of time and will continue to thrive in some form or another for decades to come. This is because the web is great at what it does—distributing platform-neutral documents.

Web applications will have their place too, but the trend is clear that users are demanding more. True RIA technologies will take over where web applications leave off.

## 1.4 Becoming acquainted with Flex

Flex is a programmatic way (you use code) to make RIAs leverage the Flash platform. Flash, of course, is famous for interactive banner ads, cool animated portions of web pages, and interactive marketing experiences, which are often used for promotional sites.

Flex has a head start with its ability to leverage the widely recognized and mature technology of Flash Player in addition to taking advantage of its widespread deployment. But it doesn't lock you out of the HTML world; you can have Flex interacting with web applications using JavaScript, while at the same time being a part of the large Adobe technology ecosystem.

### 1.4.1 Taking advantage of Adobe Flash

At the heart of Flex execution is Adobe Flash Player. This incredibly powerful, fast, lightweight, and platform-agnostic runtime engine is based on an object-oriented (OO) language called ActionScript. The experience is the same for Mac users as it is for Windows users—or users of any platform, for that matter (smart phones, PDAs, and so on).

The Flash Player on which Flex applications execute is capable of processing large amounts of data and has robust 2D graphical-rendering abilities, multithreaded processing, and support for various communication protocols. Flash puts the rich in RIA by supporting multimedia formats such as streaming video, images, and audio.

The end result is the ability to provide a rich desktop-like experience that allows developers to be innovative and creative. It also lets you present unique approaches to optimize the workflow for the end user.

#### WHY NOT DO IT IN FLASH?

Savvy Flash developers were making RIAs before Flex existed. But those coming from the development world and trying to get into Flash found it difficult to adopt the Flash mindset. Because Flash's roots are in animation (figure 1.8), its environment is based on timelines, layers, frames, frames per second, and so on. It is somewhat strange for someone with a development background based in lines of code to think of an application being a movie.

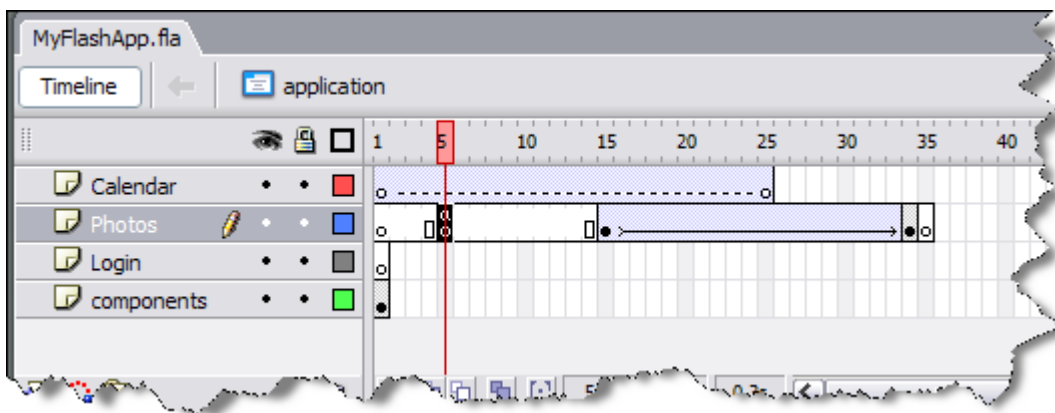


Figure 1.8 Flash has always been capable of making RIAs, but can you imagine coding based on time?

Even for seasoned Flash veterans, the cost of developing applications purely in Flash is significantly more than in other development environments (mostly due to the intensive work required to deal with change).

Although Flash and Flex can function as standalone applications, they can also interact with web applications by using JavaScript as a bridge between them.

#### 1.4.2 Flex and JavaScript can play together

If you've been developing HTML-based web applications and using JavaScript, as you get into Flex you'll notice that its ActionScript language looks incredibly similar.

That's because JavaScript and ActionScript are based on the ECMAScript standard. If you've used JavaScript extensively, you'll find comfort in familiarity. One interesting tidbit is that Flex's ActionScript was the first production language to adopt the current ECMAScript 4 standard.

In working with Flex you are not locked into technology silos. That is, it doesn't have to be all Flex or bust. Although RIAs and RWAs are different, Flex allows you to operate between technologies. To do this, Flex employs a feature called the External API, which enables JavaScript applications to communicate with Flex applications.

In the context of AJAX, Flex has an additional feature called the Flex-AJAX Bridge (a.k.a. FABridge) that makes it easy to integrate AJAX and Flex applications. If you have a significant investment in an existing AJAX application, but would like to leverage Flex's capabilities, you can use Flex to generate interactive charts from your AJAX application.

The harmony doesn't stop there; because Flex is made by Adobe, it also fits into a larger encompassing suite of technologies.

### 1.4.3 The Flex ecosystem

The amazing technology built into Flex is not its only major advantage. Because it is part of a set of technologies from Adobe, your organization can achieve a smooth workflow from designer to developer to deployment. Figure 1.12 shows where Flex sits in relation to the entire graphics suite landscape.

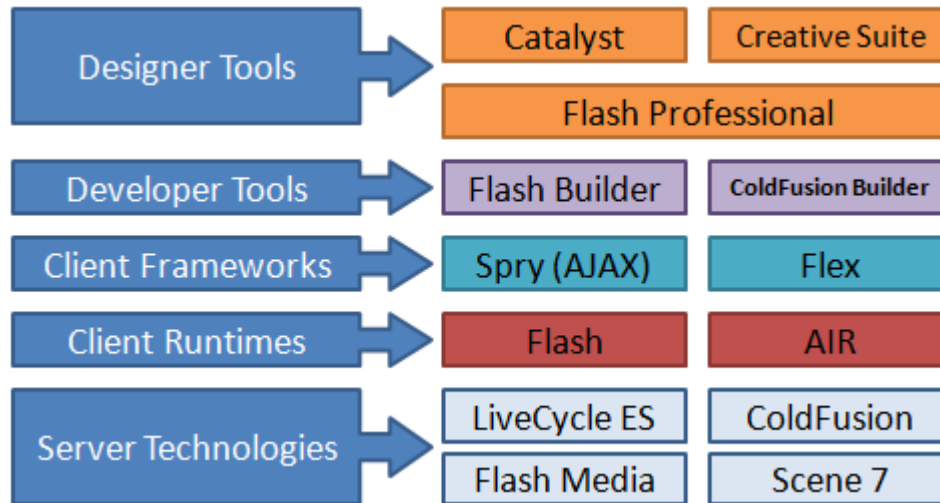


Figure 1.9 Flex is part of a big technology stack.

In the land of regular web development, a lot of time is wasted bouncing between designers and developers. As you may know, designers use tools such as Photoshop to design what the application is like, and developers laboriously slice up the images and generate CSS.

But in the Flex ecosystem, designers can export themes (skins), and developers can import them without tightly coupling the application to the design. And it keeps getting better. Catalyst, the most recent addition to the Adobe Creative Suite family, will allow you to convert an image of an application (e.g. from Photoshop) and help you convert it into a Flex application.

Another client technology that has garnered a lot of press and is directly related to Flex is AIR, which allows your Flex applications to run as native desktop applications.

#### A BLURB ON AIR

AIR stands for Adobe Integrated Runtime. AIR allows you to go one step further by transforming your Flex RIAs into what we call rich desktop applications (RDAs). Although there's no official term for it, this type of technology is also known as a *hybrid desktop internet application*.

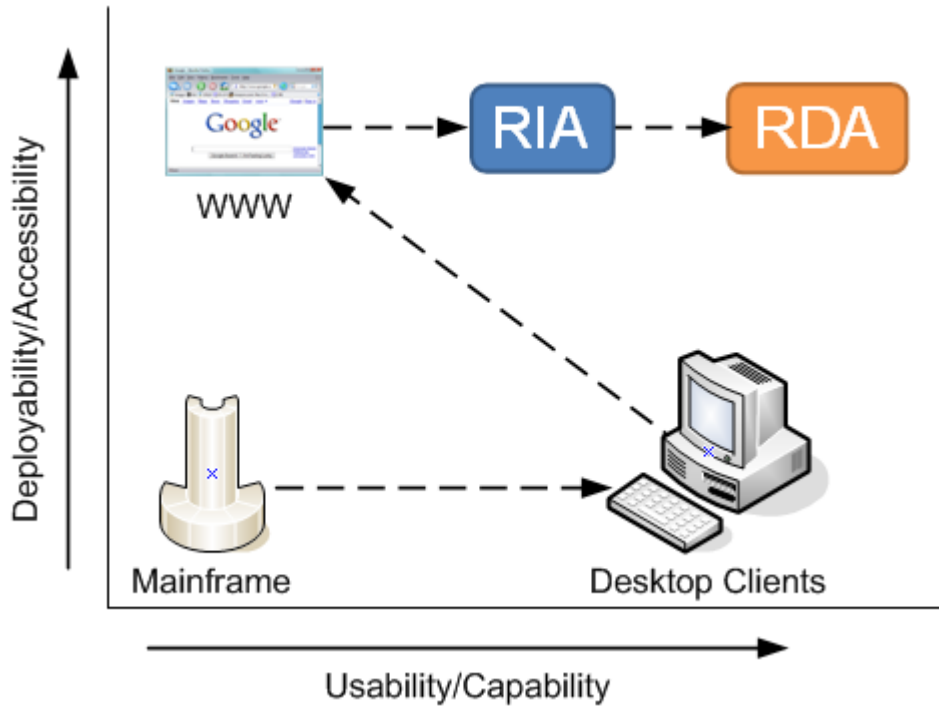


Figure 1.10 RDAs go that extra mile by achieving the desktop experience that RIAs are not able to reach. RDAs exist outside of the browser, and like a desktop application have access to the operating system's clipboard, and local file system.

Because Flex is launched via the browser, for security reasons it is limited in its ability to do certain things such as accessing data on hard drives or interacting with peripherals like scanners. AIR liberates Flex applications from the browser and lets them execute directly on the desktop (figure 1.10), giving you the full desktop experience. With AIR, you can perform functions such as:

- Access cut-and-paste information from the operating system's clipboard
- Drag and drop from the desktop into the application
- Create borderless applications that do not require the square frame of a browser around them

AIR provides additional capabilities:

- Built-in database server
- Transparent and automatic software updates to ensure everyone is using the same version
- Built-in HTML rendering engine

It is a revolutionary platform. If you decide you want to take your Flex applications beyond the browser, check out Manning Publications's *Adobe AIR in Action* (Joey Lott, Kathryn Rotondo, Sam Ahn, and Ashley Atkins, 2007; ISBN: 1933988487).

#### **A BLURB ON BLAZEDS**

BlazeDS is a trimmed-down version of LiveCycle Data Services (LCDS) from Adobe. It is a middle-tier server component that acts as a middleman between back-end components and services (other server technologies like Java and .NET), as well as connectors to database servers and messaging technologies such as Java Message Service (JMS).

Its capabilities include:

- Transferring back-end data to the Flex client using the binary AMF3 protocol.
- High-performance data transfer.

- Real-time data push using HTTP and AMF3. (It can notify your Flex application about new data, instead of your Flex application polling for new information.)
- Publish/subscribe messaging. (Through a technique known as long polling, RTMP is only available in the LCDS product.)
- Recordset paging with a database. (It can stream 50 database records at a time, or do the last/next 10 records from a query.)
- Data synchronization. (If your Flex application modifies data on the client side, it can automatically notify BlazeDS to update the corresponding record in the database in real time.)
- The best part is that it is free! As another piece of Adobe's open source software, this technology is available and can be distributed under the LGPL v3 license.

#### **1.4.4 How Flex works**

At the heart of Flex is a free SDK that provides the framework for making Flex applications. In a nutshell, it is all the out-of-the-box libraries and the compiler.

On top of that is the Eclipse-based IDE named Flex Builder. Instead of using the Flash editor to make Flash applications, you use Flex Builder.

Flex comprises two programming languages:

- The XML-based MXML tag language. (No one knows what MXML stands for, but two popular assumptions are Macromedia XML and Magic XML.)
- The ActionScript scripting language.

When developing in Flex, you use both: MXML for primary layout of the application core (the visual components) and ActionScript to script out all the logic needed to drive your application.

Although it is not particularly pertinent to our discussion at the moment, MXML is compiled behind the scenes into ActionScript. This means you can make a full-fledged Flex application using only ActionScript. (That's what it ends up being anyway.)

#### **TIP**

New users struggle to determine when to use ActionScript and when to use MXML. A simple rule of thumb is to pretend HTML is like MXML, which allows you to visually lay out how you want your application to initially appear. Then, think of ActionScript as JavaScript—it adds the brains to your application. As you become comfortable with it, you'll find you can make entire applications using nothing but ActionScript.

Using the two languages, you create your application by compiling it into a single executable file that is then deployed onto a web server (figure 1.11). Listing 1.1 has a simple example of how these two languages are related.

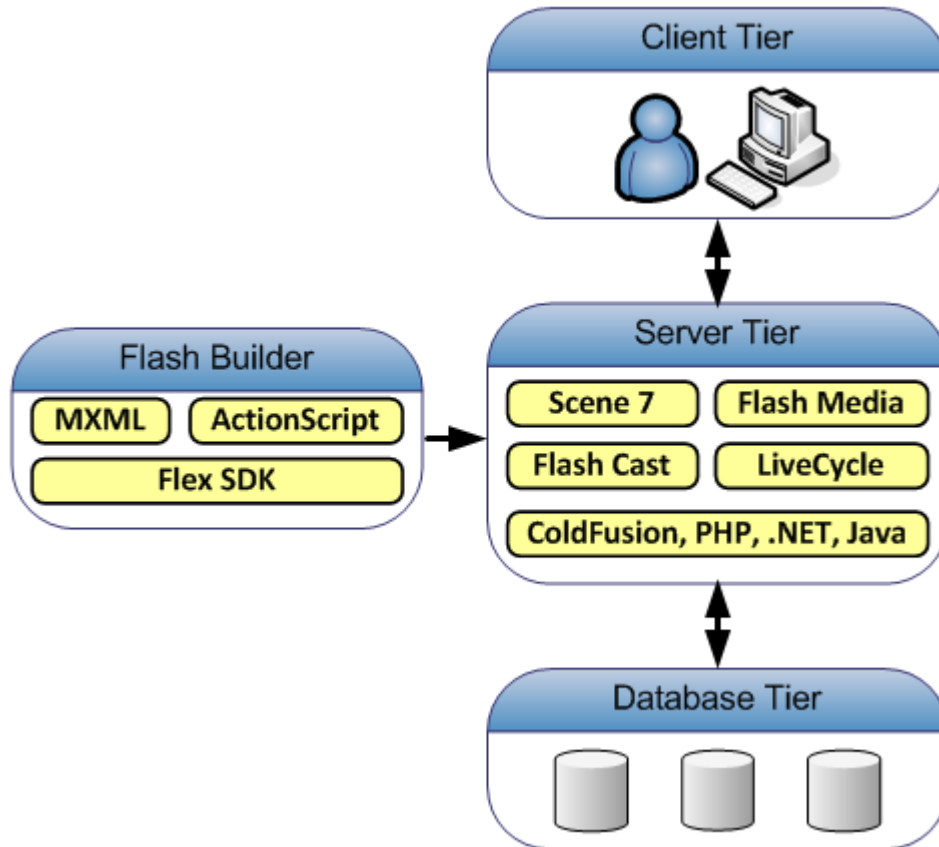


Figure 1.11 Use Flex Builder to compile your application, and then deploy it to a server.

#### Listing 1.1 Example of a simple 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/halo">

  <fx:Script>
  <![CDATA[
    import mx.controls.Alert;
    public function handleEvent():void
    {
      mx.controls.Alert.show("Event handled");           #1
    }
  ]]>
  </fx:Script>
  <s:Button label="Click on me" click="handleEvent()"/> #2
</s:Application>

```

#1 ActionScript for scripted logic  
 #2 MXML for layout of stuff

## Typesetter: Cueballs in text

Even if you've never seen Flex code, this sample may feel somewhat familiar. One reason is the tag-based MXML language (#2) is a derivative of XML. Another reason is that the script logic (#1) is similar to JavaScript.

Let's look at the typical lifecycle of the development process.

### THE DEVELOPMENT LIFE CYCLE

This is what the Systems Development Life Cycle (SDLC) of a typical Flex application looks like:

1. Using Flex Builder or the SDK, build in your local development environment by writing MXML and ActionScript code.
2. When testing, use Flex Builder or the SDK to compile your code. Doing so generates the main output .swf file (often pronounced *swiff* file).
3. Use the browser to launch this file, thus invoking the Flash Player plug-in. Your application begins to execute.
4. A Flex application typically interacts with a server tier to exchange data.
5. When the application is ready to be released, the .swf (and any accompanying files, such as images) is published to your production web server, where it is available to be invoked by your users via a URL.

For those who work with application servers like ColdFusion and PHP, note you're not pushing the source files to production, but rather a compiled application (similar to Java's .class files, but a Flex application also contains all the libraries needed for the application to work).

#### 1.4.5 Events, events, events

It is all about events. Flex is an event-driven environment, which may be a big departure from what you're used to.

In traditional web development technologies, an event represents an action such as a user clicking a link or a submit button. The server responds, executing whatever function is required—in this case, displaying a web page or sending field data.

If you've been developing web applications, you've undoubtedly created JavaScript that responds to certain user gestures such as highlighting an item on a page by changing the background color of the item as the user moves the mouse over it (figure 1.12).

```
<a href="" onMouseOver="alert('Howdy!')">mouse over me</a>
```

The diagram shows the HTML code `<a href="" onMouseOver="alert('Howdy!')">mouse over me</a>` with two brackets underneath. The first bracket is under `onMouseOver=""` and is labeled "Event Trigger". The second bracket is under `alert('Howdy!')` and is labeled "Event Handler".

Figure 1.12 Flex, like JavaScript, uses events that consist of triggers and handlers.

That's an event! But that term isn't used as much in traditional web applications because the majority of the application resides on the back end. As in the JavaScript example we just described, Flex is a client-side technology, meaning all the action occurs on the user's side.

A Flex application is driven entirely through events; something causes something to occur, and something else handles it when it occurs. The two main pieces of an event-driven application are:

- *Event triggers*—Triggers cause events: the user moving the mouse over a button, the application loading, data coming back from a web service, and so on.
- *Event handlers*—Handlers respond to events: invoking a function that changes display characteristics, committing an input form, and so on; handlers are where the logic is.

Coming from a traditional web technology, you won't be used to thinking like this. We'll gradually introduce how events are used, but you'll need to let go of the web application notion of generating pages. With Flex, the application is already loaded; all you're doing is capturing events and responding accordingly.

### 1.5 What's new in Flex 4

Since its inception, Flex technology has been evolving at an incredibly rapid pace. The product started as a pure server-side technology, modeling the conventions of most server-based web application technologies.

Flex 2 was a major overhaul of the language. Adobe split it into two portions: the client portion consists of the application framework and tools to compile a Flex application; the server portion is the data bridge. This has evolved into LCDS/BlazeDS. The Flex 2 overhaul also involved a technology rewrite—a one-time hit to provide an industrial-strength platform that could grow well into the future.

Flex 3 focused on developer tooling and maturing the framework to support larger scale applications. On the IDE side they added a profiler for measuring memory and cpu usage within an application, refactoring abilities, and an initial attempt at design to developer workflow. On the framework side they added charting enhancements, persistent framework caching, and a power grid component called the Advanced DataGrid. There was also the introduction of Adobe AIR for allowing your Flex applications to run as desktop applications.

Now in its fourth version, the focus of Flex 4 aims to improve developer productivity by helping you save time building your typical CRUD application (create, read, update, delete), a major overhaul on design to developer workflow, and data centric application developer.

The design to developer workflow, on any technology platform, has always proven to be a challenging one because the point at which a designer hands over the graphics, the developer then starts chopping it up into pieces to make it functional. The problem is that if the design changes, it results in the developer having to redo a lot of work as well as interpret the interaction as all they have is static images.

Adobe took a step back, and redefined how the tools relate to each other. Instead of having an integration perspective where it's about getting one tool to integrate, the vision was updated to view it as a platform where Flash is at the core. So it's not about just Flex, it's the whole platform (see figure 1.9) that has evolved in unison.

So with that said, here are some of the new goodies in Flex 4:

- A new class of visual components known as Spark components which replace the previous component set known as Halo components. Spark components support Flex 4's new skinning abilities.
- Skinning is completely redone, allowing designers to have fine grain control over the look and feel of all visual aspects. In Flex 3 you had to created programmatic skins if you really wanted to get serious, which of course no one did as it's too time consuming.
- A new graphics format called FXG that Flex, Flash, and Creative Suite will support. It's based on XML so you can easily define/create graphics in code.
- Data centric development wizards and capabilities that let you quickly point at any backend service (ColdFusion, PHP, Web Service, etc...) and it'll figure out what all of the available functions are, what the parameters are, and what kind of information comes back. And then you can link it to data centric components that can display and interact with the data.
- Along with data centric development are other productivity boosters like a faster compiler, getter/setter function generators, event coder generators, integrated Flex Unit support, tooltip documentation (mouse over some code and a hover window shows the documentation about that function/tag), and a network monitor for watching communications over the wire.
- Layout and visual states have been overhauled to be easier to work with. Along those lines, Flash Player 10 introduces a new text layout engine which you can leverage in Flex (making it easier to layout text).

These are the big at-a-glance features that are new Flex 4, but there are a number of additional enhancements that we'll get into as the chapters progress.

## 1.6 Summary

You've seen that historically, usability has been sacrificed for the sake of rapid deployment and a semi-neutral platform, but it came at the cost of the user experience. Amazingly, users (including ourselves) became accustomed to it. When it came to web applications, we assumed that being limited was normal.

It was just a matter of time in our information-crazed lives until the cost of this sacrifice began to outweigh the benefits. Users were suffering, and businesses were paying the price in productivity and efficiency.

RIAs demonstrate that we can achieve the best of both worlds. On the development side, a centralized deployment model on a platform-neutral environment means enhancements can be made quickly. On the user side, we can provide a desktop-like experience, allowing for fluid and engaging experiences from any device that can access the application.

Flex entered this RIA scene, leveraging its ubiquitous cross-platform Flash Player by creating a programmatic way to make a Flash application. This programmatic approach uses Flex's core languages: the XML templating language (MXML) and its scripting language (ActionScript).

Now in its third major version, Flex maintains a lead in the RIA space, although Microsoft is working hard to catch up. But what Microsoft doesn't have is Flex's ecosystem—a large technology environment that encompasses the end-to-end workflow of software development. From designer to developer to deployment, Adobe offers the environments and tools for each phase of the game.

In the next chapter, we'll explain how to set up your development environment, become familiar with the language reference, and begin to learn how to make Flex applications.