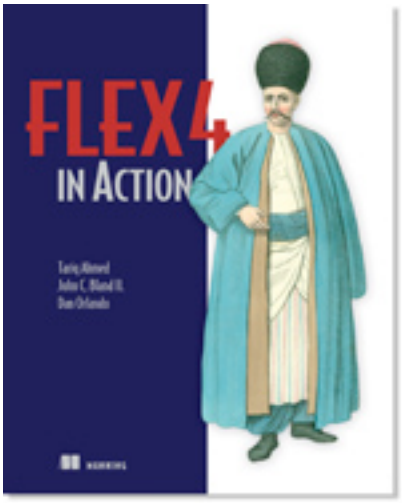
MANNING PUBLICATIONS

Green Paper from



Flex 4 in Action

EARLY ACCESS EDITION

Tariq Ahmed, Dan Orlando, and John C. Bland II

MEAP Release: June 2009
Softbound print: May 2010 (est.) | 600 pages
ISBN: 9781935182429

This green paper is taken from the forthcoming book Flex 4 in Action from Manning Publications.

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.

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, 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 A), 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.

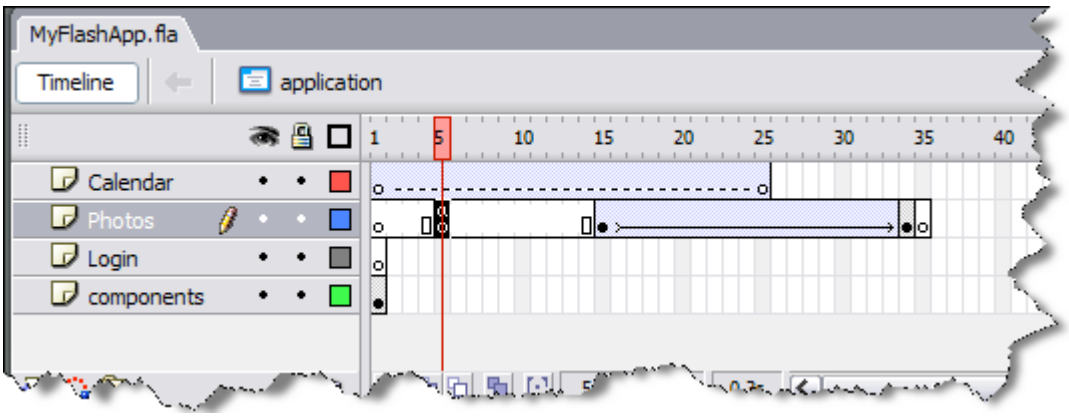


Figure A: Flash has always been capable of making RIAs, but can you imagine coding based on time?

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.

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.

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 B shows where Flex sits in relation to all of Adobe's other technologies.

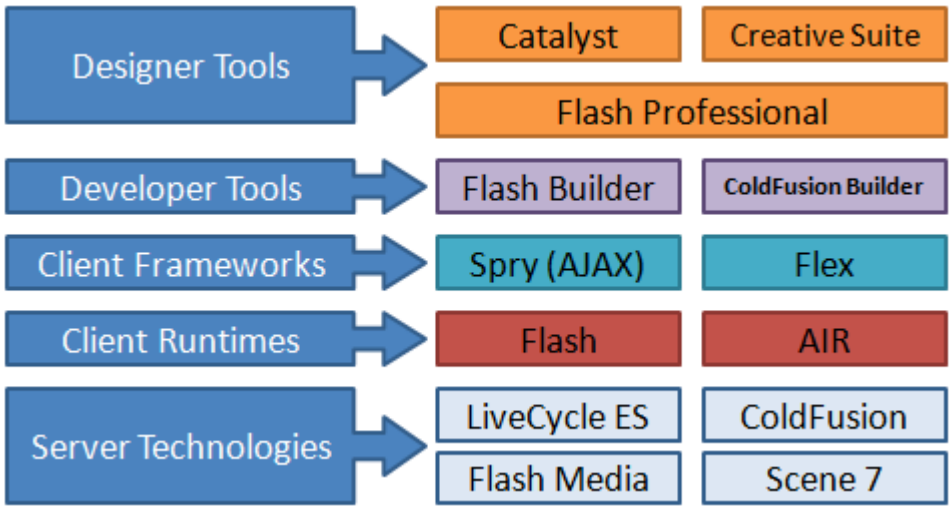


Figure B: Flex is part of a big technology stack.

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*.

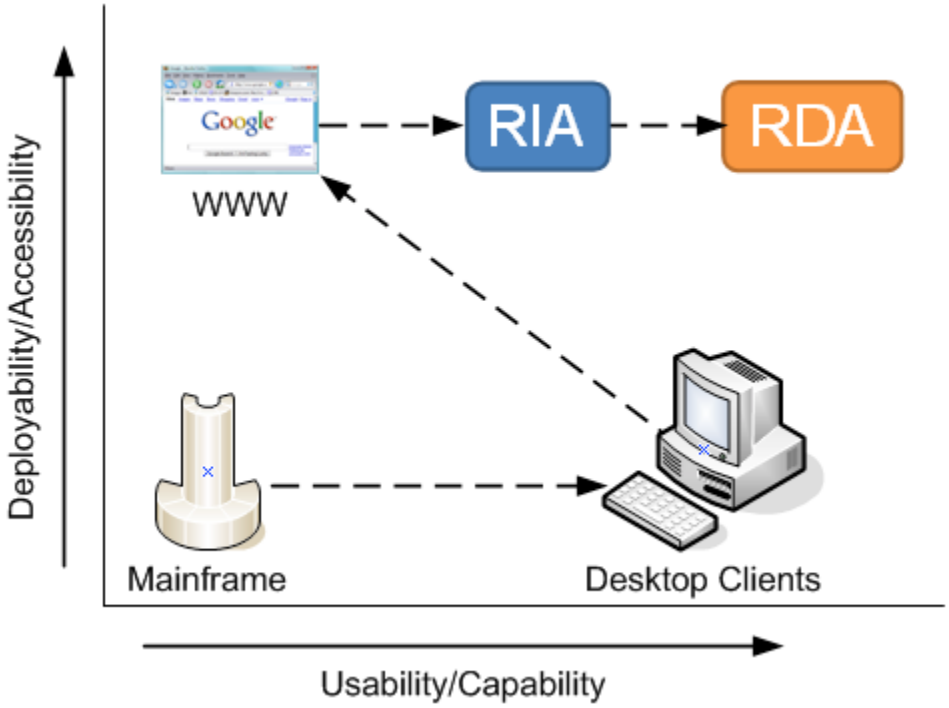


Figure C: RDAs go that extra mile by achieving the desktop experience that RIAs are not able to reach. RDAs exist outside of the browser, and like a desktop application have access to the operating system's clipboard, and local file system.

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 C), 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

Mentioned in the Flex eco-system is something called Adobe LiveCycle ES (Enterprise Suite). It's an engine similar to a workflow and business rules engine that allows companies to automate processes. The most publicly demonstrated use of this technology is automating the claims process for an insurance company.

Another popular use of LiveCycle is automating various government functions, because you can create triggers that execute sub-processes, ensure rules validation, and other such functions without having to write the code to do it.

However LiveCycle ES is aimed at big enterprises, and as a result comes with a very heavy price tag. Fortunately there's its little brother BlazeDS, which is a trimmed-down version of LiveCycle. 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. This is achieved through a technique known as long polling, unlike its LiveCycle big brother, which supports the more advanced real time messaging protocol (RTMP).

TIP

Messaging based applications are more network efficient when frequent updates are needed on many different items. For example when making a chat application, an AJAX based approach would periodically query for new information every few seconds. Using messaging, the chat client gets information pushed to it as it becomes available and thus not making unnecessary hits to the backend.

- Recordset paging with a database. (It can stream 50 database records at a time, or do the last/next 10 records from a query.)
- 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.

BlazeDS has been a major factor in the adoption of Flex in the Java community, because it allows Java developers to leverage their existing backend Java efforts by using BlazeDS as the bridge between Flex and the Java object.

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 D).

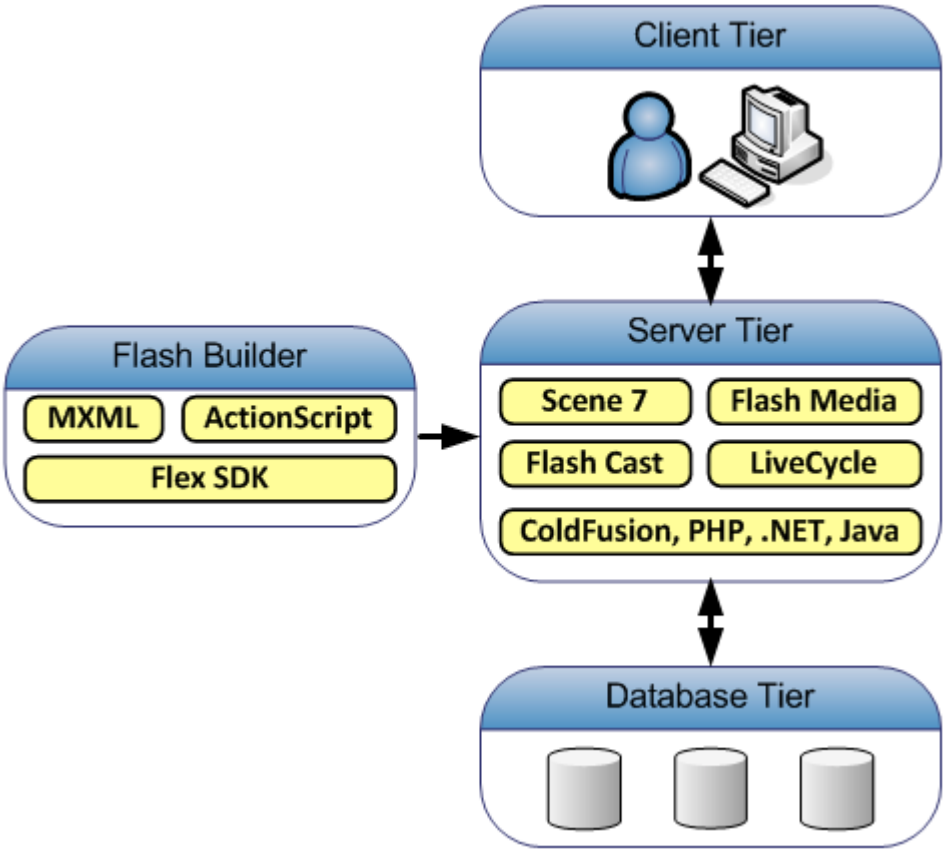


Figure D: Use Flex Builder to compile your application, and then deploy it to a server.

Ok, I can tell you're itching to see some actual code, so check out listing A which has a simple example of how these two languages are related.

Listing A: 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

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).

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 E).

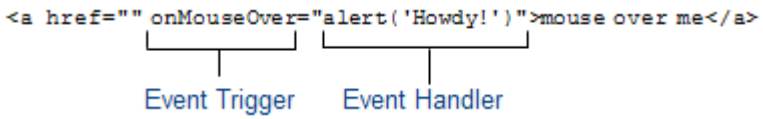


Figure E: Flex, like JavaScript, uses events that consist of triggers and handlers.

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.

Limitations

Flex it not perfect and has limitations that are true for any browser plug-in based engine. But to be objective it's worth being aware of these.

- Pop-up windows can exist only within the dimension of the player instance. Meaning that a pop-up can only move around within the area of the main application, vs. being a true pop-up that can move anywhere on the desktop. To break free of this limitation you'd have to convert your application to an AIR application.
- Because the Flash Player is a software install, many companies for support purposes standardize what's on the desktop. So if your application relies on a minimum version of the Flash Player (e.g. a Flex 4 application won't run on the antiquated Flash Player 8), and the user is a corporate user that doesn't have permission to upgrade software on their desktop, there's a dependency on that organizations I.T department to push a site wide update to all users.
- Since your loading an entire application, memory usage is something to watch for. In the book we'll learn about things like Runtime Shared Libraries that can help you manage that better, but this is an area where typically a web developer rarely cares about because the user is usually only working with a page at a time.
- There is no access to local drives or clipboard unless you go to AIR. This would be true for web applications as well, but this is a distinct advantage of desktop applications.

So with that said, the rich fluid interactive and fast experience tends to enable users with higher productivity, and thus the pros outweigh the cons by a wide margin. Plus, with each version of Flex comes new capabilities, let's see what's new in Flex 4.

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).