



METAOPTION

An Article on Adobe Flex
[By: Flex Team METAOPTION]

A Quick learning on Adobe Flex

There are several different technologies one can use to build RIAs (Rich Internet Applications). Rich Internet Applications are fast becoming an important differentiator in the area of web. RIAs combine the reach of the internet with a rich and compelling user interface that provides greater level of success and high level of satisfaction and success in a user's web based interactions, but the end user needs to be considered carefully.

For instance, one can use Ajax (Asynchronous JavaScript and XML) but this approach can introduce issues with client-side compatibility. You could use Applets, but that's not a popular choice at the moment too many web surfers disable the Java VM in their browsers.

Before going any further let's know about the Flex and where they fit into the client landscape.

Flex Basics

With Web users expecting richer and more complex interfaces, Rich Internet Applications (RIAs) are seeing a huge increase in popularity. Adobe Flex is the tool of choice for many web developers when it comes to building RIAs.

Adobe Flex is a collection of technologies released by [Adobe Systems](#) for the development and deployment of cross platform [rich Internet applications](#) based on the proprietary [Adobe Flash](#) platform.

Flex used to be one of those technologies that was only used by large corporate organizations -- the first release of Flex was priced at around US\$15,000 per CPU (a tad expensive for most developers) Since then, Flex has been released as open source software. Talk about a complete turnaround! Flex 3.0 has been such a success that Flex 4.0 has been announced for release in 2009.

As a developer, getting in early and learning all we can now is a good idea - - standing out in the Flex community will soon become extremely difficult. Developers are picking up Flex with great speed. One of the reasons for this is that Flex programming is relatively easy to learn if you're already familiar with XHTML. Of course, there's always a transition period when we're getting used to a new environment, but learning Flex won't take long!

Like any other tool FLEX also has a drawback, which is that Flex applications can be developed to run on a user's desktop thanks to the wonders of the Adobe AIR (Adobe Integrated Runtime) platform. Now we can develop and launch an RIA that runs in a web browser and as a desktop application. Deploying an application to a user's desktop with Adobe AIR is easy -- all that users need to do is click a link in the web browser to install the AIR runtime and our RIA on their machine. Now that's quick deployment!

Why Use Flex?

If you're considering building a RIA, you have a few choices of technology, including Flex, Ajax, and Microsoft Silverlight. If we look at these options objectively, the development effort required for each (and the resulting user experience) is roughly the same. One of the benefits of Flex is its reach -- the Adobe Flash Player, upon which Flex applications run, is already installed on about a billion computers worldwide!

Of course, Ajax also uses technologies that are installed on almost every computer in the world -- JavaScript, XHTML, and CSS. One of the downfalls of Ajax, however, is that cross-browser compatibility can be difficult to achieve. What might work in one browser (for example, Firefox) might not work in another (such as Internet Explorer), so the debugging process has the potential to become difficult and long-winded.

Microsoft Silverlight, on the other hand, is similar to Flex in that it runs on a browser plugin. Silverlight, however, has yet to reach the installed userbase of the Flash player.

If you're just starting out with building RIAs like me, you must agree with me is that it is the best and easiest to work with. However you should definitely try all of them to see which one you like best and find easiest to work with. Each of Flex, Silverlight and Ajax has its advantages and disadvantages. In my opinion, though, Flex is definitely the best RIA development technology available.

Overview of the Flex Framework

A lot of people steer clear of the Flex framework because they think it's complicated. But generally speaking, a framework is just a set of reusable classes that can work together to provide a base for an application.

In Flex framework, we have a stack of logic -- the controller logic -- that has been made available for communicating with a database, handling security, writing to the file system, and so on. There are also the user interface elements -- buttons, canvases, dropdown lists, and so on. All of these also form the foundation of our Flex application.

Getting Flex Up and Running

The Flex Builder 3 installer comes with everything we need built in. It automatically installs the Flex SDK, as well as the Eclipse-based IDE and the AIR runtime (which is useful if we want to build a desktop application rather than a web app. Once we have Flex Builder 3 installed on our machine, launch it .

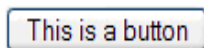
Flex is easy for web developers to learn because, at its core, it has a lot in common with (X)HTML, CSS, and JavaScript.

Suppose you wanted to create a simple web page with a form button. In XHTML you'd type the following:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://localhost/xhtml">
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Button Example</title>
</head>
<body>
  <form method="post" id="example"
action="http://www.example.com/">
  <input type="button" name="newButton" id="newButton"
value="This is a button" onclick="checkForm()" />
  </form>
</body>
</html>
```

When we view this markup in a web browser, we'll see a button displayed with the label "This is a button".

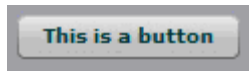


To display something similar in Flex we use a form of markup called MXML. Here's the MXML markup for our previous example:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="absolute">
  <mx:Button x="10" y="10" label="This is a button"
    id="newButton" click="checkForm()" />
</mx:Application>
```

To run this simple page as a Flex application, we need to compile our MXML code using the Flex SDK.

The result is shown below:



The first thing you'll notice is that MXML is an XML format. To indicate to the Flex compiler that I am defining in an application,

We use the `<mx:Application/>` element, in the same way we use the `<html></html>` tags to define a web page. We can then add other elements within the `<mx:Application/>` tag.

In the above example, we've added a `<mx:Button/>` tag to create a button, just as we'd use an `<input type="button" />` tag in a web page form.

As you can see, this is all very similar to constructing a traditional web page, and the framework provides us with everything we might use in XHTML (buttons, lists, etc.) and more. We have to learn the properties, methods and the names of the components in the framework, all of which are available from the Adobe Flex3 language reference.

Of course, the Flex framework doesn't just consist of user interface components; it also contains actions that our application can utilize. For instance, there's a component called HTTP Request, which our application can use to send and receive data from a server-side service (PHP, ASP.NET, etc.). When we run

our application, we don't actually see the HTTP Request, as it works in the background.

All these components have been bundled together by Adobe to form the Flex framework, so we don't have to create them from scratch.

Flex applications are made of roughly three parts:

- MXML that defines the interface,
- the ActionScript code that contains the application logic, and
- Resources such as images and audio.

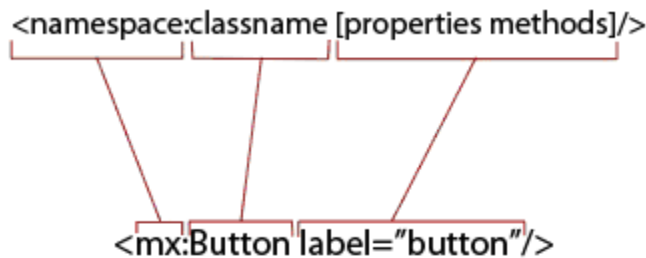
To draw an analogy with web technologies, the MXML and ActionScript would be HTML and JavaScript respectively.

Eventually, as we become more experienced with Flex, we'll want to create our own components that we'll use over and over.

Every component in MXML follows the same pattern:

1. We declare a namespace that tells Flex where to find a particular component.
2. We declare the component class we wish to use (e.g. Button) from that namespace.
3. We modify the available properties and methods using attributes, as illustrated below.

The makeup of a MXML component



Because they're defined in the class structure of the component, the properties and methods that can be used from within each component vary.

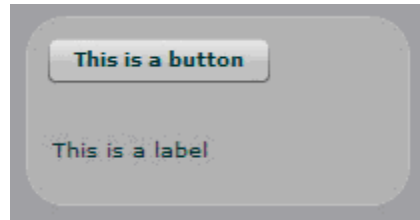
We style the visual components of our application with Flex Cascading Style Sheets. These styles can be added in the same location as the properties and methods of a component.

MXML components can have child elements (just like XML). For instance, a container element such as Canvas can have child elements like a Button or a Label.

The code below demonstrates this point:

```
<mx:Canvas x="53" y="64" width="192" height="94"
cornerRadius="20" borderStyle="solid"
    backgroundColor="#A9A9A9" id="mainCanvas">
  <mx:Button x="10" y="10" id="newButton"
    label="This is a button"/>
  <mx:Label x="10" y="57" id="newLabel"
    text="This is a label"/>
</mx:Canvas>
```

When we compile and run this code, we can see this web page:



By combining layout containers, such as Canvas, with other components, such as buttons and lists, it's possible to create great application designs in no time at all.

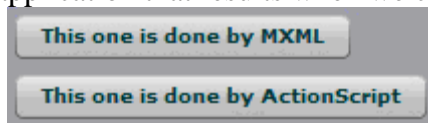
Taking Things Further: ActionScript 3.0

While MXML defines the structure of our Flex application, ActionScript 3.0 defines our application's behavior.

The following code creates the same component (a Button), first in MXML, and then in ActionScript 3.0:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute" creationComplete="init()">
    <mx:Button label="This one is done by MXML" x="10" y="10" />
    <mx:Script>
        <![CDATA[
            import mx.controls.Button;
            //Init Function is executed when the application boots
            private function init():void
            {
                //Create a new button
                var newButton:Button = new Button();
                //Modify Properties
                newButton.label = "This one is done by ActionScript";
                newButton.x = 10;
                newButton.y = 40;
                //Add the new button to the stage (Screen)
                this.addChild(newButton);
            }
        ]]>
    </mx:Script>
</mx:Application>
```

The application that results when we compile this file will look like this:



As we can see, both approaches for creating a button produce the same result -- but there's far less typing involved with MXML than with ActionScript 3.0. Designing an application with ActionScript 3.0 would be a nightmare. MXML was created to simplify the work for us.

But we still need to use ActionScript in our application, however; we'll need to define what happens when that button is clicked, for example.

Look at it in this way: we design our application with MXML, and we make it work with ActionScript 3.0. By using MXML and ActionScript, we are separating the structural code from the programming logic. This is an important philosophy to remember when building Flex applications -- especially when we are building complex components down the track.

Most of the programming done in Flex is event-driven, which means that functions are run when a component triggers an event (for example, when a mouse clicks a button on the page). The [Adobe Livedocs](#) site has some great examples of object-oriented ActionScript.

Summary

This article barely skimmed the surface of the Flex framework, although I tried to cover the basics of what the framework provides, and how MXML and ActionScript 3.0 work together. While this was a very gentle introduction to the concepts behind Flex, it should give you enough grounding in the concepts to go forth and experiment on your own.

Now that you've taken your first step in Flex development, the next step is to actually understand the components, play around with them to build your first application, and apply some ActionScript to really give it some life.

Hopefully this provides some context for getting started with Flex. If you'd like a more detailed walk-through of this subject, I recommend you try any tutorial on Flex for beginners.