FlexJS (legacy)

FlexJS™ is the name for a next-generation Flex SDK that has the goal of allowing applications developed in MXML and ActionScript to not only run in the Flash/AIR runtimes, but also to run natively in the browser without Flash, on mobile devices as a PhoneGap/Cordova application, and in embedded JS environments such as Chromium Embedded Framework. FlexJS has the potential to allow your MXML and ActionScript code to run in even more places than Flash currently does.

Overview

The original motivation for FlexJS was the leveraging of existing Flex code bases. While Flash is expected to run in browsers that run on computers with traditional keyboards for years to come, existing Flex customers are finding that they want their applications to run in places that Flash/AIR will not run because some of their end users are now allowed to use devices like tablets as their only computer. The cost of migrating an application is high, and the risks around quality control are significant, especially when migrating to a less-strict language like JavaScript.

However, as we've been working on FlexJS, it has become apparent that FlexJS should provide significant developer productivity gains for new projects as well. Flex was popular because it made it easy and efficient to create robust applications. Part of what made it easy and efficient was the Flash Player and AIR runtimes. They had consistent behavior across browsers and desktop platforms. But there were other things as well. Flex provided:

- Declarative Language (MXML)
- Semi-Structured Language (ActionScript)
- Runtime Verifier
- Choice of IDEs.

The combination of theses things made it possible to make fewer mistakes when writing code, thus saving time and improving productivity. Declarative Languages provide a schematic, or diagram of the pieces of the application, and are more terse than the equivalent in ActionScript or JavaScript. ActionScript understands classes and interfaces so it can do a better job of enforcing correct usage of APIs. The verifier catches errors at runtime. The IDEs can provide better code assistance because the language is more structured.

A presentation from ApacheCon 2015 that explores things deeper is http://events.linuxfoundation.org/sites/events/files/slides/FlexJS_ApacheCon_2015.pdf

Developing an application by using one or more frameworks really amounts to attaching framework components to each other via code you write. With JavaScript, you can literally attach anything to anything. You can assign a String to an variable that was expecting an int. JavaScript has some 'compilers' that try to catch things like that, but once you start using Object or other low-level base classes like EventDispatcher, you can easily cause the compiler to lose track of the actual API parameters involved and not find out until too late. Sure you can go around and annotate the JavaScript so the 'compilers' can try to help you, but often you may accept code in library form from somebody, and even load that code at runtime. The verifier tries to catch those kinds of problems.

One analogy is "assemble-it-yourself" furniture from IKEA and other places. If those kits only had nails or screws to attach the pieces together, and only provided instructions without diagrams or other illustrations, folks would make a lot of mistakes assembling the furniture. Instead, those furniture kits come with custom connectors that only fit into certain holes and in the instructions have diagrams. Semi-Structured languages like ActionScript allow you to establish custom connectors so that the components can only go together in certain ways. Declarative languages like MXML provide the diagrams to help you see that pieces that you are attaching. The verifier acts like an inspector, checking your work at runtime.

In theory, any JavaScript that can be encapsulated and presented with a more strongly-typed API is a candidate for use by FlexJS. FlexJS will in turn make sure you use that API correctly, catching your mistakes sooner. We have prototypes of applications that work with JQuery components. FlexJS will have its own default set of UI widgets, but folks should be able to use just about any JavaScript UI framework if it can be wrapped in an ActionScript API. FlexJS is less about the UI widgets and more about the productivity gains in using those widgets.

You can even write low-level JavaScript in ActionScript. FlexJS provides a special library that contains class definitions for the built-in objects available to JavaScript in the browsers. In other words, there are class definitions for HTMLElement, HTMLInputElement, and more. By using these types in ActionScript and cross-compiling to JavaScript you will catch more errors sooner than you would just writing JavaScript. In fact, all of the JavaScript code in the FlexJS UI libraries is written in ActionScript and cross-compiled.

The advantages of using higher-level languages should not be news: it is why Dart and TypeScript and other languages are often used to write JavaScript apps. What FlexJS also provides is the option to use a declarative language (MXML) as well. In theory, any JavaScript UI Library could add support for MXML in the future. But as of this writing, the FlexJS UI libraries are the only ones that support a declarative language and a semi-structure language from a single supplier.

For Flex Users

If you have already developed applications using Apache Flex, you are already familiar with the process used to develop applications using Apache FlexJS. That is, you have a mixture of MXML and ActionScript files with some ActionScript embedded in many (if not all) of your MXML files within <fx:Script> tags.

It is not possible (yet anyway) to directly port Apache Flex applications to FlexJS and make use of the ActionScript-to-JavaScript trans-compilation process (see "How it Works", below). Here is a basic outline of what you will need to do to make change your application from Flex to FlexJS:

First, your ActionScript "business logic" can probably remain the same. Those ActionScript files, or many functions within ActionScript files, that do not use any Apache Flex components, might just work. The ActionScript syntax for FlexJS is identical to ActionScript for Flex. Functions, variable declarations, loops, classes, and so forth are unchanged.

Second, your MXML files will need to shift from using Spark and MX components to using FlexJS components. While there are similarly available components, they are not completely identical. For example, FlexJS has the basic components such as Button, Label, and TextInput. But the FlexJS Button is a base class and you will want to use TextButton because it has a label or ImageButton if you want to use an image. TextInput provides just basic keyboard entry; if you want to restrict the input you will need a bead such as PasswordInputBead, to augment the TextInput control's abilities.

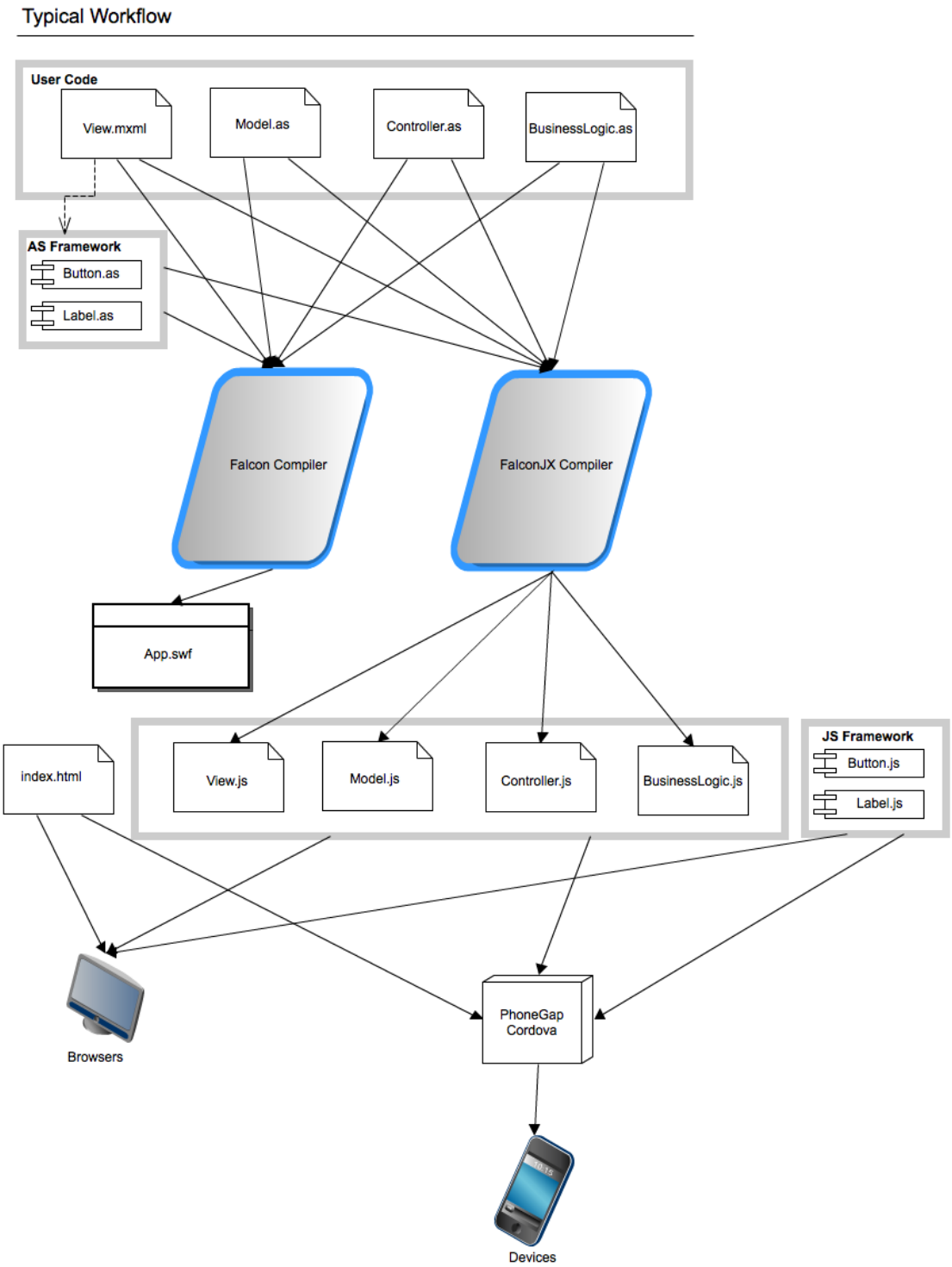
FlexJS comes with DataGrid, Panel, Slider, and many other controls as well a Group and Container to help organize and present them. The major difference is that the components only provide basic features and you use beads to add more functionality.

Here are links to popular IDEs and how to use them Apache FlexJS. You can also find these links in the Getting Started With FlexJS document.

Using FlexJS with Adobe Flash Builder
Using FlexJS with IntelliJ IDEA
Using FlexJS with FDT

How it Works

FlexJS application development (as opposed to framework component development) is based on the concept of parallel frameworks. The framework components are written in ActionScript with conditional compilation for SWF-specific or JS-specific code. An ActionScript and MXML compiler code-named "FalconJX" translates MXML and AS to JS and links in JS "classes" instead of AS classes to create the JS output.



Because ActionScript and JavaScript are based on ECMAScript, most code written in AS translates well to JavaScript. This is because, in most applications, the vast majority of code a Flex customer has written is not actually dependent on Flash. The underlying components like Button and DataGrid probably have dependencies on Flash, but there are equivalents in HTML and other JS frameworks. Thus to the extent a code base consists of assembling a bunch of UI controls into a view and integrating the view with ActionScript to business logic also written in ActionScript, it should be possible to have that code base leverage Flash-dependent controls in a SWF and leverage HTML-dependent controls without Flash in the browser.

Status

FlexJS 0.5.0 was released in November 2015. It is roughly of beta quality. The latest release is always available here. The FlexStore example that was a popular reference example in the Adobe Flex days has been ported to FlexJS. The SWF version is available here. Right-click and

choose View Source to see the MXML and ActionScript used in the example. The example uses view states, data binding, effects, and custom CSS just like a regular Flex-based Application. Then go here to view the app running in the browser without Flash. It is cross-compiled from the exact same source that created the Flash SWF version. If you right-click on this version, you will see that there is no entry about the Flash Player in the context menu which is proof that it does not use Flash, and you can choose View Source to see the minified JS output from the FlexJS compiler known as FalconJX. Also note that both the SWF and non-SWF versions are much smaller than any version you could create with the current Flex SDK and start up much faster.

You can try developing applications with FlexJS by following these instructions .

Schedule and Resources

Apache projects, including Apache Flex, are primarily staffed by volunteers working in their spare time. As such, a schedule of milestone dates is not practical. There is work going on to create Maven artifacts for FlexJS, and bugs are being fixed and more UI components and component sets are under development. How soon these things happen depends entirely on the number of folks who have time to contribute. There is still plenty of work to be done. Anyone with the time and energy is welcome to join in this effort.

We also could use resources to help build out an automated testing suite, documentation, and examples. A prototype of a Selenium-based automated testing engine for testing the JS side has been contributed but more tests need to be written and it would be interesting to find a way to leverage existing tests from the Flex SDK that are written in MXML and not Java.

Participation in any form is encouraged and welcomed. To follow the development of FlexJS, please subscribe to the Apache Flex mailing list dev@flex.apache.org. This is a high traffic list, but we try to mark all FlexJS discussion with the subject "[FlexJS (legacy)]".

Summary

FlexJS should provide the lowest-cost and quickest path to developing applications for the web and mobile devices, while also future-proofing existing Flex code bases, and providing similar developer-productivity and quality control advantages that the current Flex SDK provided.

For more details and information see FlexJS FAQ.