Contributing to the FlexJS Framework

Getting Started with FlexJS Framework Developer

If you would like to add components to the FlexJS framework, this page will help you figure out what to do and how to do it. Here's an overview:

A "component" in FlexJS is made up of a "strand" with "beads" that provide structure and functionality. The strand most often refers to the main component class and for visual components, will parent all of the elements that make up the component (such as Buttons, Text Input, and so forth). Input events (mouse, touch, keyboard) generated within the component area get handled by the component's controller bead(s). Any events generated by the component beads should be dispatched by the strand class.

Develop everything in ActionScript first. If you are making a component that uses already made components (such as a Label or a Button), there is a good chance you can have your component cross-compiled into its JavaScript equivalent; more about that later. The example shown here will assume you want to build upon existing components, but you will also see what it takes to make a lower-level component that will require hand-written JavaScript. We'll consider that an advanced topic.

To make life easier on yourself, use the same package names that the FlexJS framework uses. That way your code can move into the framework repository with little change.

Your "component" should live in: `org.apache.flex.html`. The beads that make up your component should live in: `org.apache.flex.html.beads` or one of its sub-directories, depending on its purpose.

FlexJS assembles a component from its beads through the use of the ValuesManager and style sheet. The ValuesManager class looks for your component in the defaults.css style sheet and picks up the class reference for your view and model beads. These beads will be instantiated for your component and added to its strand.

Component Example

> *For more details about creating components, see Creating Components. The code shown in the example is not intended to be a true component and a number of items have been left out; it is for illustrative purposes only.*

Consider a horizontal version of the NumericStepper where there is a Label showing a value and a Button to decrement the value to the left of the Label and a Button to increment the value to the right of the Label. This is a composite component and will be easy to cross-compile to JavaScript.

The component is called, "NumericAdjustment", or `org.apache.flex.html.NumericAdjustment`. This component needs the following beads:

NumericAdjustmentModel: This bead will be your components data model and hold the value for the component (the one on display between the increment and decrement buttons).

NumericAdjustmentView: This bead will create the three other elements (two Buttons and a Label) and set event listeners for those buttons which will change the value of the Label and dispatch a change event. The change event allows someone to use the component like this:

```
<basic:NumericAdjustment change="handleChange()" />
```

This is a simple component. Since the Buttons produce their own events, this component does not need its own mouse event controller beads. If you wanted to listen for the "+" and "-" keyboard events while this component has focus, then you would create a NumericAdjustmentKeyboardController bead.

While developing this bead you can use <fx:Style> block to declare the style for your components and this will wind up in the defaults.css file.

```
NumericAdjustment {
 iBeadView: ClassReference("org.apache.flex.html.beads.NumericAdjustmentView");
 iBeadModel: ClassReference("org.apache.flex.html.beads.models.NumericAdjustmentModel");
}
```

The NumericAdjustment component class, org.apache.flex.html.NumericAdjustment, has very little content of its own:

```
/**
 * The event dispatched whenever the value of the component changes.
 */
[Event("change")]

/**
 * The NumericAdjustment component displays a value with increment and decrement buttons.
 *
 * @langversion 3.0
 * @playerversion Flash 10.2
 * @playerversion AIR 2.6
 * @productversion FlexJS 0.0
 */
```

```
public class NumericAdjustment extends UIBase {
 /**
 * The current value of the component
 *
 * @langversion 3.0
 * @playerversion Flash 10.2
 * @playerversion AIR 2.6
 * @productversion FlexJS 0.0
 */
 public function get value():Number {
 return NumericAdjustmentModel(model).value;
 }
 public function set value(oldValue:Number):void {
 NumericAdjustmentModel(model).value = oldValue;
 }
}
```

The component class has [Event] metadata to indicate that the component can dispatch change events and declares a value property. Notice that the property is not stored in the component itself but in the component's model. The model property is part of the base class, UIBase.

> *The first time the model property is accessed, the framework code fetches the model's class definition from the style sheet and instantiates and instance of it.*

The NumericAdjustmentModel bead has a single property, value, which holds the value for the component:

```
/**
 * The NumericAdjustmentModel class holds the data model for the NumericAdjustment component.
 *
 * @langversion 3.0
 * @playerversion Flash 10.2
 * @playerversion AIR 2.6
 * @productversion FlexJS 0.0
 */
public class NumericAdjustmentModel implements IBeadModel {

private var _strand:IStrand;

/**
 *  @copy org.apache.flex.core.IBead#strand
 *
 *  @langversion 3.0
 *  @playerversion Flash 10.2
 *  @playerversion AIR 2.6
 *  @productversion FlexJS 0.0
 */
 public function set strand(value:IStrand):void {

_strand = value;
 }

 private var _value:Number;

/**
 * The current value of the component
 *
 * @langversion 3.0
 * @playerversion Flash 10.2
 * @playerversion AIR 2.6
 * @productversion FlexJS 0.0
 */
 public function get value():Number {

return _value;
 }
 public function set value(newValue:Number):void {
 if( _value != newValue) {
 _value = newValue;
 IEventDispatcher(_strand).dispatchEvent(new Event("change"));
 }
 }
}
```

Whenever the value in model changes, the model dispatches a change event using the strand (cast to IEventDispatcher).

The NumericAdjustmentView bead has a little more work to do. Most of the set up is done when the bead has been assigned its strand:

```
/**
 * The NumericAdjustmentView class is a bead that creates the elements of the NumericAdjustment component.
 *
 * @langversion 3.0
 * @playerversion Flash 10.2
 * @playerversion AIR 2.6
 * @productversion FlexJS 0.0
 */
 public class NumericAdjustmentView implements IBeadView {

private var _strand:IStrand;


/**
 *  @copy org.apache.flex.core.IBead#strand
 *
 *  @langversion 3.0
 *  @playerversion Flash 10.2
 *  @playerversion AIR 2.6
 *  @productversion FlexJS 0.0
 */
 public function set strand(value:IStrand):void {

_strand = value;

 _leftButton = new Button();
 _leftButton.text = "-";
 _strand.addElement(_leftButton);
 _leftButton.addEventListener("click",handleDecrement);

 _label = new UILabel();
 _label.text = "";
 _strand.addElement(_label);

 _rightButton = new Button();
 _rightButton.text = "+";
 _strand.addElement(_rightButton);
 _rightButton.addEventListener("click",handleIncrement);

 IEventDispatcher(_strand).addEventListener("change",handleChange);

 sizeAndPosition(null);
 }

 /**
 * @private
 */
 private function sizeAndPosition(event:Event):void
 {
 //TODO: put the components in order: incr button, label, decr button
 }

 /**
 * @private
 */
 private function handleChange(event:Event):void {

var model:NumericAdjustmentModel = _strand.getBeadByType(IBeadModel) as NumericAdjustmentModel;
 _label.text = String(model.value);
 }

 /**
 * @private
 */
 private function handleDecrement(event:Event):void {
 var model:NumericAdjustmentModel = _strand.getBeadByType(IBeadModel) as NumericAdjustmentModel;
 model.value -= 1;
 }

 /**
 * @private
 */
 private function handleIncrement(event:Event):void {
 var model:NumericAdjustmentModel = _strand.getBeadByType(IBeadModel) as NumericAdjustmentModel;
 model.value += 1;
 }
};
```

The view bead creates the component elements (buttons and label). It also listens for a change event that the model will dispatch when the value changes. Doing it this way, rather than from the code that changes the model (the handleDecrement and handleIncrement functions), allows the component to updated by anything that changes its model - which could be some other component.

> *The view bead should also listen for changes to the strand's width and height (you'll have to cast the strand to the UIBase class) and size and position the two buttons and label using the sizeAndPosition() function as the event listener.*

Putting it Together

Once you have the component's strand and beads ready, you can test them in your sample application by putting them into your initial UIView component.

After you have it working in ActionScript, you can quickly test it in JavaScript by cross-compiling it. The FalconJX compiler will build JavaScript versions of all of your component pieces. You can find them in the bin/js-debug (and bin/js-release) directory in the same directory/package path. Opening the bin/js-debug/index.html file in a browser should result in an HTML page that works exactly like the Flash SWF version.

> *Do not forget to include asdoc tags in your component as shown in the example code above.*

When your component is ready to become part of the framework, it can go into the FlexJSJX project if the component can be 100% cross-compiled into JavaScript. If you had to create any part of it in JavaScript, or modify any of the generated JavaScript files to get the component to work, then the ActionScript code must be placed into the FlexJSUI project.

If you find that your component cannot be 100% cross-compiled, you can use the generated JavaScript files as a starting point. Just copy the JavaScript files that make up your component into the frameworks/js/FlexJS/src directory and work with them there.

The FalconJX compiler will see that there are corresponding JavaScript files to your ActionScript sources already present in the `frameworks/js/FlexJS/src` directory and it will not cross-compile your ActionScript sources.

Once your ActionScript code has been put into the proper framework source directory, there are couple more steps before it can become fully functional:

1. Update the project's FlexJSJXClasses.as (or FlexJSUIClasses.as) file. You need to include and import statement and the class name so that the class winds up in the SWC.
2. Update the defaults.css file for the project and include the style declaration that ties the beads to the component strand.
3. Update the basic-manifest.xml file. This file maintains the basic namespace and allows you include your component in MXML files using something like `<basic:NumericAdjustment/>`.