Making An Interactive Image Gallery with { ActionScript } and XML

By Chad Jordan – November 15th, 2008

In this guide you will learn:

1. The main interface and setup of project elements using Adobe Flash Professional CS4
2. A fundamental approach to object-oriented programming with ActionScript 3.0
3. How to dynamically load data into a flash project using XML *(Extensible Markup Language)*
4. Implementation of tweens, easing effects, event listeners, arrays, and Booleans

Introduction

Making projects in Flash has remained a mixed bag for me not just because of the technical and artistic methodologies, but rather the requirements of installing both Java and Flash Player for websites that require excessive hardware resources.

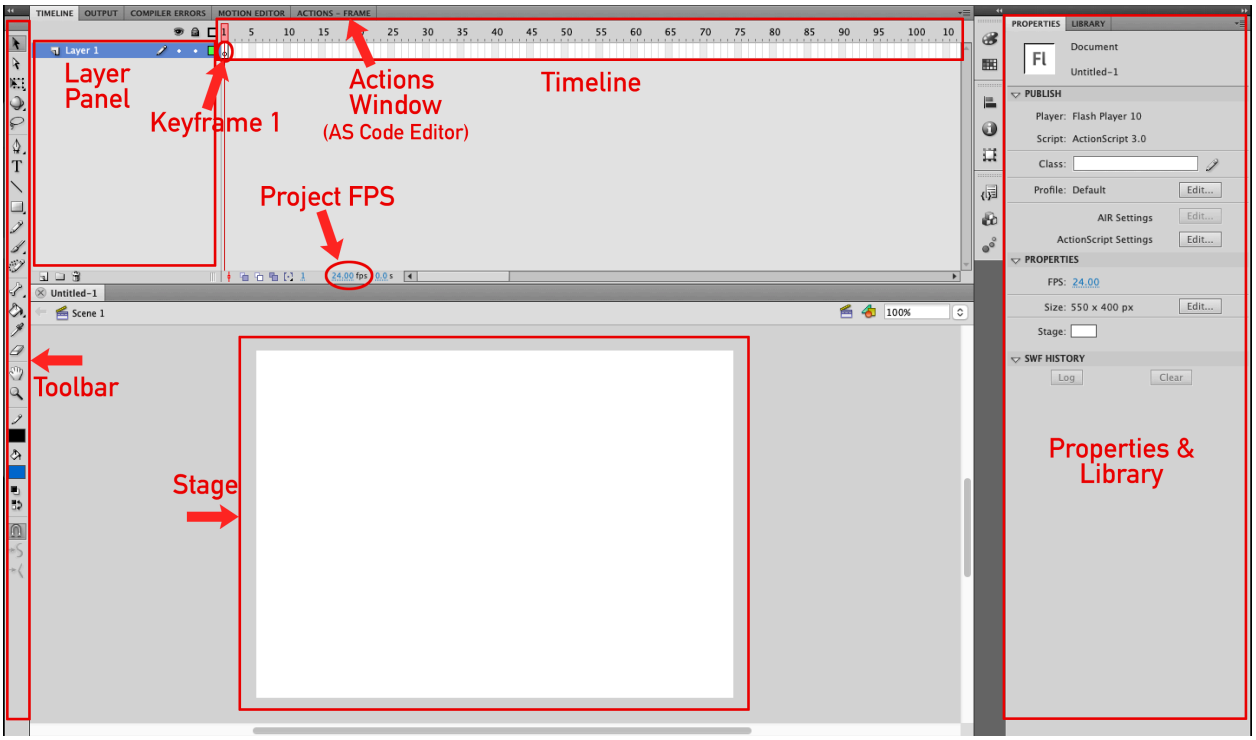
1) There can be issues regarding version control when installing Java on your computers that can leave some users confused on how to resolve the problems with specific versions of Java.
2) The requirement of Flash Player to be installed in order to view Flash websites or work with interactive applications is never good.

No matter the application/website being built, any requirement to power the technology should remain minimal for the sake of usability/reusability, ease of resources, bloatware, sustainability, and compatibility. However, there are vast numbers of clientele that love Flash, and ActionScript is the lightweight version of JavaScript. To its credit, ActionScript does allow a lot of really cool, unique, original content to be created for interactive games, animation for a web series, and *flashy* websites. I've known people who started their programming experience learning ActionScript before moving into heavier, thicker programming languages, and this seems to be very helpful for learning how to program. While I did not start with ActionScript, I can absolutely see and agree with this notion of learning how to program with variables, datatypes, conditional statements, loops, arrays, Booleans, and utilizing external data. Even though I'll be giving some brief insight into prepping some of these interactive elements, holistically this guide is going to focus on the implementation of ActionScript 3.0 programming to build an interactive web application. For any developers or people wanting to learn the fundamentals of ActionScript programming, this guide should be very straightforward for you. Like my previous guides, I'll be displaying and explaining the code as I implement it for my interactive image gallery. Anyone who has written code in Flash understands that the code editor being used to write ActionScript is all built into the Flash Professional software. While I will be building an application that is fairly straightforward in terms of object-oriented programming, this is by no means a beginner's guide. Therefore, readers should already have some exposure to programming languages and feel confident using Flash. **The primary purpose of this guide is the code** but I will be providing my code examples and explanations of said code along the way, one step at a time, just as I've done in previous guides. I am by no means a professional Flash developer, but I've written numerous applications in the past using object-oriented and procedural programming paradigms. Why does this matter? If you read my previous guides, hopefully you learned that **systematic problem-solving**, applying logic, and then learning the syntax of the language that you're wanting to write in can help you adapt to

other programming technologies for creating your content. When writing in AS3 *(ActionScript 3.0)*, the code is still dependent on using the Flash software which is obviously a drawback. This means we need to understand how to tie in the various elements that we'll be writing the functionality for in the code. These reference points created in the software, are required in order to have our code interact with the created elements. With all of this being said, let's begin creating in Flash!

Observing the Interface

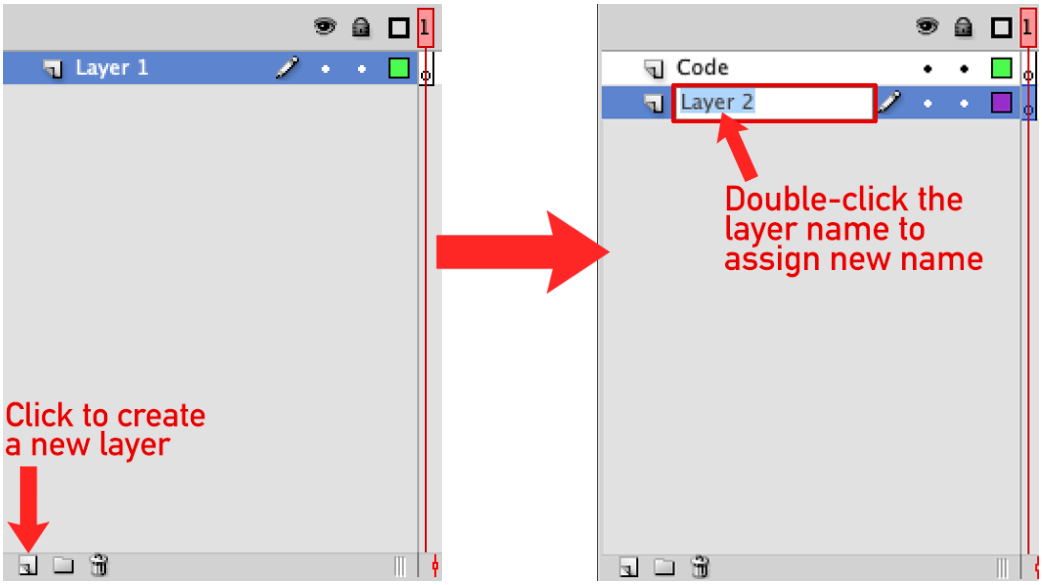
The Flash software is a multimedia platform designed to manipulate and animate vector graphics for interactive web applications. If you've used Flash, you know this is done through the primary Adobe software. Your new, *Untitled* project will be blank like this. Below are some of the primary areas to become familiar with when using the software. The **toolbar** has very similar tools and iconographies to Photoshop, and Illustrator, so if you're proficient in either of those, you'll feel right at home with the toolbar in Flash. The **layer panel** is also very similar to the layer panel in Photoshop, except this time the layers are tied to a timeline for the various animations that you want to create. If you're familiar with how to animate in 2D or 3D software, you know that setting this movement is referred to as *keyframing*. The first keyframe is at the beginning of the timeline. The **Actions Window/frame** is the code editor where we write all of the code for ActionScript. For this guide, I'll be spending the vast majority of my time in there. The **project FPS** setting is just the location where you change the pace of your frames per second for the movement your project. I like my project to move smoother and less choppy, so in this guide I'll be adjusting my frames per second to 30.00.



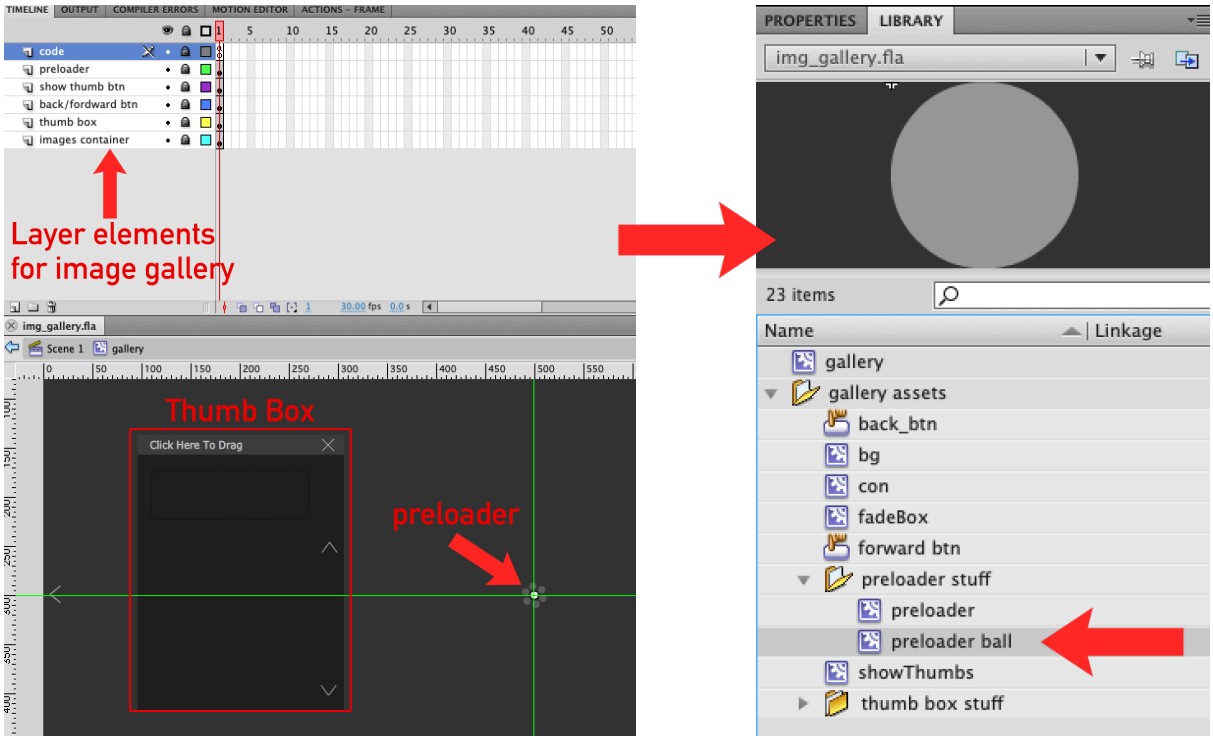
From a designer's perspective, think of the **stage** as a blank canvas where all of your content is displayed. You can click on the various content elements that you create in the stage, which will then auto-select the active layer in the layer panel. The **properties and library** pane is a place to make specific alterations to various elements within the project.

Prepping the Elements

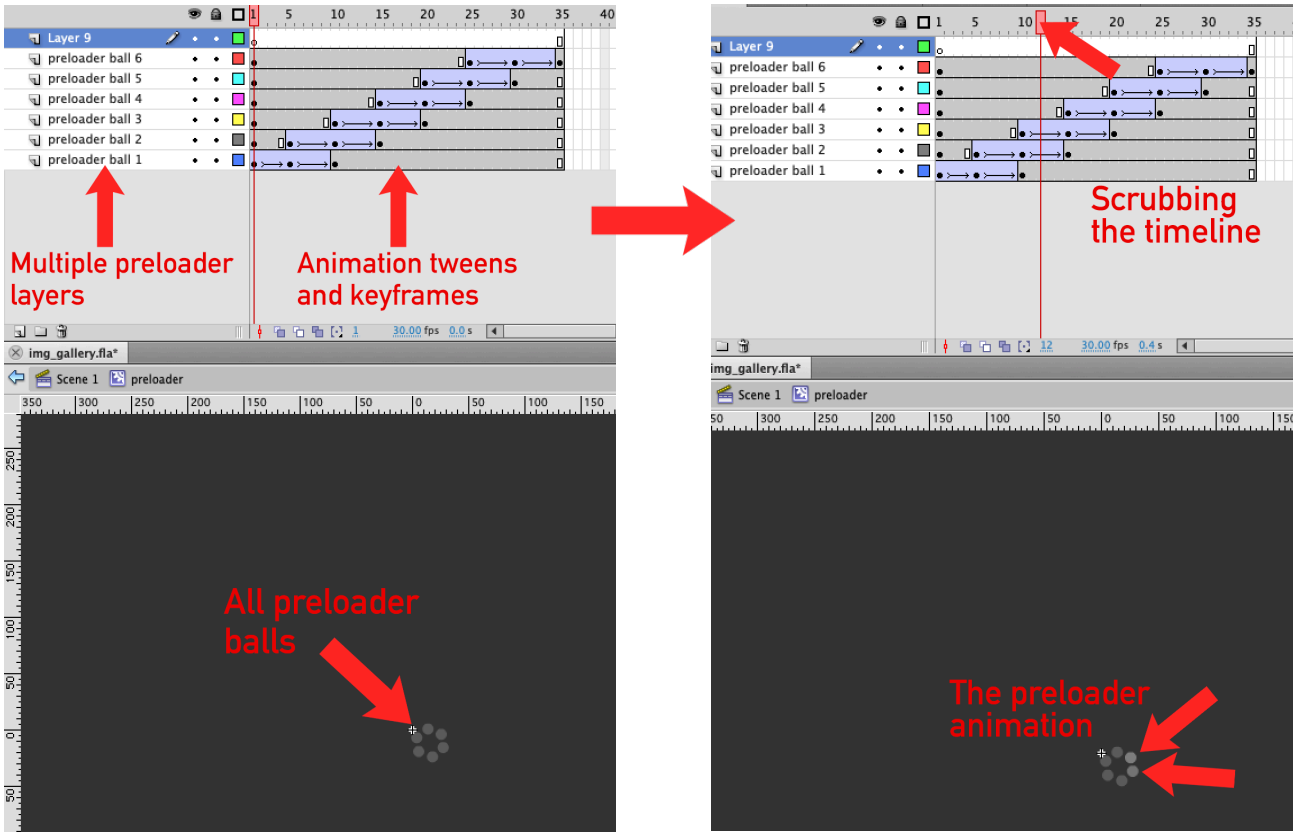
When creating new elements in the layers panel, the principles of achieving this is much like other design software like Photoshop. Every aspect of your created content that you're planning on having some form of functionality, needs to have another layer, and masked *sub-layers* within that. You can also *right-click* any layer and go into **properties** to name, and assign layer elements such as a movie-clip, a mask, a button, a container, etc.



This screenshot on the left displays the extra layers that I've created for the elements of my image gallery. The image on the right displays the first preloader ball that I created as a vector graphic. This is done with ease on the **stage** using the **toolbar** just like you would using Photoshop or Illustrator, and just as you create multiple layers with that software, we do the same thing in Flash for each layer. A standard preloader will have six different layers.



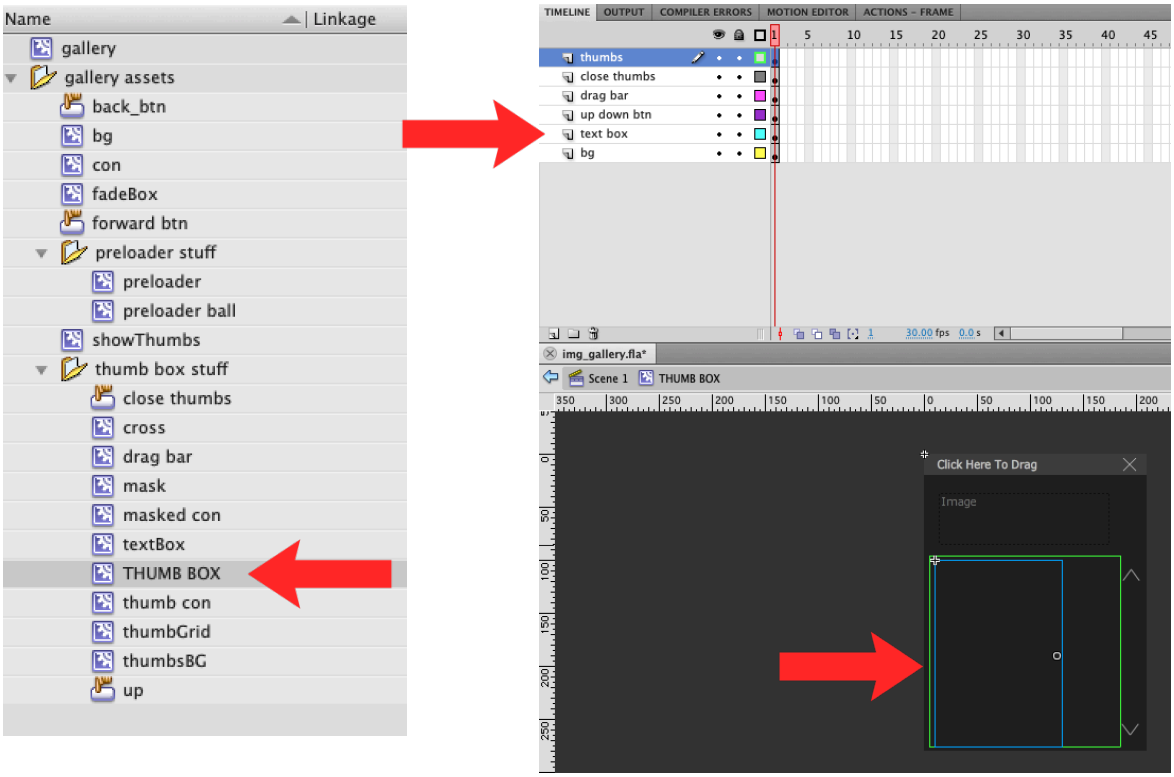
As you can see from the image on the left, I have six individual layers that will make up my preloader. As mentioned earlier, the layers that you create in Flash are used along the timeline with different types of elements attached to them. For the sake of animation, we can set these up as **tweens**. They're called tweens because in the world of animation tweens are short for *'in-between'* and this is known as the process of generating images that go between keyframes. As you hopefully already know from the concepts of animation, keyframes are the images at the beginning and end of a smooth transition. The image on the right displays the animation effects as I begin to scrub through the timeline, passing across the animation tweens.



You can see from the image on the left we are still a solid color with the preloader because we are still on frame 1 and have not initiated the start of our animation through the timeline yet. Even though it's rather small, we see from the image on the right displaying our change in the preloader balls the further I scrub across the timeline. The ActionScript code will repeat this animation process over and over until all of my images have loaded into the gallery into the **thumb box** and within the main viewport container.

This process with the preloader is only one small aspect of my program. The thumb box is a very good example of how to understand the XML portion of this program. Within Flash, we begin by creating a simple image container that will house all of the thumbnails for the image gallery.

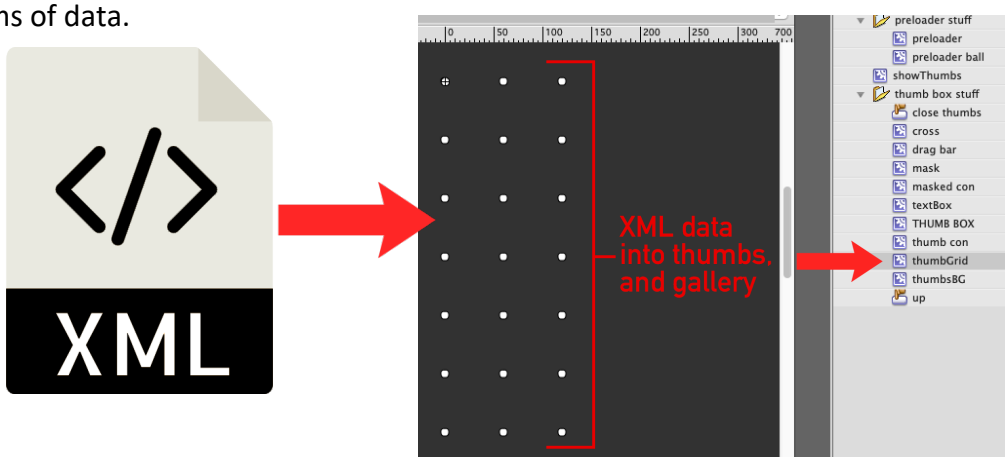
You can see from these screenshots that I've created multiple layers for the thumb box, and nearly all aspects of my project have been created as **movie-clips**. When you click on the corresponding component *(in this case movie-clip)* in the **properties and library** panel you will then see all of your layers populate in the layer panel for the component you click.



Doing this will also select *(highlight)* the corresponding element on the stage. These green and blue bounding boxes display the proportions of the elements within my thumb box. From the above example, you can see how I created my thumb box using basic design tools from the toolbar. This is how we create all of our designs for the stage.

Setting Up the XML Document

Now It's time to prepare the XML document as the external reference point for dynamically loading content into Flash. XML is a markup language based on SGML *(Standard Generalized Markup Language)* and XML's primary function is to create formats for data that are used to encode information for documentation, transactions, database records, and multiple other forms of data.



Starting off with any blank document such as *Notepad*, *TextEdit*, *Wordpad*, etc. I've named my XML document 'imageURLs.xml' and it will contain the filenames for the data that is loaded into the project. For this application, the XML code in my file is displayed as the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<gallery>
	<image imagePath="images/pic01.jpg" thumbPath="thumbs/thumb01.png" info="Image 1"> </image>
	<image imagePath="images/pic02.jpg" thumbPath="thumbs/thumb02.png" info="Image 2"> </image>
	<image imagePath="images/pic03.jpg" thumbPath="thumbs/thumb03.png" info="Image 3"> </image>
	<image imagePath="images/pic04.jpg" thumbPath="thumbs/thumb04.png" info="Image 4"> </image>
	<image imagePath="images/pic05.jpg" thumbPath="thumbs/thumb05.png" info="Image 5"> </image>
	<image imagePath="images/pic06.jpg" thumbPath="thumbs/thumb06.png" info="Image 6"> </image>
	<image imagePath="images/pic07.jpg" thumbPath="thumbs/thumb07.png" info="Image 7"> </image>
	<image imagePath="images/pic08.jpg" thumbPath="thumbs/thumb08.png" info="Image 8"> </image>
	<image imagePath="images/pic09.jpg" thumbPath="thumbs/thumb09.png" info="Image 9"> </image>
	<image imagePath="images/pic10.jpg" thumbPath="thumbs/thumb10.png" info="Image 10"> </image>
	<image imagePath="images/pic11.jpg" thumbPath="thumbs/thumb11.png" info="Image 11"> </image>
	<image imagePath="images/pic12.jpg" thumbPath="thumbs/thumb12.png" info="Image 12"> </image>
	<image imagePath="images/pic13.jpg" thumbPath="thumbs/thumb13.png" info="Image 13"> </image>
	<image imagePath="images/pic14.jpg" thumbPath="thumbs/thumb14.png" info="Image 14"> </image>
	<image imagePath="images/pic15.jpg" thumbPath="thumbs/thumb15.png" info="Image 15"> </image>
	<image imagePath="images/pic16.jpg" thumbPath="thumbs/thumb16.png" info="Image 16"> </image>
	<image imagePath="images/pic17.jpg" thumbPath="thumbs/thumb17.png" info="Image 17"> </image>
	<image imagePath="images/pic18.jpg" thumbPath="thumbs/thumb18.png" info="Image 18"> </image>
	<image imagePath="images/pic19.jpg" thumbPath="thumbs/thumb19.png" info="Image 19"> </image>
	<image imagePath="images/pic20.jpg" thumbPath="thumbs/thumb20.png" info="Image 20"> </image>
</gallery>
```

Depending on how many images you are wanting to use, is up to you, but for my application, I'm going to be loading 20 images into my file. The code makes a reference call using the image tag and references the current location of the image file and then assigns it to 'imagePath'. We also see the same reference being done for the small thumbnails that are loaded into the thumbGrid attribute in Flash. This will be done in the ActionScript code. The XML document will be saved in the root of the project directory with the Flash project files. With this file now in place, let's program with ActionScript!

Programming in ActionScript 3.0

As mentioned earlier in the introduction, this guide is primarily about writing ActionScript code, so from here I am leaving the rest of the design portion alone and jumping into the syntax side of things. However you decide to create your own work is entirely up to you. As the designer, you know how you want your content to work, but in order for everything to have functionality, the code is the most important aspect of the application. As mentioned on page 2, ActionScript is written in the code editor found in the **Actions** window of Flash. This is accessed just as you would expect by clicking **Window > Actions**.

Once in the Actions window, we have to select the proper layer that we wish to right code under. Per my earlier screenshots, you may have noticed that I created a layer called 'Code' and this is the active layer that we want to make sure is selected so we can begin writing our

code for the gallery. I begin by importing the proper animation libraries into ActionScript. Next, on **line 4**, I make variable declarations for an array to track which image is being displayed to the user, and currentImage is the active image that begins in the XML list. Beginning on **line 7**, the DropShadowFilter attribute is a filter for the floating thumb menu.
1st number = distance, 2nd number = angle, the hexadecimal number = color, next is the amount of alpha, the last 2 figures are the x & y blur. **Line 10** gives the main images a subtle glow and border for when we roll over the images with the mouse.

```
1  import fl.transitions.Tween;
2  import fl.transitions.easing.*;
3
4  var thumbArray:Array = new Array();
5  var currentImage:Number = 0;
6
7  var ds:DropShadowFilter = new DropShadowFilter(8,45,0x000000,.8,10,10);
8  var dsDrag:DropShadowFilter = new DropShadowFilter(10,45,0x000000,.8,10,10);
9
10 var imageGlow:GlowFilter = new GlowFilter(0x222222);
11
12 var imageFade:Tween;
13
14 var scrollDownTween:Tween;
15 var scrollUpTween:Tween;
16 var scrollClickNum:Number = 0;
17 var scrollStartNum:Number = 0;
18 var scrollAmount:Number = 0;
19
```

Beginning on **line 14**, these are the variable declarations for scrolling up and down along the thumbnails using the button controls on the thumb box.

In this next section starting on **line 20**, I give a function call to 'new Array' for using the scroller element to access the 'maskedCon' component that I declared in the *properties and library* panel of Flash. The data that is pulled from this, is then assigned to the *thumbsArray* variable. This array is used to load the thumbnails in the correct place of the 'thumbGrid' component. I create more variable declarations for referencing the XML file. These requests are stored in the variables and referenced in the functions that I write for storing and using the XML data.

```
20 var thumbsArray = new Array(scroller.maskedCon.con.thumbCon01,scroller.maskedCon.con.thumbCon02,
21                             scroller.maskedCon.con.thumbCon03,scroller.maskedCon.con.thumbCon04,
22                             scroller.maskedCon.con.thumbCon05,scroller.maskedCon.con.thumbCon06,
23                             scroller.maskedCon.con.thumbCon07,scroller.maskedCon.con.thumbCon08,
24                             scroller.maskedCon.con.thumbCon09,scroller.maskedCon.con.thumbCon10,
25                             scroller.maskedCon.con.thumbCon11,scroller.maskedCon.con.thumbCon12,
26                             scroller.maskedCon.con.thumbCon13,scroller.maskedCon.con.thumbCon14,
27                             scroller.maskedCon.con.thumbCon15,scroller.maskedCon.con.thumbCon16,
28                             scroller.maskedCon.con.thumbCon17,scroller.maskedCon.con.thumbCon18,
29                             scroller.maskedCon.con.thumbCon19,scroller.maskedCon.con.thumbCon20);
30
31 var j:Number = 0;
32 var imageLoader:Loader = new Loader();
33 var thumbsOn:Boolean = true;
34 var imagesXML:XML;
35 var XMLReq:URLRequest = new URLRequest("imageURLs.xml");
36 var XMLLoader:URLLoader = new URLLoader();
37 XMLLoader.load(XMLReq);
38 XMLLoader.addEventListener(Event.COMPLETE, XMLDone);
```

Beginning on **line 40**, the function XMLDone is designed as an event listener and allows us to dynamically allocate the data from the XML file, and provides the functionality required to load the data into the Flash project. Starting on **line 44**, firstImageURL places the first image on the stage, and then the for loop on **line 51** iterates sequentially through the load data and places the images in order according to the XML instructions. Starting on **Line 59**, we add event listeners to the thumbnails.

```
40  function XMLDone(event:Event):void {
41      imagesXML = new XML(XMLLoader.data);
42      scrollAmount = Math.ceil(imagesXML.image.length()/3)-3;
43
44      var firstImageURL:URLRequest = new URLRequest(imagesXML.image[0].@imagePath);
45      imageLoader.load(firstImageURL);
46      imagesCon.addChild(imageLoader);
47      imageLoader.filters = [imageGlow];
48      scroller.txtBoxBG.textBox.text = imagesXML.image[currentImage].@info;
49      imageLoader.contentLoaderInfo.addEventListener(Event.COMPLETE, imageDone);
50
51      for (var i:uint = 0; i<imagesXML.image.length(); i++) {
52          var thumbsReq:URLRequest = new URLRequest(imagesXML.image[i].@thumbPath);
53          var thumbLoader:Loader = new Loader();
54          thumbLoader.load(thumbsReq);
55          thumbsArray[i].addChild(thumbLoader);
56          thumbArray.push(thumbLoader);
57          thumbLoader.alpha = .7;
58
59          thumbLoader.addEventListener(MouseEvent.CLICK, showBigImage);
60          thumbLoader.addEventListener(MouseEvent.ROLL_OVER, thumbOver);
61          thumbLoader.addEventListener(MouseEvent.ROLL_OUT, thumbOut);
62          thumbLoader.addEventListener(MouseEvent.MOUSE_DOWN, thumbDown);
63          thumbLoader.addEventListener(MouseEvent.MOUSE_UP, thumbUp);
64      }
```

The next section of code involves writing a series of mouse event functions for the thumbs.

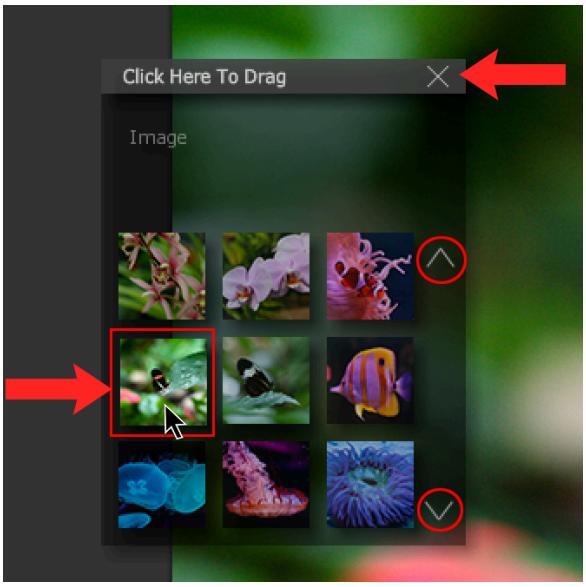
```
66      function thumbOver(event:MouseEvent):void {
67          event.target.x +=1;
68          event.target.y +=1;
69          event.target.alpha = 1;
70      }
71
72      function thumbOut(event:MouseEvent):void {
73          event.target.x -=1;
74          event.target.y -=1;
75          event.target.alpha = .7;
76      }
77
78      function thumbDown(event:MouseEvent):void {
79          event.target.y+=1;
80      }
81
82      function thumbUp(event:MouseEvent):void {
83          event.target.y-=1;
84          //do not edit
85          currentImage = thumbArray.indexOf(event.target);
86          scroller.txtBoxBG.textBox.text = imagesXML.image[currentImage].@info;
87      }
88
89      function showBigImage(event:MouseEvent):void {
90          currentImage = thumbArray.indexOf(event.target);
91          var imageURL:URLRequest = new URLRequest(imagesXML.image[currentImage].@imagePath);
92          imageLoader.load(imageURL);
93          preloader.visible = true;
94          imageLoader.addEventListener(Event.COMPLETE, imageDone);
95      }
```

This section of the program controls the forward and backward movement of selecting the images. Even though the thumb box provides faster navigation through the images, the user should still have the option to close out of the thumb box if they just want to scroll from left to right, and vice versa. **Line 120** places the images on the stage. The event listeners wait for a mouse click and the function is executed when the direction of the button is clicked on the left and the right sides of the image container.

```
 97      imagesForward_btn.addEventListener(MouseEvent.CLICK, imagesForward);
 98          function imagesForward(event:MouseEvent):void {
 99              if (currentImage == imagesXML.image.length() - 1) {
100                  currentImage =-1;
101              }
102              currentImage +=1
103              preloader.visible = true;
104              var imagesForwardURL:URLRequest = new URLRequest(imagesXML.image[currentImage].@imagePath);
105              imageLoader.load(imagesForwardURL);
106              imageLoader.addEventListener(Event.COMPLETE, imageDone);
107          }
108          imagesBack_btn.addEventListener(MouseEvent.CLICK, imagesBack);
109          function imagesBack(event:MouseEvent):void {
110              if (currentImage == 0) {
111                  currentImage = imagesXML.image.length();
112              }
113              currentImage -=1;
114              preloader.visible = true;
115              var imagesForwardURL:URLRequest = new URLRequest(imagesXML.image[currentImage].@imagePath);
116              imageLoader.load(imagesForwardURL);
117              imageLoader.addEventListener(Event.COMPLETE, imageDone);
118          }
119
120      function imageDone(event:Event):void {
121          imagesCon.addChild(imageLoader);
122          preloader.visible = false;
123          imageFade = new Tween(imageLoader,"alpha", None.easeNone, 0,1,1,true);
124          imageLoader.filters = [imageGlow];
125          imageLoader.x = -imageLoader.width/2;
126          imageLoader.y = -imageLoader.height/2;
127      }
128
129  }
```

When the thumb box is open, we want to have a way to scroll up and down through the images when pushing the arrow buttons. These functions allow those controls to work.

```
131  scroller.down.addEventListener(MouseEvent.CLICK, thumbsDown);
132  function thumbsDown(event:MouseEvent):void {
133      if (scrollClickNum <scrollAmount) {
134          scrollClickNum ++;
135          scrollStartNum = (scrollClickNum-1)*60;
136          scrollDownTween = new Tween (scroller.maskedCon.con, "y", Strong.easeOut,
137                                       -scrollStartNum, -scrollStartNum -60,1,true);
138      }
139  }
140
141  scroller.up.addEventListener(MouseEvent.CLICK, thumbsUp);
142  function thumbsUp(event:MouseEvent):void {
143      if (scrollClickNum >0) {
144      scrollClickNum --;
145      scrollStartNum = (scrollClickNum+1) *60;
146      scrollUpTween = new Tween(scroller.maskedCon.con, "y", Strong.easeOut,
147                                -scrollStartNum, - scrollStartNum +60, 1,true);
148      }
149  }
150
```



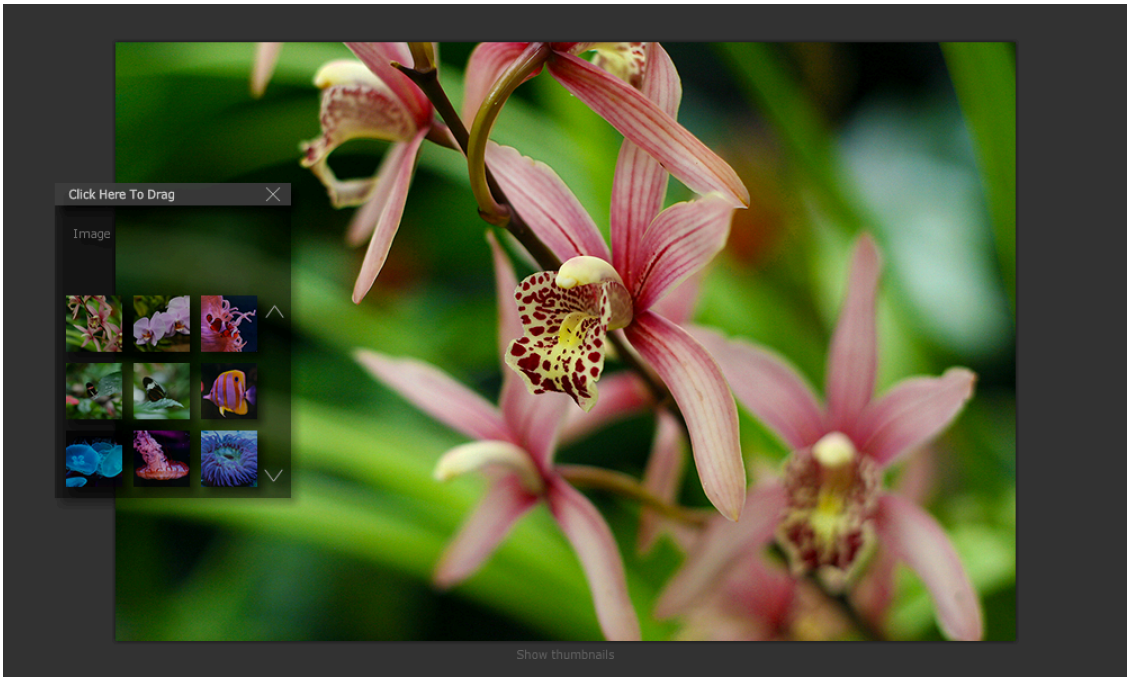
These results show the progress thus far of what the thumb box looks like with the images loaded into it, along with the control buttons. When the thumb box is open, we want the ability to drag it around anywhere within the program. In this next section of code on **line 151** the scroller element accesses the dragBar attribute which initiates the Event Listener and passes the MouseEvent parameter to the function. Starting on **line 167** we setup a function that gives the user the option to close the thumb box if they prefer to scroll through the images without the box on the screen. We have to give the user the ability to reopen the thumb box if they desire, so to do this we write a command on **line 176** to display the thumb box if it's not currently active. Following that we write a function on **line 178** for the *'showThumbs'* attribute. It uses event listeners to wait for user input on mouse events.

```
151 scroller.dragBar.addEventListener(MouseEvent.MOUSE_DOWN, dragThumbsStart);
152 function dragThumbsStart(event:MouseEvent):void {
153     scroller.startDrag();
154     scroller.filters =[dsDrag];
155     scroller.y -=1;
156     scroller.x -=1;
157 }
158
159 scroller.dragBar.addEventListener(MouseEvent.MOUSE_UP, dragThumbsStop);
160 function dragThumbsStop(event:MouseEvent):void {
161     scroller.stopDrag();
162     scroller.filters =[ds];
163     scroller.y +=1;
164     scroller.x +=1;
165 }
166
167 scroller.closeThumbs.addEventListener(MouseEvent.CLICK, closeThumbsBox);
168 function closeThumbsBox(event:MouseEvent):void {
169     scroller.visible = false;
170     imagesForward_btn.visible = true;
171     imagesBack_btn.visible = true;
172     showThumbsBTN.gotoAndStop(2);
173     thumbsOn = false;
174 }
175
176 showThumbsBTN.addEventListener(MouseEvent.CLICK, showThumbs);
177
178 function showThumbs(event:MouseEvent):void {
179     if (thumbsOn == false) {
180         scroller.visible = true;
181         imagesForward_btn.visible = false;
182     imagesBack_btn.visible = false;
183         thumbsOn = true;
184         showThumbsBTN.gotoAndStop(1);
185     }
186 }
```

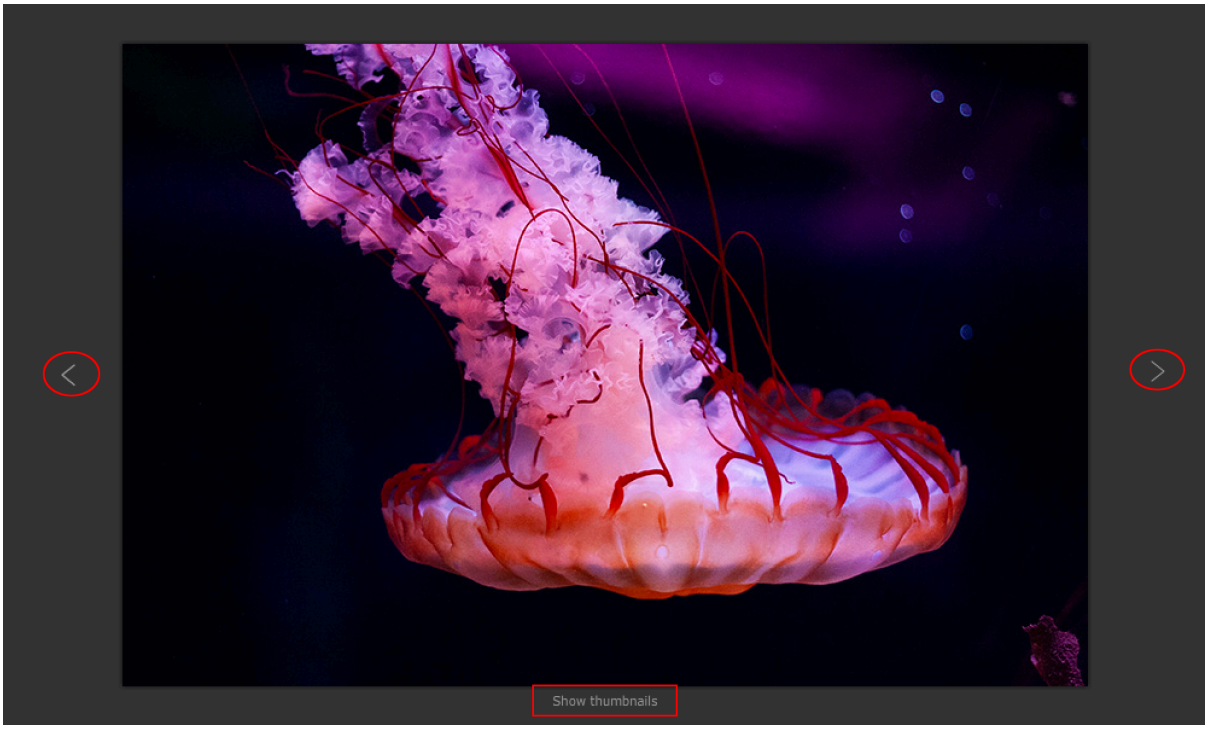
We have to give the user the ability to resize the window at runtime and to do this, I reconfigure the x & y positions of the stage based on how the window is resized by the user. This section of the code takes the default input, and renders call-back functions for translating new locations for all pieces of content inside of the program window.

```
188  stage.align = StageAlign.TOP_LEFT;
189  stage.scaleMode = StageScaleMode.NO_SCALE;
190
191  stage.addEventListener(Event.RESIZE, rearrangeStage);
192
193  function rearrangeStage(event:Event):void {
194      showThumbsBTN.y = stage.stageHeight - 30;
195      showThumbsBTN.x = stage.stageWidth/2-(showThumbsBTN.width/2) - 3;
196      imagesCon.x = stage.stageWidth/2;
197      imagesCon.y = stage.stageHeight/2;
198      preloader.x = stage.stageWidth/2;
199      preloader.y = stage.stageHeight/2;
200      imagesBack_btn.x = 50;
201      imagesBack_btn.y = stage.stageHeight/2;
202      imagesForward_btn.x = stage.stageWidth -50;
203      imagesForward_btn.y = stage.stageHeight/2;
204  }
205
206  showThumbsBTN.y = stage.stageHeight - 30;
207      showThumbsBTN.x = stage.stageWidth/2-(showThumbsBTN.width/2) - 3;
208      imagesCon.x = stage.stageWidth/2;
209      imagesCon.y = stage.stageHeight/2;
210      preloader.x = stage.stageWidth/2;
211      preloader.y = stage.stageHeight/2;
212      imagesBack_btn.x = 50;
213      imagesBack_btn.y = stage.stageHeight/2;
214      imagesForward_btn.x = stage.stageWidth -50;
215      imagesForward_btn.y = stage.stageHeight/2;
216
217  imagesForward_btn.visible = false;
218  imagesBack_btn.visible = false;
219  showThumbsBTN.buttonMode = true;
220
221  scroller.filters =[ds];
```

This concludes the programming portion for the image gallery. I've chosen some photos from my own nature photography that I've shot in the past. Here is the result of the app at runtime:



Every photo was properly read into the program, I tested all aspects of the application's functionality, and everything runs exactly as it should! The compiled swf file from this application can be embedded and used on any Flash website. As discussed earlier, the user needs to have the option to close out of the thumb box, and by doing so, the following buttons appear and are also fully functional. Each photo makes a smooth transition fading in and out.



Conclusion

As you can see from this guide, interactive applications with ActionScript can be created with little effort compared to how much more would be required with JavaScript. However, we also have to utilize the Flash software to tie everything together. JavaScript also doesn't require the same resources that writing programs in ActionScript does, both on the software and hardware side of things. This is all the more reason why I personally believe ActionScript, while a good programming language to learn, is not good for reusability, stability, and compatibility. The methodology of how it's created is not likely to attract the majority of developers in the industry regardless of how much organizations and other companies love the technology. I definitely believe it's good to display diverse, interactive knowledge in one's portfolio of work. Diversity in technical knowledge proves strong adaptability which is very important in tech. I hope this guide has been helpful for you. All screenshots within Flash, the written code, and captured images of the application within this guide were created by me based on a program assignment from my Web Animation course in ActionScript. If you have any questions about this guide or any other general inquiries, you can email me at technologicguy@gmail.com