Flash Actionscript 3.0 Primer
Variables, Functions & Events

by Jerron Smith

There are three very important elements that you need to understand in order to be successful when working with Actionscript. Together **Variables**, **Function** and **Events** are what allow for interactivity in the Flash environment. Essentially: **Variables** hold information for later use, **Functions** actually do things, and **Listeners** allow functions to be triggered by specific events like a user clicking a button or the swf file loading.

A word about Migration from Actionscript 2.0 to Actionscript 3.0

While there are many major changes that have occurred in the migration from AS2 to AS3 a few of note that the average designer/animator should pay attention to:s

1: Code can no longer be placed directly onto symbols:

2: Anonymous functions are no longer supported:

3: While most of the timeline functions (*stop(), play()*) remain the same many other functions have been eliminated(*loadMovie()*) or replaced(*getURL()* is now *navigateToURL()*). Additionally, many of the property names have been changed as well, so "_xScale" is now "scaleX". This was done to brig the language more in line with other scripting languages.

Variables

Variables are simply containers for data, they hold information. In this regard a very good analogy for a variable is a cup. It really isn't the cup itself that is important it is whatever the cup contains that is really important, the cup itself is merely a vessel, a way of carrying something around. Like there are many different types of cups in the world so to are there many different types of variables. A variable can contain a number, a string(words), a boolean (true or false) value, a URL(web address), an array (list), or any of a wide varies of different types of data. The specific type of date that a specific variable contains is called it's data type.
Variables must be declared before they can be used, and follow a fairly straight forward pattern:

```
var variableName:DataType = value;
```

Lets say you wanted to declare a variable that would hold a numerical value, and set this value to 0:

```
var myVar:Number = 0;
```

Now lest say you wanted to create a variable that held a boolean (true or false) value. Well it follows the same pattern:

```
var myVar:Boolean = true;
```

Note: In the above examples the variable name myVar is used merely as an example. You can use any name you like for your variables and it is often very helpful use descriptive names. There are a few rules though; don't start variable names with a number, don't use any reserved words (the ones that turn blue), don't use spaces (the example uses a capitalization system called camel casing in place of spaces).

Now lets say that you wanted to create a variable that stored a web adress (URL) for later use, that would look something like this:

```
var myVar:URLRequest = new URLRequest("enter address here");
```

In this case the left side of the declaration sets up the name and declares the data type of the variable while the right side (the equal sign beign used to separate them) create it using the new keyword and sets its value (the address entered inside the parenthesis).

Creating Functions

Unlike variables which hold information functions are used to actually perform actions, they are the work horses of actionscript. They get things done. A function can contain a single command (called an argument) or multiple commands. In order to use functions you must do two things; declare the function (much like a varaible must be declared) and call or run the function.
Lets say you have an object on your stage with the instance name of mybox01_mc and you want to create a function that will rotate it 45 degrees and scale it up by 50%. Well the code for that might look something like this:

```
function myFunction() {
	myBox01_mc.rotation += 45;
	mybox01_mc.scaleX += .5;
	mybox01_mc.scaleY += .5;
}
```

Note: the "+=" pair of symbols are used to create a relative change, that is a change from the symbols current value. If you wanted to make an absolute change, that would completely ignore the starting value of the symbol you can simply use the "=" symbol. Also note that Scale does not use 0-100 as it's values but instead the more numerically accurate 0-1 (where values about or below 1 are written in decimal notation, so .5 would equal 50%). Scale and Opacity values both use this system.

If you were to test your movie now, absolutely nothing would happen, because while you have declared the function you haven't actually called it yet. On a line further down in your code after you have declared your function (if you have multiple functions it would occur after you have finished declaring all of your functions), you would want to add:

```
myFunction();
```

That's all it take to call a function just type out the function name followed by a pair a parenthesis and a semicolon. Assuming that you named everything properly and used the proper syntax you should now have a function that makes the box rotate and scale up when the movie runs.
Now a problem arises, what if you had two or three symbols on screen that you wanted to move as well? Using the above way of doing things you would have to write out thee different functions with three separate calls (one to each function). This doesn't seem very efficient does it? A better way to proceed would be to make the function more modular and reusable.

Creating a Modular Function

To create a function that is modular, that is to say a function that can be reused to effect any object really isn't that hard. First you declare a function like you normally would, but instead of explicitly defining the object you want the function to run on you create an internal variable in this case called "myMC" and tell the function that it is going to hold information about a MovieClip. This allows you pass the name of the movie clip to effect from the function call to the function itself.

```
function myFunction(myMC:MovieClip) {
	myMC.rotation += 45;
	myMC.scaleX += .5;
	myMC.scaleY += .5;
```

```
}
```

Now when you call the function you have to add the name of the movie clip you want to effect. The call to this new, more modular function would look like this:

```
myFunction(myBox01_mc)
```

The name of the movie clip to effect isn't the ony argument that can be passed to the function. You can also pass other information along to the function by creating additional variables, each one would need to be separated by a comma.

```
function myFunction(myMC:MovieClip, myRotation:Number) {
    myMC.rotation += myRotation;
    myMC.scaleX += .5;
    myMC.scaleY += .5;
}
```

The call to this new, even more modular function would look like this:

```
myFunction(myBox01_mc, 45)
```

Note: If you tell a function to expect two arguments you have to send two arguments (and in the right order) if you don't it causes an error. The way around this would be by setting a default value fro the variable like so:

```
function myFunction(myMC:MovieClip, myRotation:Number = 45) {
}
```

Events

The previous examples of functions were all triggered by the calling the function manually. This can be very useful sometimes to have the functioned triggered when the movie's playhead hits the frame with the function call, but at othertimes it isn't really a feasible solution. If you want to create functions that run in response to specific events that occur in your flash movie you are going to need to learn how to build Event Listeners and Event handlers. Event Listeners are objects (non physical objects) that sit around and wait (or listen) for a specific event to occur. That Event can be either user initiated (like a button click or roll over) or triggered at a specific time in the movie (like when it first loads). Once triggered an Event Listener fires off or runs an Event Handler. A handler is really just a specialized function that is called in response to a listener. An Event Listener usually looks somethng like this:

```
symbolName.addEventListener(EventType.SpecificEvent, eventHandlerName);
```

An Event Handler is a specializd function that responds specifically when called by a listener. There are a few things that are a bit different but mostly they are indistinguishable from other functions:

```
function eventHandlerName(myEvent:EventType):void {
    //arguments to perform
}
```

Note: The only real difference is the presence of "void" as the return type, specifying that the function will not return any data when it runs. For those of you used to AS2, listeners are a major departure from the way most people would create functions that ran as a result of user action in AS2. Formerly you could add code to the timeline that looked something like this:

```
symbolName.onRelease = function(){
```

```
    //arguments to perform;
}
```

This is called an anonymous function and would work perfectly alright when using AS2. Basically you were allowed to attach a event directly to the symbol instance and have that call a function. This code will cause a compiler error if used in an AS3 project.

Creating an Event Listener

Lets say you have a movieClip on your stage name "menu01_mc", it's going to be part of your Flash movies navigation and you want to allow the user to drag it around the screen. To do this you would have to create a pair of Event Listeners and Handlers. The first would respond to the user pressing down on the movieClip and the second would react when they release the mouse button.
First lets turn on the button mode for the movie clip, this allows the user to get the finger shaped button icon when interacting with it:

```
menu01_mc.buttonMode = true;
```

Then you will want to declare your event listeners (one for each action you want the movieclip to respond to:

```
menu01_mc.addEventListener(MouseEvent.MOUSE_DOWN, dragme);
menu01_mc.addEventListener(MouseEvent.MOUSE_UP, nodragme);
```

These two lines add listeners that are listening for specific mouse events (the first listeners for the mouse down event, the second for the mouse up event). Once the listeners detect that the user has pressed down on the object named menu01_mc it triggers the function named dragMe, and when it detects that the user has released the mouse over menu01_mc it triggers teh function called noDragMe.

Creating an Event Handler

As stated previously an event handler is just a function with a very specialized purpose. In this case it will be to start dragging the menu01_mc object when the user presses on it and to stopdragging the object when the user releases the mouse. In a very similar fashion to when you created a modular function you will make the handler modular so that instead of targeting an explicitly named object is is going to target whatever the initiating object that starts the event is. So you would be able to create additional draggable menus by simply creating more listeners without having to create more handlers.

```
function dragMe(myEvent:MouseEvent):void {
    event.target.startDrag();
}

function noDragMe(myEvent:MouseEvent):void {
    event.target.stopDrag();
}
```

In the preceding functions dot syntax is used to specify which object on the stage should be dragged. Read in english the argument for the dragMe function would sound something like this, "start the drag of the target of the event", in other words figure out which object initiated the event and affect only that object.

Linking to a Web Page

In AS2 creating a link to a web page was pretty easy and straightforward. The code usually looked a bit something like this:

```
symbolName.onRelease = function() {
    getURL("enter URL here");
}
```

In AS3 however it takes a bit more work; first you must store the URL you want to use in a URLRequest variable, then you will create an Event Listener that waits for a mouse click event and then runs an Event Handler that actually instructs the browser to navigat to the URL held in the listener. For the following code example the name of the URLRequest variable will be myLink01, the object on the stage that triggers the event will be called mywebButton01_mc, and the function that navigates to the webpage will be called myWebLink01.
First the variable is declared:

```
var myLink01:URLRequest = new URLRequest("enter URL here");
```

Second the Event Listener is declared:

```
myWebButton01_mc.addEventListener(MouseEvent.CLICK, myWebLink01);
```

Third the Event Handler is declared:

```
function myWebLink01():void {
    navigateToURL(myLink01);
}
```

Dynamically Load a Movie Clip from the Library

In AS2 the commands for loading a movie dynamically from the Library, loading an external image or Flash file, and an XML or Text file were all different. In AS3 in order to streamline the code they all take a very similar approach to the way they are written.

The first thing you will have to do is create a MovieClip and define a custom class for it in the Properties or Linkage dialog boxes from the Library. In this case the custom class is named "myCustomClass".

Secondly you will write the code that is used to create a new instance of the myCustomObject class and display it on the stage:

```
var myCustomObject01:blackball = new blackball;
var myCustomObject02:blackball = new blackball;
this.addChild(myCustomObject01);
this.addChild(myCustomObject02);
```

In this example the first two lines are object requests that creates an instance of the myCustomObject class. This new objects are then added to the stage using the addChild command. The name of the object request (where the variable name would normally go) becomes the instance name of the new obejct when it is on the stage. In this example two objects are created, the first named myCustomObject01 and the second is named myCustomObject02.

Dynamically Load an External Image

To load an image (or swf) dynamically from a folder on the server requires a little more work but is essentially the same process as loading a movie clip form your library.

First you will define a URLRequest variable to hold the location of the file you want to load into the program.

```
var image01:URLRequest = new URLRequest("enter URL here");
```

Second you will need to create an object request to create a Loader object.

```
var myLoader:Loader = new Loader();
```

In this case the loader object is named myLoader and it is instantiated it with the new keyword.

Third, the add child command is used to add the loader to the stage.

```
this.addChild(myLoader);
```

Forth, you have to display the image in the loader you previously created using the load method of the Loader class.

```
myLoader.load(image01);
```

In the preceding example, the load method of the myLoader object loads the URLRequest into the object named myLoader.

Well that brings us to the end of this short tips guide.
Good Luck! and I hope you have enjoyed these tips...And I also hope that I didn't screw anything up...JK.

About the Author
Jerron Smith likes to describe himself as an Editor, Animator & Educator. A multi-faceted artist and video producer with nearly two decades of experience, Jerron works with a wide variety of different media. He has experience in both digital video/television production, and post-production as well as extensive knowledge of the Web and Print Design industries. He serves as an adjunct instructor in the Communication Arts department at the New York Institute of Technology, where he instructs courses in Computer Graphics and Video Editing. He served as the lead author on Photoshop Elements 7 Digital Classroom (2009) and a contributing author on Flash CS4 Digital Classroom(2009), in addition to serving as a contributing author or editor on several other book projects. Jerron can be reached through his website: www.thepixelsmith.com