Advanced ActionScripting
Flashkit San Jose 15 August 2002
Phillip Kerman

Introduction:

If you are interested in creating interactive Flash applications, you need to write instructions using the ActionScript language. But translating your detailed ideas into the language Flash understands can be challenging. With an emphasis on practical solutions rather than theory, this session is designed for those who have ActionScript experience and want to move beyond fundamental programming concepts. You'll walk away with a clear understanding of how new Flash MX features can help you work including Shared Objects, Local Connection, Watch variables, Listeners, and homemade properties. Once you learn how to apply Object-Oriented Design to your projects you'll be more creative, efficient, and productive.

Fundamental new features:

Buttons and Text Fields (dynamic or input) all now have instance names. This means you can address a button or text field and change position properties (**_x** and **_y**). In addition, buttons have the new properties **enabled** and **useHandCursor** which—when set to true or false—control whether a button is clickable or whether the user will see the "finger" cursor. Text Fields have the new **text** property to supplant the old way of associating a variable with a text field (which had the side effect of turning any variable into a string). Everything you can do to a text field using the Properties panel (and more) is accessible through ActionScript.

The TextFormat object extends the ways you can format text. The TextField object is different because it always refers to a specific instance of text onscreen (like a Movie Clip). In the case of TextFormat objects, you store (text format) instances in variables, modify their properties such as **font**, **color**, **size** but then have to apply the format to a specific text field instance onstage in order to see any effect. Think of TextFormat objects as style sheets that can be applied to specific fields onscreen.

The new drawing methods add less than a dozen new methods to Movie Clips, but offer great power because lines and fills can be created at runtime. This means graphs and charts can display dynamic information. Perhaps charting a user's results to a quiz or any timely information that's loaded from an outside source. Of course you could also produce a primitive draw tool where users can create graphics. The process to create drawings using script involves creating a new (empty) movie clip; then moving the "pen" to where you want to start a line or fill; optionally specify that you're about to draw a fill; then using the lineTo or curveTo methods effectively drag the pen to a new location. Basically, you have to specify every detail though you can create a code library of common tasks such as drawing a square. It can definitely become a meticulous process.

New callback functions let you consolidate nearly all your code in a single keyframe. For example, if the following code is attached to a button:
`on(press){trace("hi");}` ...you could instead have this code in a keyframe:
`my_btn.onPress=function(){trace("hi");}//` where "my_btn" is the instance name of your button. On the surface this simply means you don't have to hunt around to find code that's nested in buttons inside clips and so on. However, it also means that you can change what scripts execute for any given event any time. For example, you can effectively turn off the **press** event on the button above with the script: `my_btn.onPress=null;` (A cool side effect of these callbacks is that clip instances can now behave like buttons—using the **onPress** callback for example.)

Additionally new callback functions have been added to trap previously unavailable events including some for asynchronous commands. For example (some of the most significant): **onSoundComplete** triggers when a Sound object sound finishes playing; **onChanged**, **onSetFocus**, or **onKillFocus** trigger when a specified text field is changed, touched, or untouched respectively; **onResize** triggers when the user changes the browser window size (for a movie HTML size 100% and scale:noscale). All this means is that you can write scripts that respond to events that were previously either impossible or very hard to track.

Externally linked JPGs and MP3s mean users only need to download the media they request. Simply, **loadMovie()** can now specify either a **.swf** or a **.jpg** file and the Sound object now includes a **loadSound()** method. Additionally, the new **duration** and **position** properties (and the previously mentioned onComplete callback) make tracking sounds possible.

Timed and Automatic scripts:

SetInterval lets you identify a function to a trigger automatically at a specified frequency. For example `setInterval(function(){trace("hi")}, 1000)` will display **"hi"** once a second (1000 milliseconds). You'll probably want to store the intervals in variables: `myInt=setInterval(func,1000);` so that later you can clear it using: `clearInterval(myInt);`

Watch lets you identify a function to trigger automatically whenever a specified variable changes. That is, you can "watch" a variable. A watch identifies the variable in question and the "response function" you want triggered whenever the variable changes. The syntax for that response function is very specific. For example, the second and third parameters contain the variable's former and current value. Also, the last line inside the response function must return the new value (`return newVal` if "newVal" is the name of the third parameter)—otherwise, the variable may never get changed. In the presentation I'll demonstrate a practical example of using watch. (Note: watch only works with variables and not so-called getter/setter properties.)

Listeners are similar to the callback events. That is, you define a function to trigger when a specified event occurs. It's just that with listeners there's a different set of available events plus you can easily mix and match, remove, or add multiple listeners that trigger when a specified event occurs. Also, because listeners work with the Key, Selection, and Mouse objects (in addition to text fields) you can "listen" for events that are otherwise impossible to trap. That's because traditional callback events are tied to specific instances where listeners can track keys the user presses, selections the user makes and when they use their mouse (which all don't involve any "instance" per se). Instead of laying out all the syntax, see the accompanying presentation material.

Custom classes and true homemade properties:

Flash 5 had ways to create custom objects (classes) as well as homemade properties but they were never tied to anything visual. That is, you could do cool stuff with objects but you'd only be doing them to variables which don't automatically have any visual counterpart onscreen. The profound Flash MX feature is that now, when you create formal classes, you have the opportunity to base your class on the built-in Movie Clip class. That is, you can make a class that has everything a movie clip does plus whatever you add.

The new #initclip and #endinitclip commands give you a way to ensure your class definitions (that go inside a clip) execute before the clip starts life as a Movie Clip. That is, if you want to tie a clip instance to your class you must make sure it's an instance and not an instance of the Movie Clip class. Code inside #initclip will execute before the clip instantiates so you'll have a chance to change its class—because once it's a Movie Clip you can't change it.

Making a custom class means that you can create custom methods (think "functions") and custom properties. While this is certainly possible by placing a function inside the first keyframe of a Movie Clip symbol, that will eat up memory unnecessarily with duplicated code. (All the code *in* a clip will be duplicated for each instance of that clip.) Finally, the new **`addProperty()`** method lets you design your own custom properties. You get to define what happens when the property is changed and how the value for the property is derived. For example, you could design a **`_left`** property for your custom class. Then simply executing: **`oneInstance._left = otherInstance._left;`** would align the objects' left sides. While all of this takes some time do design and program, it can payback with enormous time savings.

Reaching out:

The SharedObject (local) feature lets you read and write data to the user's hard drive much the same as cookies do. This way you can let users pick up where they left off in your website of letting them automatically bypass an intro animation after they've seen it once. The really cool part is you can store any data type in a SharedObject. For example, cookies are always string data—while you can parse strings into numbers, it's a pain. Now you can easily store strings, numbers, arrays, and objects that import back into Flash in their native format.

The LocalConnection object makes it easy for one movie to trigger functions in another movie playing in a separate browser window. For example, you might have several links inside your Flash movie that pop open a new window with additional information that keeps getting updated as you navigate a site. I could make a "guided tour" option on my website and—from a separate window—effectively press buttons in the main page. While the uses are rather unique, the basic idea and process is the same for when using the NetConnection object (in FlashCom, below).

The FlashVars tag and the LoadVars object give you two new ways to get data into your Flash movie. FlashVars is just a cleaner way for HTML files to set initial values for variables (effectively replacing the old way: **"my_movie.swf?onevar=value"**). The LoadVars object is similar to the old **loadVariables()** method but provides several enhancements. First, you don't need a clip instance just to load variables. Variables get loaded into a generic object that includes an onData() event triggered when data is fully loaded. Also, LoadVars provides the methods **getBytesLoaded()** and **getBytesTotal()** in order to monitor downloading lots of variables.

If all of this wasn't enough, the new Flash Remoting feature provides an entirely new protocol to connect Flash to application servers. You can easily bind a listbox inside Flash to an data source which will remain synchronized any time that data changes. Like FlashCom (below), Flash Remoting is nearly as large a topic as Flash itself.

Five Minutes of FlashCom:

You'll probably hear and see that the Flash Communications Server lets you stream live or recorded audio and video into a Flash movie. While that's true (and pretty amazing at that), I don't think it will prove to be the quintessential FlashCom feature. The mere fact that you can very easily make persistent connections to server logic makes all kinds of amazing applications possible. Also, the ready-built components make creating applications a snap—but doing things by hand it not super difficult. It does require a new thought process, but it's generally very easy.

My books:

I've included everything I think is important to the general Flash developer in the book: "Sams Teach Yourself Macromedia Flash MX in 24 Hours"(ISBN: 0672323710).

My scripting book takes a competent Flash user and turns them into a Flash programmer. This book is also suitable as your first programming book though it concentrates on Flash. "ActionScripting in Flash MX" (ISBN: 0735712956).