MUJ
MACROMEDIA USER JOURNAL™
http://www.muj.com

A Meaningful Conversation Between JavaScript and Flash

Phillip Kerman

If you think Flash has limited "programming" capabilities, think again. (And if you think Flash 4's new *Action Scripts* are a full-fledged programming language, you should think again *again*.) Anything JavaScipt can do, Flash can too. Flash can send and receive messages to and from JavaScript, so it's the logical step for those who want to go beyond the basic features of HTML and Flash. This article picks up where Flash's "FS Command" starts.

WHY bother learning JavaScript? After all, Flash 4 now has scripting capabilities like custom variables and built-in properties that you can access or change. What's the point learning *another* programming language if Flash can do it all?

That's exactly what I thought, having proposed to deliver a presentation on this topic at the Macromedia User Conference (UCON '99). I wondered if there was still a need for communication between JavaScript and Flash. There is. There are still limits in Flash that JavaScript can take you beyond. Besides, JavaScript is a full-fledged programming language, and Flash 4's "Action Script" language is, frankly, limited. Finally, I believe that every minute invested learning JavaScript is time well spent because the knowledge is transferable—indirectly when you learn other languages (like Director's Lingo dot-syntax), and *directly* as JavaScript becomes the default language for user add-ons to Macromedia products (more about that in the summary).

You *can* do a lot within Flash. The need for JavaScript arises after you exhaust the possibilities of "tell target" and "load movie" (see Darrel Plant's article "Messaging in Flash: Using Tell Target and Load Movie Commands" in the June 1999 issue). "Tell target" lets you direct messages to specific movie clip instances, and "load movie" enables you to modularize your content. This article assumes you understand these two techniques because we pick up where they stop, and analogies will be made to them.

Every Flash is an island

The fundamental limit of Flash (now that it supports variables) is that Flash can't talk and it can't listen. An exception is the form *POST* and *GET*. However, if you want to *tell* Flash to start a movie in the middle—you have to *say* it *inside* Flash. Suppose you have a long introduction animation on your home page. After the user watches it, they probably don't want to sit through it again. A "skip introduction" button (right *in* the Flash movie) is a common solution to this annoyance. But what if your HTML document remembered that the user watched the animation and *told* the Flash movie to skip ahead? It can! You can store a variable, say "seen_intro," in JavaScript (in an invisible HTML frame, if you want), and then, depending on the value of "seen_intro," you can *tell* the Flash movie to "GoTo" a particular frame number.

Are you talkin' to me?

Flash can *talk* to JavaScript, and JavaScript can *talk* to Flash. We'll let JavaScript do the talking in a minute. The way Flash

talks to JavaScript is through the "FS Command" (FS stands for "Future Splash"—the original name for Flash, in case you're wondering). It's a bit confusing, because if you look up FS Command in the Flash manual, you'll only learn about FS Command's *other* use—namely, to "talk" to the standalone projector. Things like Quit and Full Screen, for example. These have nothing to do with JavaScript.

When you want Flash to talk to JavaScript, simply place an FS Command action in the Flash movie (in a frame or under a button), and when the user reaches that action (that is, reaches the frame or presses the button), a signal is sent to the "host" HTML file, and JavaScript is invoked. The JavaScript can then simply execute a built-in function (like "alert"), invoke a custom function, or process the call in multiple steps. Once the message is sent from Flash, it's up to JavaScript to deal with it.

JavaScript can talk to a Flash movie in one of two basic ways. Either JavaScript simply *tells* the Flash movie to do something (like go to a frame, or stop, etc.) *or* JavaScript can *ask* Flash for information—like, "Hey, Flash, what frame are you on right now?" Then JavaScript can do what it wants with that information. JavaScript can talk to Flash at any time—but since JavaScript is an event-based language, the "discussion" usually begins after a standard event occurs (like onMouseOver, onClick, or onLoad).

It's not always a clear case of JavaScript talking to Flash, or vice versa. Flash can say to JavaScript: "Do something now," but part of that *something* can include JavaScript asking Flash for information, then telling Flash to do something.

Before you start

A few basics are worth learning first. The JavaScript language is organized in the "document object model" (or DOM)—a logical way to access properties by first specifying the object, then the property. Think of it as general to specific. So if you want to refer to the weather in my hometown, Portland, Oregon, you *don't* say "the weather in Portland, Oregon USA"—rather, "USA.Oregon.Portland.Weather". Also, don't forget that when referring to properties, you can be *setting* them or *getting* the current value. (In either case, with Portland, the weather will be rain.)

Now, let's move on to the "embed" tag. Although you might not need to memorize the HTML embed tag, you should understand the basics. The Flash "object" needs to be named (ID="movieName" for IE and Name="movieName" for Netscape). In Netscape, there's another tag required: "liveConnect=TRUE" (which causes that interminable "Starting Java…" message to appear in the browser—more about that later). Alternatively, you can choose "Flash with FS Command" from the "Template" drop-down in the "HTML" tab in Flash's "File > Publish Settings" dialog box. However, you probably won't learn anything doing it this way, so just use these ready-built templates as guides. And be careful: I found a bug in this particular template, as it's actually missing the "Name=movieName" tag for Netscape (to fix the template, open the "FSCommand.html" and add "NAME=$TI" to the end of the line containing "swLiveConnect=true").

Flash talking to JavaScript

The message flow when the FS Command action is encountered is pretty straightforward. The message comes from Flash and is always channeled through the JavaScript function "DoFSCommand". Once inside the DoFSCommand, JavaScript will execute any number of lines of code you include. What if you want two separate buttons in Flash to invoke two separate actions? They both still go through the DoFSCommand, but the first parameter received is the string you've typed into the FS Command action's "Command" field. If the separate actions have different "command" parameters, then it's a simple matter to have JavaScript determine the value of the command. (If command="this" then do this, if command="that" then do that.) The DoFSCommand is the "clearinghouse," and all messages *from* Flash are channeled through this function.

Figure 1 shows one big DoFSCommand function that sorts out three separate actions from Flash. The first two buttons send the command parameter of "say_something" with an additional "argument" parameter of "Red" and "Green," respectively. (See **Figure 2** for detail on the button's action in Flash.) The "do_and_say" command causes the DoFSCommand JavaScript to step through several procedures. So, you see, it's not just Flash saying "DoFSCommand"… it's Flash saying "DoFSCommand, this command, with this additional argument." And, when JavaScript *hears* DoFSCommand, it knows to expect two parameters and can determine the value of each. Finally, notice that in the examples, the function name isn't "DoFSCommand" but rather "flashObj_DoFSCommand"—that's because the *name* given to the embedded Flash movie

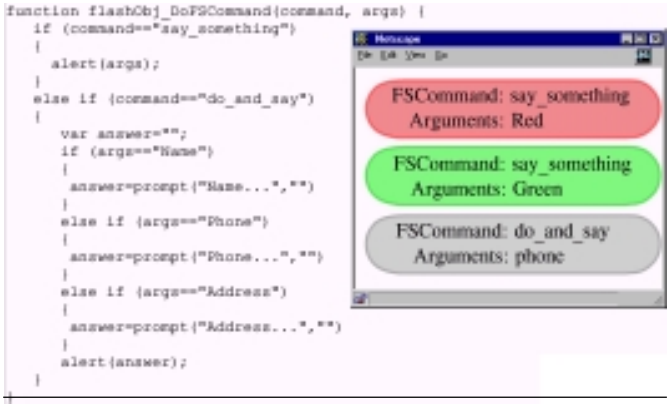

Figure 1. Three buttons in one Flash movie can all "talk" to JavaScript. Each button invokes the DoFSCommand function, but depending on the value of the parameters sent (command, args), the function will act differently.

is added as a prefix to the JavaScript function. (The Flash movie's name, in this case, is "flashObj".) This way, you can have several Flash movies in the same Web page and they'll have their own private DoFSCommands.

JavaScript talking to Flash

JavaScript talks to Flash by referring to the embedded "Flash Object" (that is, the name or ID we gave it in the embed tag). Remember, JavaScript can either *tell* Flash to do something or *ask* Flash for information. In either case, the technique is the same: if the Flash movie's "name" is "flashObj", then the JavaScript reads flashObj.doSomething() or aVariable=flashObj.getInformation(). There isn't really a "doSomething" or "getInformation" property (or "function")—the bulk of properties available (including new ones for Flash 4) are in **Table 1** on page 4. You'll find things like Play(),GotoFrame(), and IsPlaying(). Just remember, some properties simply *do* something (like "play") and other properties *return* values (like "isPlaying").

The occasion for JavaScript talking to Flash is usually the result of a JavaScript event. That is: JavaScript can talk to Flash any time, so when does it start talking? On some event. Events such as onClick() and onMouseOver() are basic JavaScript events. So, to make a text "stop" button for your Flash movie, you can have a hyperlink in the HTML that reads:

```
<a  href=# onClick="flashObj.stop();">Stop</a>
```

The URL "#" prevents the link from actually going anywhere. In this case, the JavaScript talking to Flash is right down there in the href. It might be more efficient to call your own JavaScript function that, in turn, talks to the Flash movie.

Not just a one-way street

Remember, it's not just Flash or JavaScript talking to the other. The way Flash talks to JavaScript is via the FSCommand, and the way JavaScript talks to Flash is by accessing one of the available methods. Each case is a one-way conversation. However, once the conversation starts, it can keep flowing. Flash can invoke the DoFSCommand, which, in turn, calls another JavaScript function. You need to think of each step in detail, but that doesn't mean you have to be limited to simple one-way conversations. For example, one Flash movie can tell JavaScript to tell *another* Flash movie to stop. In a recent project, we put all of the sounds in one Flash "audio" movie (which never reloaded). Then any of the other Flash movies that appeared would tell JavaScript to tell the audio Flash movie to go to a particular frame (with the desired audio). The point is, don't limit your thinking.

The easy way

The "liveConnect" tag will cause Netscape to pause while the runtime for JavaScript boots up. This is necessary anytime you want Flash to use the FSCommand. However, there's a much simpler solution (that doesn't work on some older browsers). You can use the Flash action "getURL" and specify the URL as "javascript:alert("Hello");"—simply "javascript:" followed by the actual JavaScript code. It can be a bit hard-wired, and it doesn't work on some browsers—however, it's easy and doesn't delay during that long "Starting Java…" message. This method is almost identical for how Flash talks to Director's Lingo—namely, a "getURL" action with "lingo:theLingoCode()".

Sounds easy, but…

There are several "gotchas" of which to be aware. I lied when I said all messages from Flash to JavaScript travel through the FS Command. In Microsoft Internet Explorer (IE), the VBScript subroutine is invoked, but inside this subroutine, the regular JavaScript FS Command is "called." I suppose you *could* duplicate your code, but it makes sense to keep all of the code in one place (the FS Command) and simply re-direct IE to the JavaScript.

There's not much creativity to writing the VBScript re-direct (just copy and paste it from the HTML produced from Aftershock or Flash 4's new "publish" command). However, since some browsers don't like the VBScript just sitting out, most people build the entire subroutine as a string and then use the "write" function to put it in the HTML document. (See the template "FSCommand.html" in Flash 4's "HTML" folder for a sample.)

Continuing with the theme "they've gotta be different" (Netscape and IE, that is): In Netscape you can refer to the embedded Flash object (which we named "flashObj") with window.document["flashObj"]. In IE, you simply use window["flashObj"]. In either case, you follow this by a decimal and the property you want to get or set. (Window isn't necessary when you're referring to a Flash object in the current window—however, it's a good practice to include it

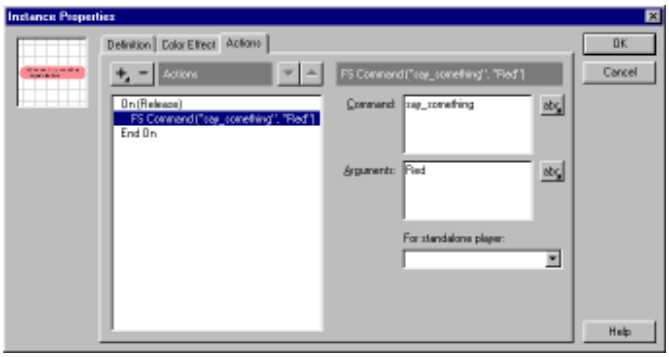

Figure 2. Flash talks to JavaScript though the FSCommand action. In the field for "Command:" the first parameter ("say_something") can be entered, and in the field for "Arguments," the second parameter can be entered. (You can enter more than one parameter in the "Arguments" field; however, JavaScript will need to sort things out.)

and quite necessary when you refer to objects in other windows.) This method should be familiar to anyone who's referred to different frames or windows in HTML—it's the same method. Of course, this (object reference) difference doesn't require a lot of duplicated code—just write a simple function (like the "getObj" in **Listing 1**) that returns the appropriate reference. Important note: It appears that in Netscape, it's much safer to refer to the Flash object by "embed index number." That is, instead of saying window.document["flashObj"], if you *know* the Flash object is the first thing to be embedded, you should say: window.document.embeds[0].

Listing 1. Since Netscape and Microsoft Internet Explorer refer to embedded Flash objects differently, you can call this "getObj" function every time you need to refer to the Flash object.

```
function getObj (whatObj)
{
var IE = navigator.appname.indexOf["microsoft") != -1;
    if{IE}
    {
        return window[whatObj}
    }
    else
    {
        return window.document[whatObj]
    }
}
```

Table 1. Most of the properties of a Flash object are controllable from JavaScript. Notice, the bold text needs to be replaced with explicit integers (**int**) or strings (**str**).

Telling the Flash movie to do something

Property	Result	Notes
Play()	Plays the movie.	
StopPlay()	Stops the movie.	
GotoFrame(**int_frameNum**)	Jumps to a particular frame in movie.	Remember to use "Play()" if you want "goto & play".
Rewind()	Rewinds the movie.	
SetZoomRect(**int_left, int_top, int_right, int_bottom**)	Modifies the zoom rectangle.	
Zoom(**int_percent**)	Sets the zoom rectangle by percentage.	
Pan(**int_x**, **int_y**, **int_mode**)	Pan movie by x/y pixels or percentage	*(if mode 0=pixels, 1=percentage)*

Asking the Flash movie for information

Property	Return value
TotalFrames()	Integer of total frames.
PercentLoaded()	Integer 0 through 100.
IsPlaying()	Boolean TRUE if movie is playing (otherwise FALSE).

Telling a specific movie clip to do something (*tell target*)

Property	Result	Notes
LoadMovie(**int_Level, str_URL**)	Loads a movie into specified level.	
TGotoFrame(**"flash_0/clip ",int_frame**)	Goes to a specific frame in the specified clip.	Remember to precede with "flash_0" (or in whatever level the clip exists).
TGotoLabel(**"flash_0/clip ",str_label**)	Goes to a specified label in the specified clip.	See above.
TPlay(**"flash_0/clip "**)	Makes the specified clip start playing.	See above.
TStopPlay(**"flash_0/clip "**)	Stops the specified clip.	See above.

Asking a specific movie clip to for information

Property	Return value
TCurrentFrame(**"flash_0/clip "**)	Integer of current frame number in the specified clip.
TCurrentLabel(**"flash_0/clip "**)	Current frame.

New for Flash 4:

Telling Flash

Property	Result	Notes
SetVariable(**str_name, str_value**)	Sets a user variable to a value.	
TSetProperty(**"flash_0/clip ", str_propName, str_value)**	Sets any property of a specified movie clip to a value.	See "Available properties" below to see range of values for "**propName**".
TCallFrame(**"flash_0/clip ", int_frame**)	Executes the new "call" function to effectively "go" to a frame in a clip by number.	
TCallLabel(**"flash_0/clip ", str_frame**);	Executes the new "call" function to effectively "go" to a frame in a clip by name.	

Asking Flash

Property	Return value
GetVariable(**str_name**)	Returns the string value of a variable.
TGetProperty(**"flash_0/clip ", int_propNum**);	Returns the integer value of a property. Notice you don't refer to properties by string name—rather, the **int_propNum** is found on the "Available properties" list below.

Available properties

Property	Code number	Property	Code number
POS_X	0	VISIBLE	7
POS_Y	1	WIDTH	8
SCALE_X	2	HEIGHT	9
SCALE_Y	3	ROTATE	10
CURRENT_FRAME	4	TARGET	11
TOTAL_FRAMES	5	DROPTARGET	14
ALPHA	6	URL	15

As if the Netscape and IE differences weren't funky enough, there's more:

- **Learn to count "0,1,2,3…":** Learn to count starting with zero. Flash calls the first frame 1, while JavaScript starts with 0. So if you want to go to frame 1, and you're in JavaScript, you need to say frame 0.
- **Movie clip hierarchy:** Luckily, you'll find many properties of Flash movie objects that allow JavaScript to talk to specific movie clips by their "instance" name. However, you must remember to precede the instance name with "_flash0/" (if the clip is on the stage level, "_flash1/" for level 1, etc.). Additionally, a movie clip in a movie clip can be accessed easily with "flash0/clip/clipInClip".
- **Kick it!** In Flash, the default for a GoTo action is effectively "GoTo and Stop". "GoTo and Play" requires you to check the appropriate box. Similarly, when you use JavaScript to GoTo a frame, you should include a second line that gives the movie a little kick-start. The property Play() will do the trick.
- **Wait for it:** There's no way around this one… you simply cannot talk to a Flash movie until it's entirely downloaded. Of course, there are strategies to deal with this fact. If it's the Flash movie that starts the conversation, you can simply use the built-in "if Frame is Loaded" action to stall until the *last* frame is loaded before telling JavaScript to start talking. But often, you want the JavaScript to tell the Flash movie something onLoad (the JavaScript event called when the page loads). Luckily, there is one property you *can* access before the movie is entirely downloaded—percentLoaded. Create a loop in your JavaScript that keeps checking whether percentLoaded has reached 100 before starting to talk to the Flash movie. Use such a loop (see Listing 2) anytime there's any possibility the JavaScript could be trying to talk to Flash before it's downloaded.

Listing 2. Although JavaScript can't start talking to the Flash object until it's downloaded, you *can* ascertain if it's reached 100 percent loaded. This loop stalls until the movie has completely downloaded.

```
var movie_ready="false";
var theObj=getObj ("flashObj");
while (movie_ready=="false")
   {
    if (theObj.PercentLoaded() == 100)
         {
          movie_ready="true";
         }
   {
//Go on...
```

Workarounds

Any function that's called directly or indirectly from a Flash movie talking to JavaScript can't include the simple HTML "location.href=page.html". It does seem to work on some browsers, and when the "page.html" is an absolute URL (that is, "http://www.server.com/page.html"). The workaround is to simply set a timeout of 0 and then execute the location function:

```
setTimeout("location.href='page.html'",0);
```

Another function that doesn't seem to like working within the FS Command is "window.open()". Again, the setTimeout function is used:

```
setTimeout("window.open('other.html')",0);
```

JavaScripters unite!

Learn JavaScript. The time spent is well worth the investment. Macromedia is providing more and more support to JavaScript, in the form of a yet-to-be-released Xtra for Director to execute JavaScript, in the form of objects for Dreamweaver and for new custom behaviors for Fireworks (yet to be released), and, of course, in the way Flash and JavaScript can talk to each other.

Even with Flash 4, there's a big need for JavaScript. JavaScript is a much more sophisticated language (multiple data types, etc.), and it's got a bigger user base, too. You'll be able to find JavaScript resources far and wide.

Plus, all of the code you create in JavaScript will be transparent—that is, you'll have everything sitting out in a neat text file instead of hidden under a million little button actions or frame actions in multiple scenes, etc. This transparent code will translate to other languages, too.

So Flash is great, and Flash 4 is even better—but when you reach the limit of Flash, JavaScript is there to take you beyond what *either* can do alone. (You can see a fully annotated version of the original presentation at http://www.teleport.com/~phillip/ucon99/presentation/presentation.html.) ▲

Phillip Kerman splits his work between *doing* projects and *showing* others how. He writes articles, prepares and delivers courses, as well as presents at international conferences—including the upcoming Alternative Authorware Conference (TAAC '99) in Orlando in October. Quite often, his role in projects involves creating and maintaining templates, though he's no stranger to dredging through lots of details. He's an expert with Director, Authorware, and Flash. His clients also value his input on informational design and production efficiency matters. www.teleport.com/~phillip, phillip@teleport.com.