Captivate-to-Flash ActionScript Communication

Philip Hutchison
Instructional Technologist
UCSF Medical Center
http://pipwerks.com/lab/captivate

Prepared for the eLearning Guild DevLearn 2007 conference
November 7, 2007

Table of Contents

Session information

Session description

Many Adobe Captivate users need to make ActionScript calls from their Captivate SWF to a Flash container SWF, but Captivate does not support making direct ActionScript calls.

Session participants will see demonstrations of two primary workarounds for Captivate's limitation. You will learn how to use the two workarounds to make limited ActionScript calls to their Flash container SWF. Participants will have the demonstration files to take home.

In this session, you will learn:

- About Captivate's ActionScript limitations
- The capabilities and limitations of the two workarounds being discussed
- Workaround 1: How to use imported Flash SWFs to send ActionScript calls to a Flash container SWF
- Workaround 2: How to use Flash's ExternalInterface class to send ActionScript calls to a Flash container SWF

Intended audience

Intermediate and advanced developers with intermediate-level experience using Adobe Captivate 2.0 or 3.0, ActionScript, and JavaScript, including a basic understanding of variables and functions.

Background: What seems to be the problem?

Captivate has no ActionScript feature!

- Captivate is meant to be an easy tool for non-programmers to use, and therefore does not include scripting capability.
    - This bucks the trend established by Director, Authorware and Flash.
- Generated SWFs are built with ActionScript, but Captivate provides no method for tapping into a SWF's ActionScript functionality.
- SWFs can make *JavaScript* calls, but only in limited circumstances.
- Captivate 3 offers no advances in custom scripting functionality over Captivate 2.
    - We must continue to make do with workarounds (for now?).

What are some possible uses for Captivate-to-Flash communication?

- Letting the Captivate SWF unload itself from Flash container at a specific time.
- Telling the Flash container to load a different SWF (useful for branching scenarios that extend beyond a single SWF).
- Instructing the Flash container to perform some kind of action at specified points in the Captivate SWF. Examples: A course progress meter; color or graphical changes in the player SWF based on actions taken in the Captivate SWF (such as a hot/cold meter); pop-up messages; timer displays; a custom scoreboard for completing certain activities; logging/tracking user choices in a branching scenario; etc.

How do we overcome these limitations?

With workarounds, of course! There are two significant workarounds that can be used to enable Captivate SWFs to send ActionScript calls to Flash SWFs.

Workaround 1: Flash SWFs can be imported into a Captivate file and used to send ActionScript calls to a Flash player/container SWF

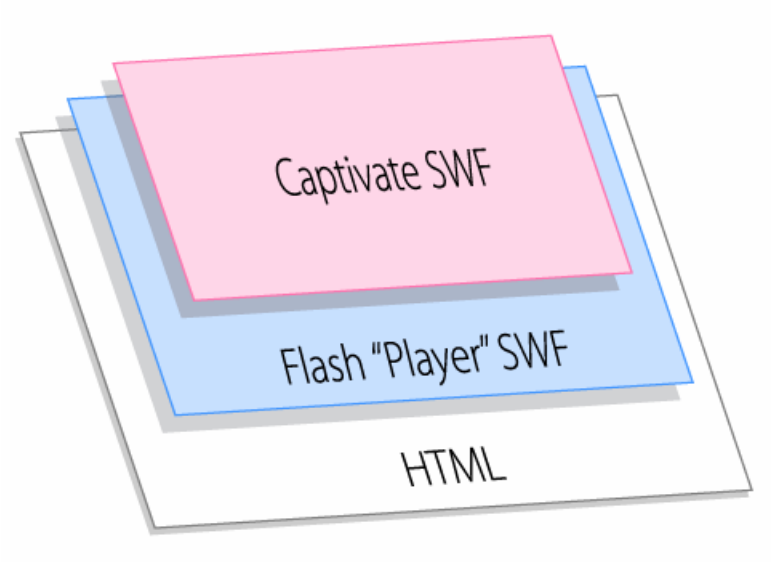
Workaround 2: Flash's ExternalInterface class can be used receive JavaScript calls from Captivate and convert them into ActionScript calls that get sent to a Flash player/container SWF.

Important note: All communication is ONE-WAY from Captivate to Flash. These workarounds do **not** enable Captivate to **receive** custom ActionScript calls.

First things first: Get the files

Before we can get into our Captivate-to-Flash communication workarounds, we need to create a Captivate file, a Flash "player" file (referred to from here on as "player" or "player SWF"), and an HTML file to hold the player SWF.

The HTML file loads the player SWF, which in turn loads the Captivate SWF. The nesting structure looks like this:

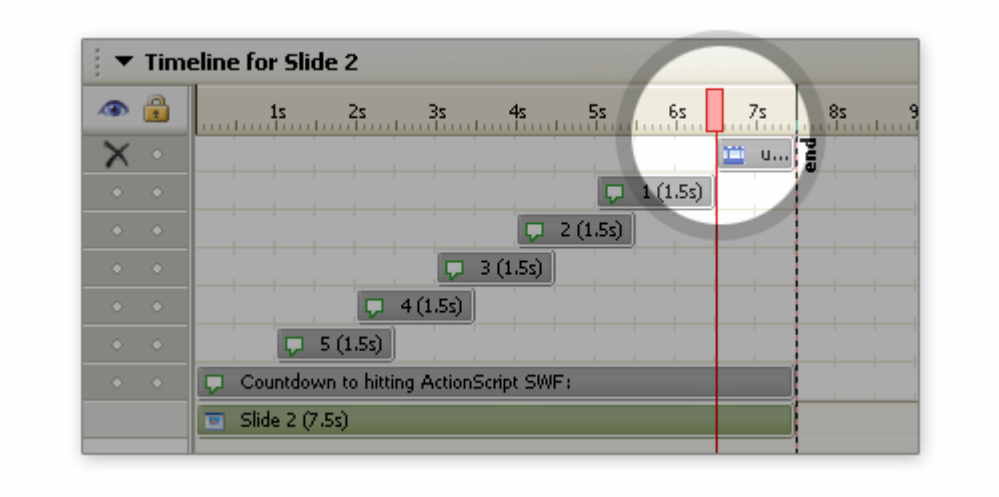


To save time, I've already created these files. To download the files, visit http://pipwerks.com/lab/captivate.

Workaround 1: Embed Flash SWF into Captivate

How it works

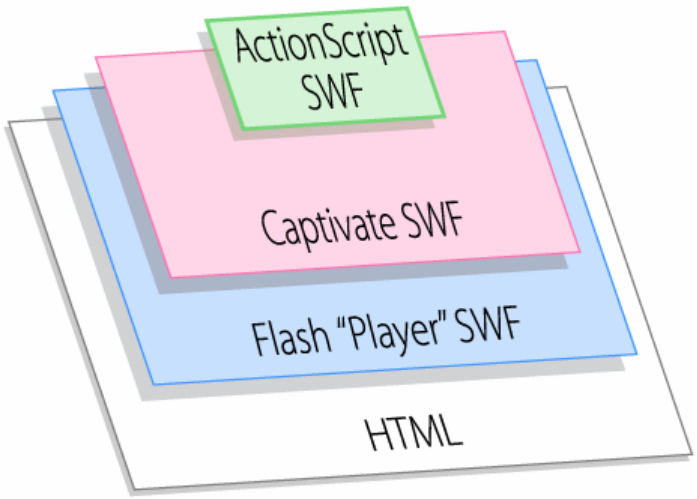
The key to this workaround is importing an 'ActionScript SWF'—a Flash SWF containing one line of ActionScript—into the Captivate movie. The imported Flash SWF's ActionScript will be executed when the Captivate play head reaches the imported SWF.



It's important to note that Captivate's handling of the imported ActionScript is best described as temperamental, and should be limited to a single, simple function call, such as `_root.unloadMe();`.

Files required

1. HTML file.
2. Flash "player" FLA.
3. Captivate file.
4. Flash "ActionScript" FLA containing one line of ActionScript (will be embedded into Captivate file prior to publishing).

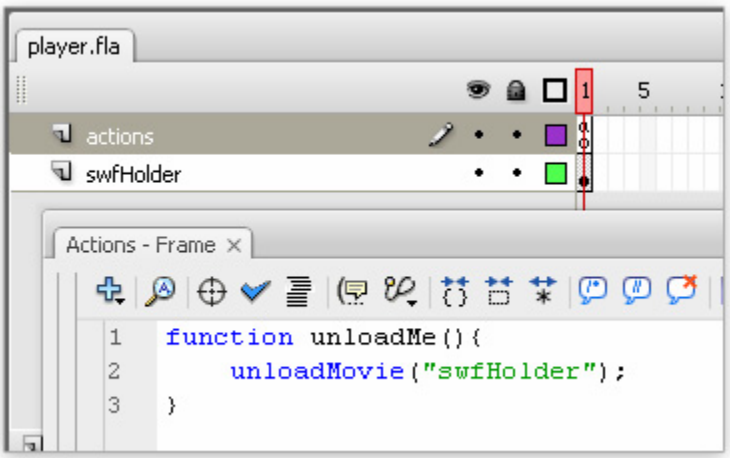


Instructions

1. In player.fla, create the function you'd like the Captivate SWF to invoke

Open player.fla and write a function that you would like your Captivate SWF to invoke. The function should be contained in the first frame of the FLA's timeline.

For this example, we'll write a function named "unloadMe" that does exactly what the name implies: when invoked, it will unload the Captivate SWF from the player.



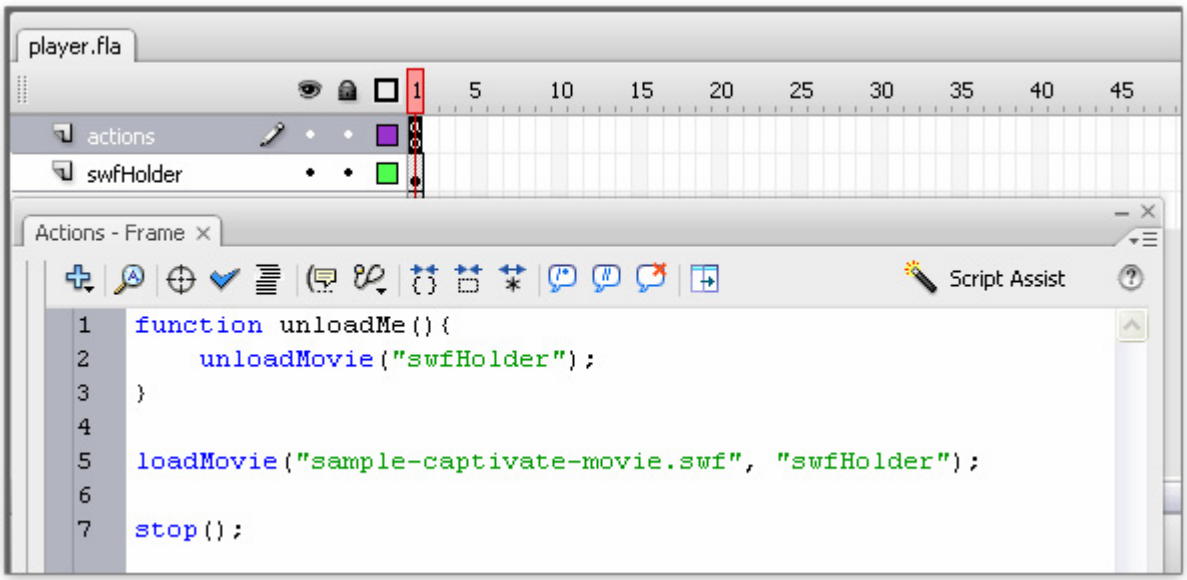

```
function unloadMe(){
  unloadMovie("swfHolder");
}
```

2. Add the loadMovie code

While we still have player.fla open, let's add the *loadMovie* code we'll need to automatically load our Captivate SWF for us. We also need to add the stop() command to prevent the player.swf from looping.

```
loadMovie("sample-captivate-movie.swf", "swfHolder");

stop();
```



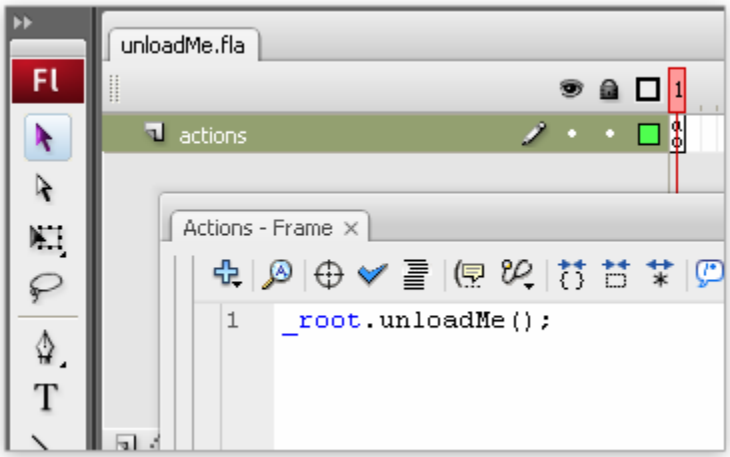
3. Publish player.fla

Publish the file. Turn off the HTML option, since we don't need any HTML generated for this SWF. Also make sure the SWF is published as Flash Player v7 for maximum compatibility (this demonstration uses ActionScript 2.0).

4. Create the "ActionScript SWF"

Create a new, blank FLA in Flash. Add the following line of ActionScript to the first frame:

```
_root.unloadMe();
```



Note: I've named the default layer "actions"; this isn't a requirement, but it's a good practice.

Save the FLA file as "unloadMe", which is the same name as the function it contains. This makes identifying "ActionScript" SWFs much easier if you're using more than one in your Captivate file.

Publish the file. Turn off the HTML option, since we don't need any HTML generated for this SWF. Also make sure the SWF is published as Flash Player v7 for maximum compatibility (this demonstration uses ActionScript 2.0).

A word about _root

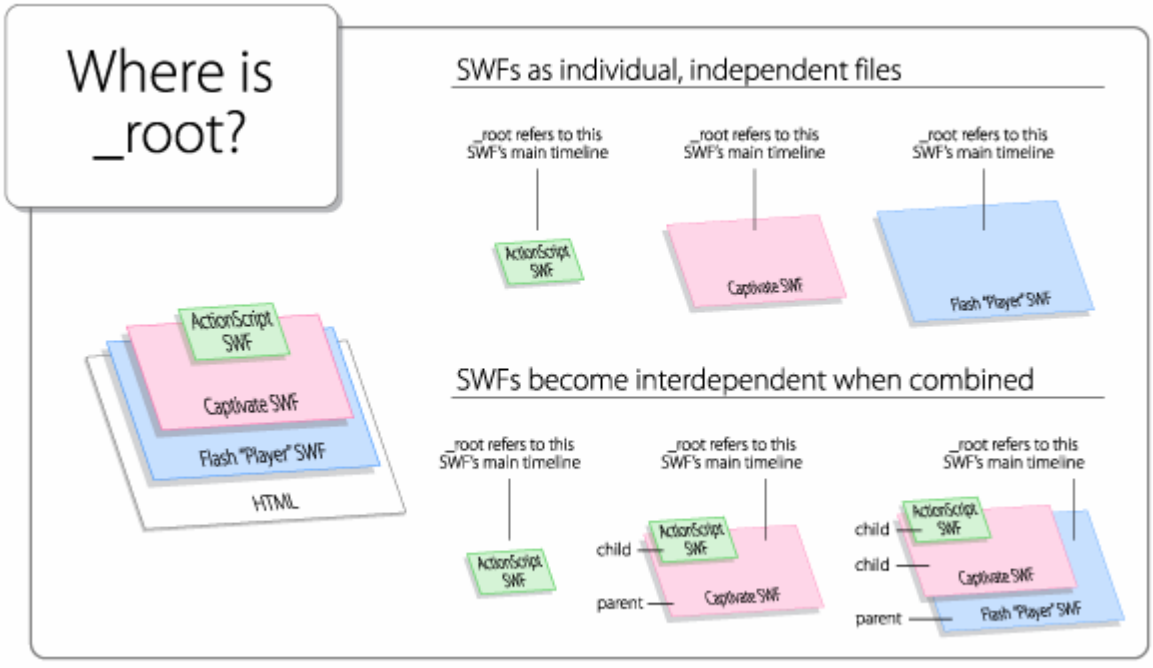
Warning! Boring code explanation! You can skip this section if you're comfortable trusting me when I say you should add `_root` to your functional call.

"Root" refers to the main timeline of the parent movieclip; adding the prefix `_root` to `unloadMe()` eliminates scope problems. In simple terms, adding `_root` helps Flash Player determine where the function `unloadMe()` is located, and which SWF is responsible for executing it when it gets invoked by the Captivate SWF.

If `unloadMe()` is invoked by the Captivate SWF *without* the prefix `_root`, the Flash Player plug-in will look for a function named `unloadMe()` in the main timeline of the *Captivate* SWF. Since the function is not stored in the Captivate SWF, Flash Player won't be able to find and execute the function.

By adding the prefix `_root` to `unloadMe()`, we've instructed Flash Player to look for a function named `unloadMe()` in the main timeline of the *parent* SWF.

In our scenario, the Captivate SWF has been loaded into the *player* SWF; in Flash Player's eyes, this makes the Captivate SWF a child of the player SWF. Therefore the reference to `_root` will point to the timeline of the *player SWF*.

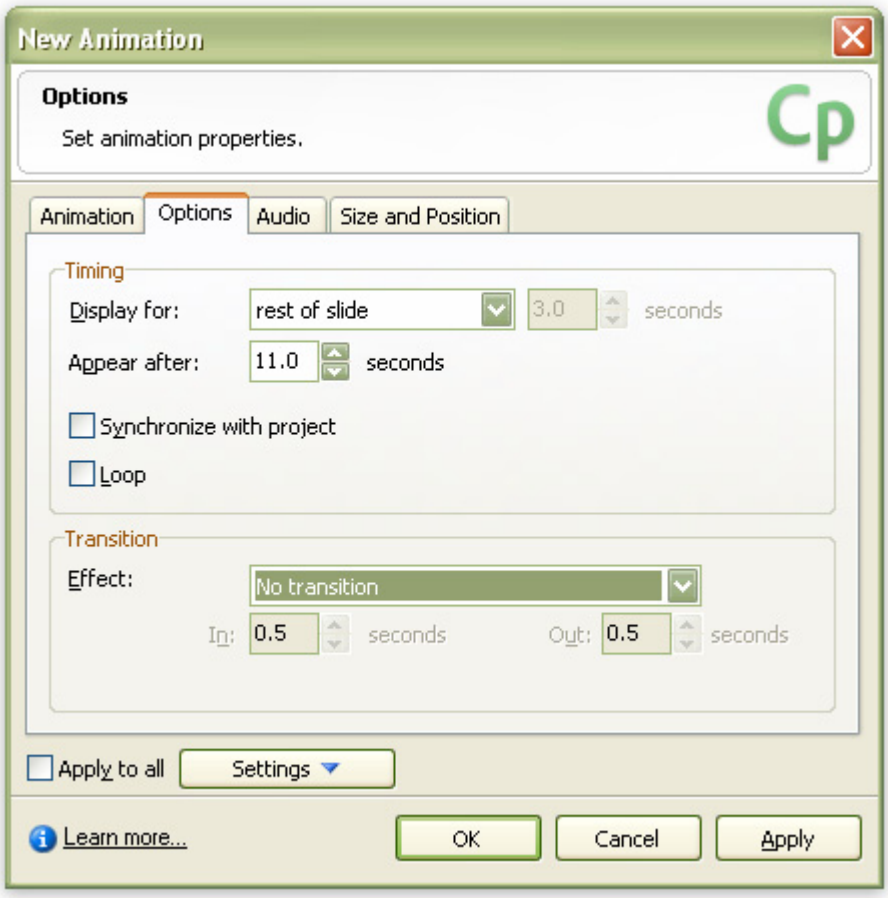


5. Import the ActionScript SWF into the Captivate movie

Open the Captivate file and navigate to slide two. *Note: The file provided for this demonstration is a Captivate 3 file.*

Choose *Insert* > *Animation* from the menu at the top of the screen. Select the *unloadMe.swf* file. You will see a warning about using "root". This is safe to ignore (see *A word about _root*); click *Yes*.

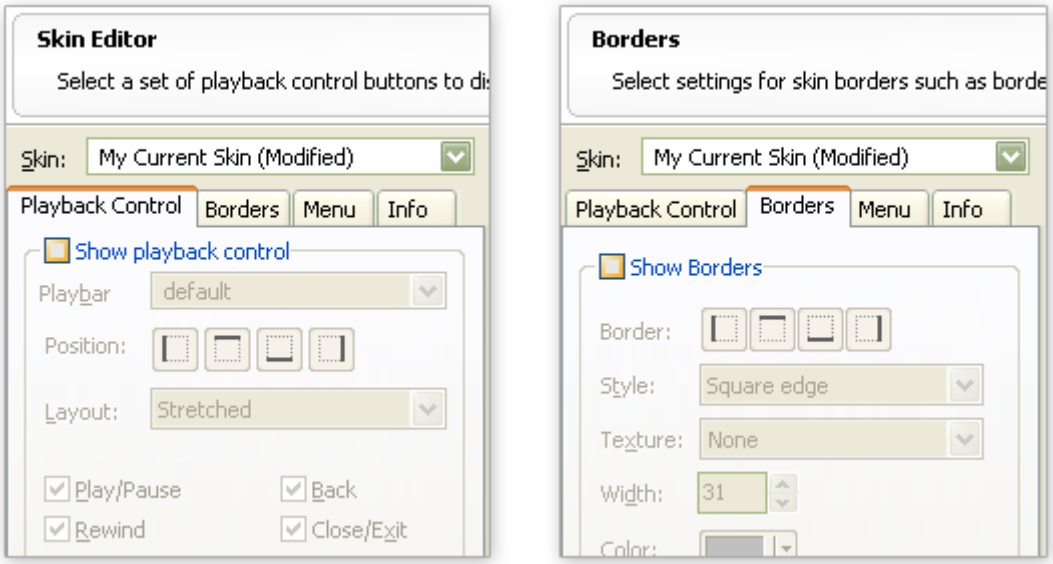
The *New Animation* window will appear. Leave the settings in the *Animation* tab as-is. Go to the *Options* tab; under *Display for*, select "rest of slide". Under *Effect*, select "No transition." Click *OK*.



The ActionScript is now a part of your Captivate movie. Save the Captivate file, but don't publish it yet.

6. Turn off skinning

Since this Captivate movie will be loaded into another SWF (our custom player.swf), it's safe to turn off Captivate's skinning feature. Go to *Project* > *Skin* and uncheck "Show playback control" — we don't need Captivate's playback controller — then uncheck "Show borders".



Note: If you preview the new Captivate SWF in Captivate or anywhere outside of the player SWF, nothing will happen when Captivate plays the ActionScript SWF. If you recall, this is because the function `unloadMe()` does not exist in the Captivate file; the function is stored in the player SWF. It can only be executed if the Captivate SWF has been loaded into the player SWF.

7. Publish the Captivate file

Click the *Publish* button. The *Publish* window will appear. Use the following settings:

- Flash SWF as the output type.
- Uncheck "Export HTML" (you will get a warning, but it's safe to ignore it).
- Select Flash Player 7 as the Flash Player Version. This example will work in Flash Player 7 and higher. In some of your own projects, you may need to choose a higher version of Flash Player.

8. Test it!

Now that your files are ready, test them out! Make sure the following files are in the same folder:

- index.html
- player.swf
- sample-captivate-movie.swf

Open index.html in your web browser. The sample Captivate movie should automatically load. Click the *Click here to go to the next slide* button. You should see the countdown, followed by the SWF unloading itself.

Workaround notes

Pros

- Easy to send pure ActionScript calls from Captivate SWF directly to Flash "player" SWF.
- Works with older versions of Flash Player.
- Can be easily used in local environments, such as on a hard drive or CD-Rom.
- Easy to control timing for invoking the ActionScript.
    - The ActionScript isn't bound to Captivate objects (such as buttons) or events (such as the 'end of slide' event), which enables you to invoke ActionScript at almost any time during the movie.

Cons/limitations

- Bulky: this workaround requires creating and importing a new SWF for each custom ActionScript call. This gets tedious when using a large number of imported SWFs.
- Limited usefulness: The embedded Flash SWF can't do much within the Captivate SWF beyond calling a function stored in the player SWF. For example, Flash-based forms and components won't work when imported into a Captivate movie, and conditional statements often fail due to scope issues.
- This workaround is restricted to nested SWFs (a Captivate SWF loaded into a container/player SWF); the Captivate SWF cannot send ActionScript calls to any other SWFs.

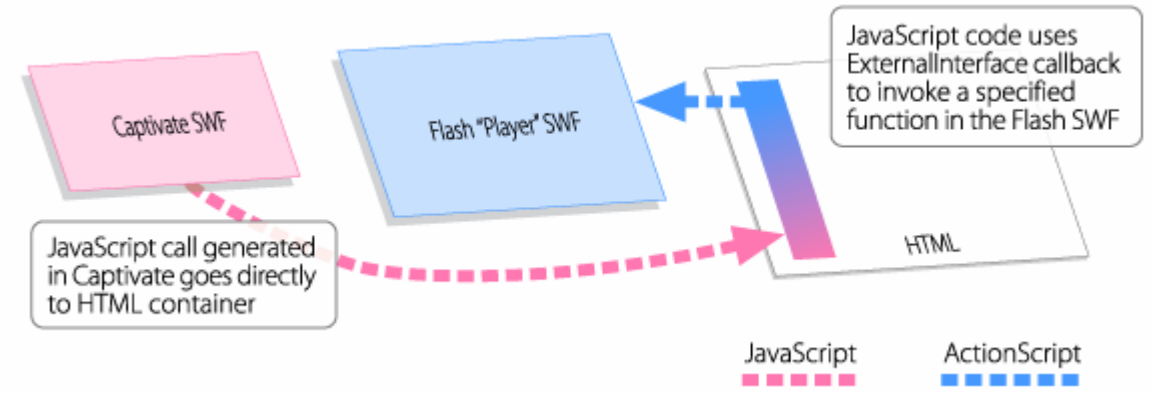
Workaround 2: Use Flash's ExternalInterface class

How it works

Captivate can make JavaScript calls; these JavaScript calls can be used to relay pre-defined instructions to a Flash-based SWF. The relay mechanism is a Flash class named ExternalInterface.

The basic chain of events looks like this:

1. Captivate invokes a JavaScript function located in the HTML…
2. … which in turn invokes a Flash ExternalInterface callback located in the HTML…
3. … which in turn invokes an ActionScript function located in the Flash SWF



For this demonstration, a Captivate SWF will be configured to call a single JavaScript function named `unloadMe()` at the end of the last slide. We will utilize ExternalInterface to prompt the Flash "player" SWF to unload the Captivate SWF.

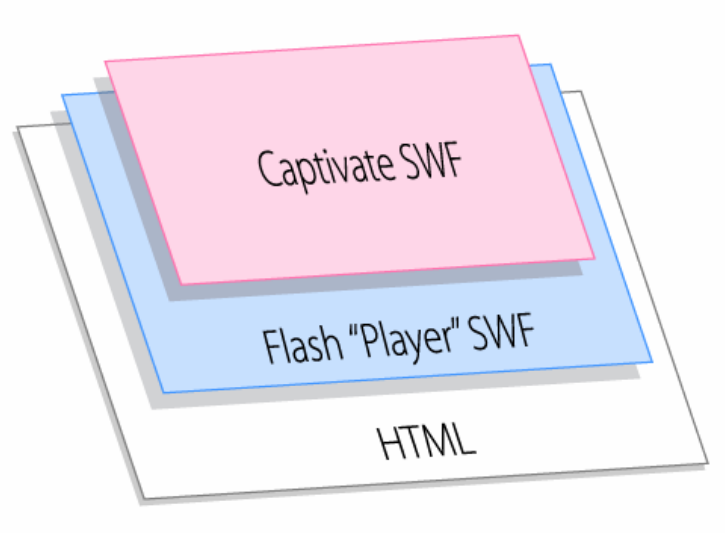
Important notes

- ExternalInterface requires **Flash Player 8** or higher.
- If run locally (from a CD or hard drive), your Flash Player security settings must be edited to allow files in that particular directory to communicate with each other. **The demonstration will not work locally if these settings are not enabled**. Here's how Adobe explains it:

> Security note: *For local content running in a browser, calls to the ExternalInterface.addCallback() method will only work if the SWF file and the containing web page are in the local-trusted security sandbox.*

Files required

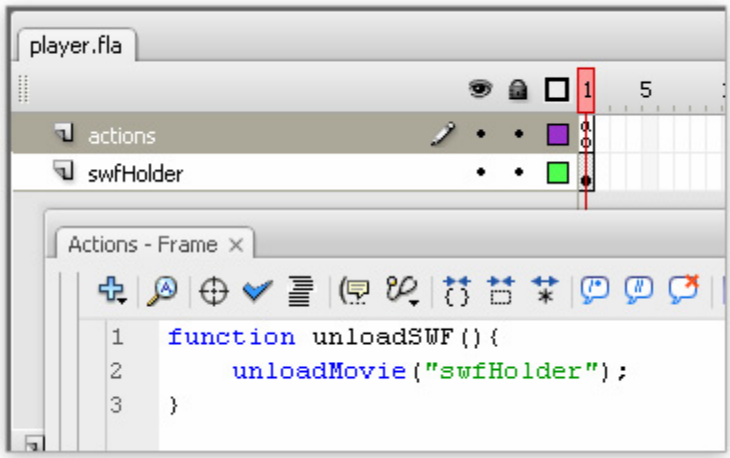
1. HTML file.
2. Flash "Player" FLA.
3. Captivate file.



Instructions

1. In player.fla, create the function you'd like the Captivate SWF to invoke

Open player.fla and write the function that the Captivate SWF will invoke. The function should be contained in the first frame of the FLA's main timeline.



For this demonstration, we'll write a function named "unloadSWF" that will unload the Captivate SWF from the player.

```
function unloadSWF(){
   unloadMovie("swfHolder");
}
```

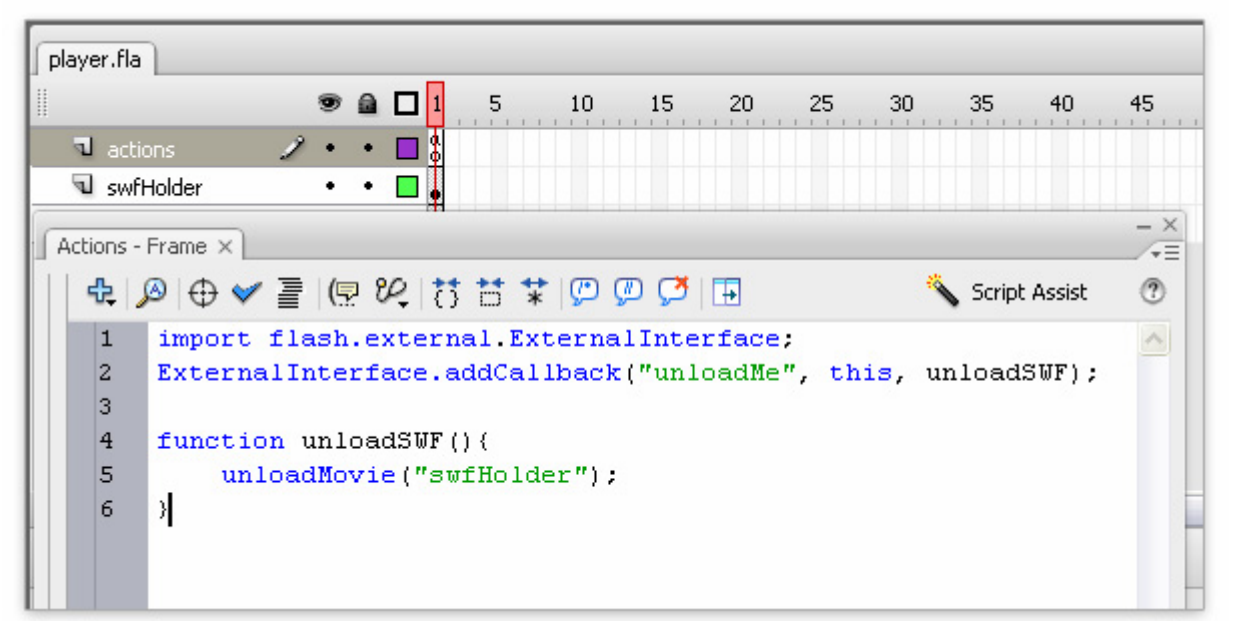
2. Add the ExternalInterface code to player.fla

Add this line to the top of the frame script in player.fla's first frame:

```
import flash.external.ExternalInterface;
```

This tells Flash that we will be using its ExternalInterface class, ensuring our ExternalInterface callback will be properly supported. Now add the callback:

```
import flash.external.ExternalInterface;
ExternalInterface.addCallback("unloadMe", this, unloadSWF);
```



The two most important parameters in this line are also the two most confusing. The first parameter, *"unloadMe"*, is the name of the *JavaScript* function that will be called in the HTML. It's a string and must be surrounded by quotes. The last parameter, *unloadSWF*, is the name of the *ActionScript* function that will be called. It's an actual function call and therefore shouldn't be surrounded by quotes.

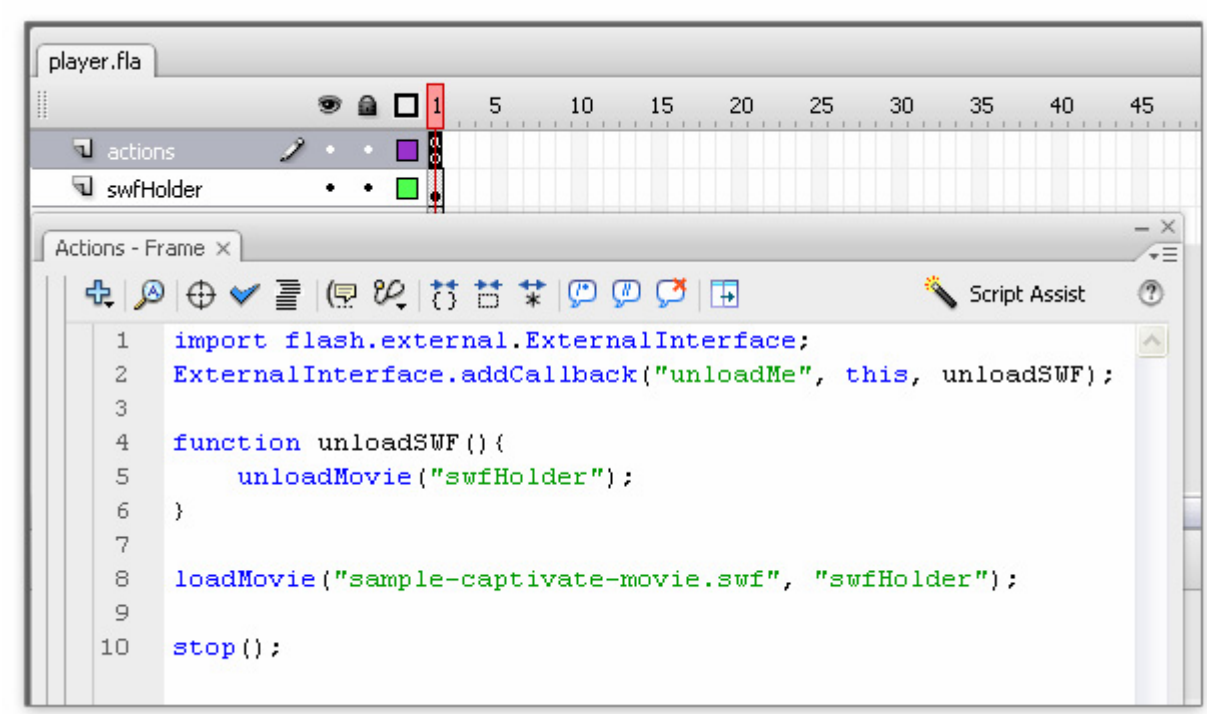
In case you're wondering, the middle parameter, *this*, is simply stating that the callback belongs to *this movieclip* (in our case, it's the main timeline). You shouldn't need to change *this* to anything else.

3. Add the loadMovie code

While we still have player.fla open, let's add the *loadMovie* code we'll need to automatically load our Captivate SWF for us. We also need to add the stop() command to prevent the player.swf from looping.

```
loadMovie("sample-captivate-movie.swf", "swfHolder");

stop();
```



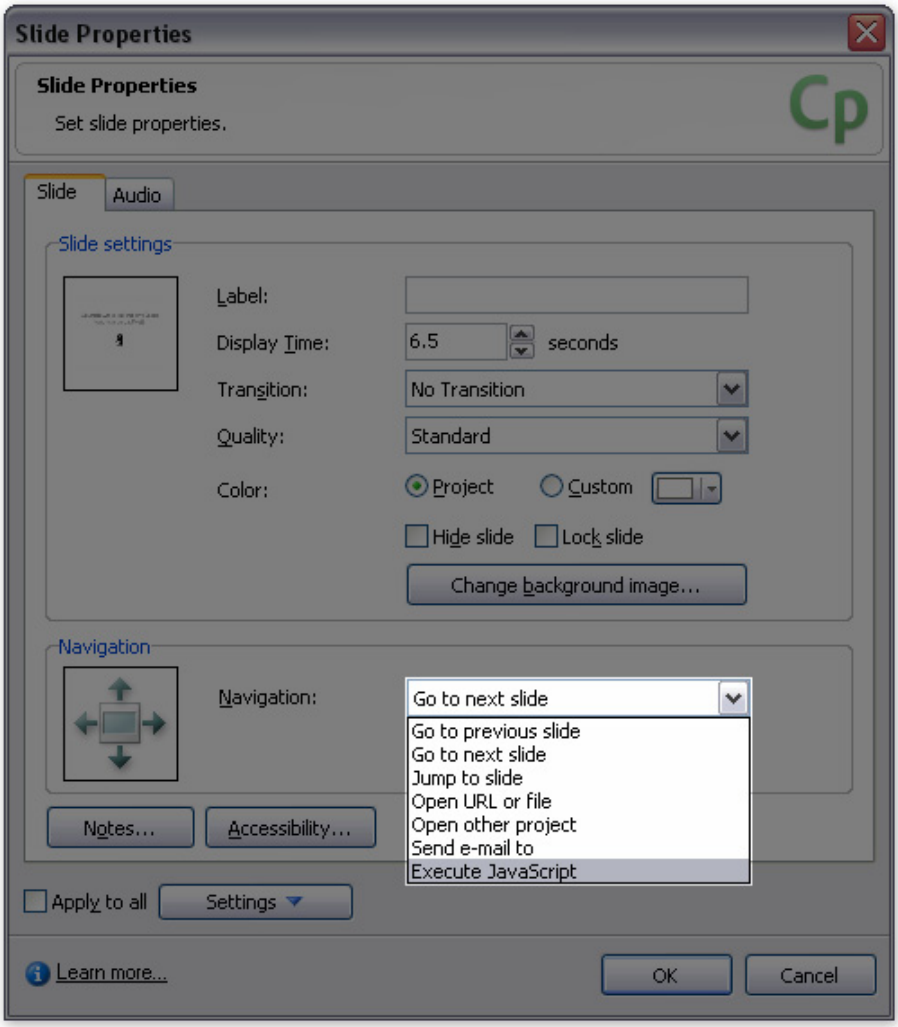
4. Publish player.fla

Publish the FLA file. Turn off the 'publish HTML' option, since we won't be using any of the HTML generated for this SWF. Also make sure the SWF is published as Flash Player 8 or greater (ActionScript 2.0) or else the ExternalInterface code will fail.

Note: Even though this demonstration uses ActionScript 2.0, you can also use ActionScript 3.0; you'll just need to adapt the ExternalInterface code to its newer ActionScript 3.0 syntax.

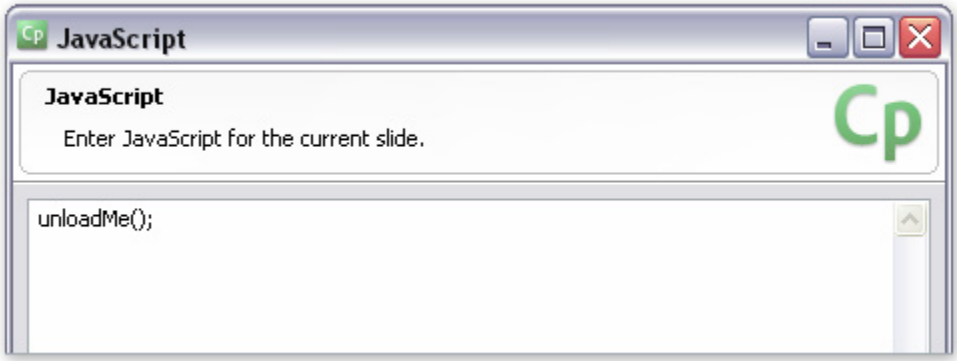
5. Add the JavaScript call to the Captivate file

Open the Captivate file *sample-captivate-movie.cp*. Go to the *Slide Properties* for slide two. Select the *Execute JavaScript* from the *Navigation* drop-down menu.



The *JavaScript* window will appear. Any JavaScript entered here will be executed at the conclusion of this slide. For our demonstration, we only need to add a single function call:

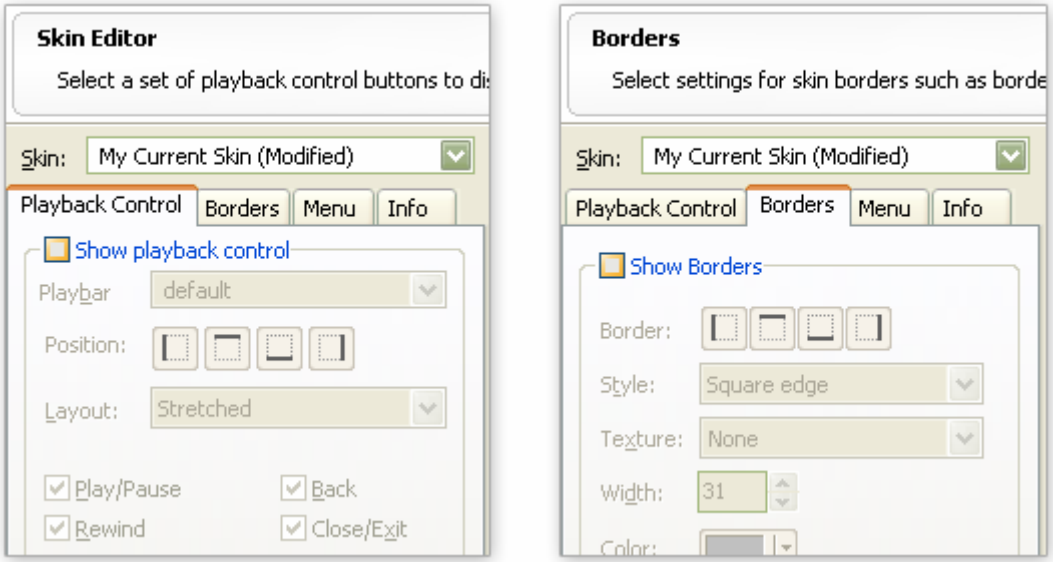
```
unloadMe();
```



Click *OK* to close the *JavaScript* window, then click *OK* to close the *Slide Properties* window. The JavaScript is now a part of your Captivate movie. Save the Captivate file, but don't publish it yet.

6. Turn off skinning

Since this Captivate movie will be loaded into another SWF (our custom player.swf), it's safe to turn off Captivate's skinning feature. Go to *Project* > *Skin* and uncheck both "Show playback control" and "Show borders".



Note: *If you preview the new Captivate SWF in Captivate or anywhere outside of the player SWF, nothing will happen when Captivate executes your JavaScript. This is because the function* `unloadMe()` *does not exist in the Captivate file; the function is stored in the HTML file. It can only be executed if the Captivate SWF has been loaded into the player SWF.*

7. Publish the Captivate file

Publish with the following settings:

- Flash SWF as the output type.
- Uncheck "Export HTML" (you will get a warning, but it's safe to ignore it).
- Output SWF as Flash Player version 8 or higher.

8. Add the necessary JavaScript to HTML file

To have Captivate's `unloadMe()` JavaScript command properly relayed to the player SWF, we need to configure the JavaScript contained in the `<head>` of the HTML file. Remember, the JavaScript in the HTML file is the relay point; the chain of events is:

1. *Captivate invokes a JavaScript function located in the HTML,*
2. *which in turn invokes a Flash ExternalInterface callback located in the HTML,*
3. *which in turn invokes an ActionScript function located in the Flash SWF.*

We'll start by creating an empty function that matches the name of the function we used in Captivate:

```
function unloadMe() {

}
```

Next, we'll add the ExternalInterface callback to the function. For the callback to work, we need JavaScript to identify the player SWF as an object. We can do this by using the very common and widely supported `document.getElementById()` DOM method.

Note: For this demonstration, the player SWF's ID has already been defined as "captivatePlayer" in the SWFObject embedding code.

```
function unloadMe() {

   var player = document.getElementById("captivatePlayer");

}
```

Once the SWF has been identified as an object by JavaScript, we can implement the *"unloadMe"* callback we defined in the FLA file:

```
function unloadMe() {

   var player = document.getElementById("captivatePlayer");
   player.unloadMe();

}
```

That's it for the JavaScript! It's pretty straightforward. If you prefer, you can also use the following shorthand syntax:

```
function unloadMe() {
   document.getElementById("captivatePlayer").unloadMe();
}
```

Don't forget to save the HTML file.

9. Test it!

Now that your files are ready, test them out! Make sure the following files are in the same folder:

- index.html
- player.swf
- sample-captivate-movie.swf

Open index.html in your web browser. The sample Captivate movie should automatically load. Click the *Click here to go to the next slide* button. You should see the countdown, followed by the SWF unloading itself.

Reminder: *If run locally (from a CD or hard drive), your Flash Player security settings* ***must*** *be edited to allow files in that particular directory to communicate with each other.*

Workaround notes

Pros

- Less bulky/tedious than imported SWF solution.
- Easy to quickly to add or edit customs scripts.
- Not restricted to nested SWFs; a Captivate SWF using ExternalInterface can communicate with any properly configured SWF contained in an HTML page.

Cons/limitations

- Requires Flash Player 8 or greater.
- Relies on JavaScript being available in the browser.
- JavaScript calls are bound to Captivate objects and/or events.
    - No multitasking or queuing
    - Can't 'stack' actions on objects such as buttons
- Is difficult to implement in local environments (such as on a hard drive or CD-Rom) due to Flash Player's security restrictions.

When is the best time to use a particular method?

Situation	Imported SWF workaround	ExternalInterface workaround
When supporting Flash Player 7	✓	
When supporting Flash Player 8 or greater	✓	✓
When using custom Flash-based "player" SWF to load external Captivate-based SWFs	✓	✓
When very few ActionScript calls are needed from Captivate	✓	
When many ActionScript calls are needed from Captivate, especially if they are frequently edited		✓
When the ActionScript call needs to be invoked mid-slide or without being attached to a user event, such as a button click	✓	
If the ActionScript is only invoked at the completion of the Captivate SWF	✓	✓
When user action in Captivate (such as clicking button) requires Captivate event such as "jump to slide" or "continue"	✓	
When custom ActionScript calls also need to work with HTML page's JavaScript		✓

Additional examples

Additional examples illustrating the techniques covered in this session are available for free at http://pipwerks.com/lab/captivate.

About the author

Philip Hutchison is an Instructional Technologist and e-Learning Developer who develops courses and course delivery mechanisms for UCSF Medical Center – one of the nation's Top Ten hospitals. Philip's work spans many fields, including curriculum development, Web development, Flash development, print production and graphic arts, audio production, and LMS administration. He holds an M.A. in Education (Instructional Technologies) from San Francisco State University. Prior to his life in e-Learning, he split his time between professional print production and non-profit radio (college and NPR).

This presentation was prepared for the eLearning Guild's DevLearn 2007 conference, November 2007.