FLASH CS 4|5
ACTIONSCRIPT

First, we should look at the history of Flash in a bit more depth. Knowing where it started as compared to where it is now is important in understanding the reasons behind implementing a coding language into the software.

As you may remember, Jonathan Gay developed a small program in the mid-90's he called Future Splash Animator. That program had a timeline and a stage, and some limited drawing tools. It was, essentially, a program for creating animated GIFs, which were just starting to make a limited appearance on web sites—envelopes with wings, indicating email, for instance. It was based on Java, and initially, neither Adobe nor Fractal Design (now defunct, but originally the creator of Painter, an early competitor of Photoshop) was interested in it. Weird but true.

A year later in 1996, however, Microsoft learned of Future Splash and used it to create their first web presence, MSN. Disney also started using it for their online work. Macromedia heard about it and offered to purchase it. They renamed it Macromedia Flash.

It wasn't until version 5, released in August, 2004, that Flash was able to do much except basic animation. Flash 5, however included ActionScript 1. To make something interactive, a developer could select the element on the stage, open the coding window, and add code to "play," "stop," or "getURL," for instance. There were options to do a lot more, of course, yet many of the things that developers wanted to do weren't possible with that version, like generating animation using code, as versus using frames on the timeline.

In 2004, Macromedia released Flash MX 2004 and Flash MX Professional 2004 (version 7). With that release, ActionScript 2.0 was introduced, offering programmers the opportunity to create animation through code, as well as reference symbol instances on the stage to provide interactivity.

Obviously, Macromedia was adding a lot more to each release of Flash than just improvements in the coding options. Components, a video compression system (codec), Behaviors, and other technical aspects were adopted along the way. (There has always been a bit of a tug of war between artists using Flash for animation, and application developers using it to create games, web sites, content for mobile devices, and stand alone applications.)

Actionscript 3.0 was introduced with Flash CS3, after Adobe purchased Macromedia, providing stronger "object oriented programming" (OOP) capabilities and code reusability for complex applications.

ACTIONSCRIPT BASICS

The largest difference between ActionScripts 1 and 2, and 3, is whether you can select a button or a movie clip on the stage and add code to it. In both AS2 and AS3, you must give a symbol instance an instance name and then place code on a keyframe in the timeline that refers to that name. That doesn't sound as though ActionScript 2 and ActionScript 3 are all that different, however, the code organization in AS3 has changed significantly.

Compare these ActionScript1 and ActionScript2 and ActionScript3 code snippets:

Let's dissect each example. I know; boring. But the sooner you can grasp the basic concepts, the sooner you will be able to trouble shoot problems, if not even write your own code from scratch!

ACTIONSCRIPT 1

To make a movie clip symbol interactive using ActionScript 1, you would select the symbol on the stage and add this code to the Actions pane:

```
on(press){
        startDrag(this, false, 0, 0, 500, 400);
}
on(release){
        stopDrag();
}
```

ACTIONSCRIPT 2

To make a movie clip symbol interactive using ActionScript 2, with an instance name of "symbol_mc" you would select a keyframe in the timeline and add this code to the Actions pane:

```
symbol_mc.onPress = function() {
        startDrag(this, false, 0, 0, 500, 400);
}
symbol_mc.onRelease = function() {
        stopDrag();
}
```

ACTIONSCRIPT 3

Using ActionScript 3, you would place a movie clip symbol instance on the stage and give it an instance name (symbol_mc), and then add this code in a keyframe on the main timeline:

```
function dragThis(event:MouseEvent):void {
        symbol_mc.startDrag();
}

function dropThis(event:MouseEvent):void {
        symbol_mc.stopDrag();
}

symbol_mc.buttonMode = true;

symbol_mc.addEventListener(MouseEvent.MOUSE_
DOWN, dragThis);

symbol_mc.addEventListener(MouseEvent.MOUSE_
UP, dropThis);
```

In the first, we select the symbol on the stage, and tell Flash what the user has to do (click and hold the mouse button = on(press)).

Then we tell Flash what to do as long as the mouse button is being held (start dragging = startDrag) the movie clip symbol (this).

Note that what you want Flash to do is within { } curly braces.

The other instructions within the parentheses tell Flash that you don't want the user's cursor to snap to the center of the symbol (false), and indicate the margins within which the symbol can be dragged around the stage in pixels (left, top, right, and bottom)).

In the second example, using ActionScript 2, we name the movie clip instance on the stage by selecting it and then naming it in the Properties pane.

Then in a new layer (labeled actions, of course), we add a keyframe where we want this behavior to occur and in the Actions pane (while the frame is highlighted, NOT the symbol instance) we tell Flash specifically which symbol instance (symbol_mc) we want to work on.

Immediately following, we tell Flash what the user has to do (onPress). (Note the difference between the on(press) in AS1, and the onPress syntax in AS2.)

These two pieces, the target, and the behavior, are strung together, separated by a dot (.). This is called dot syntax and it's an important concept to grasp. More on that in a bit.

Finally, we define this as a function. The following code, within the { }, tells flash what to do when the user presses the named instance—it's exactly the same as the code we used in the first example.

Why is this better than ActionScript 1? It keeps the code on the timeline, so it's easier to locate a problem and fix it (as versus trying to find the problem element on the stage as well as checking frames). It also allows the use of dot syntax, or creating a target path, so Flash knows where inside a movie clip symbol to go, for instance. It's cleaner and more efficient code.

In the ActionScript 3 example, we name the symbol and refer to it within a frame action—much like AS2. The difference is in the way the code is written.

In AS3, we first define a function—code that tells Flash what to do when that function is invoked. Note within that function the code indicates Flash should run the code when the user does something with the mouse (MouseEvent). The code within the function definition is the same used in the previous two examples (startDrag).

In this case, we need two functions – one to start dragging the clip, and one to stop dragging it.

The next piece of AS3 code tells Flash to treat the movie clip symbol like a button—that is, to change the cursor from an arrow to a hand when the user mouses over it.

The last piece is called an event listener. This is where we tell Flash what mouse event has to occur (MOUSE_DOWN) and what function to run when that happens (dragThis). Likewise, we add a second event listener to drop the movie clip.

VOCABULARY

As you've just seen, you can string several things together, separating them using dots (.) The elements and AS code you use tells Flash a lot when using dot syntax, from where you are within a Flash file, to the symbol instance name, to what Flash needs to do when something happens. All of these elements within ActionScript have names.

A **keyword** is any word that Flash has "reserved" to use as ActionScript. You've probably noticed these words turn blue (by default) when you type them in the Actions pane.

Variables are containers for information. The information can be text (a string, which is always surrounded by quotes), a number, or a Boolean (true or false). If and when you get into writing a lot of code, you'll learn that when you define a variable, you'll want to use strict data typing; that is, define what kind of information can be stored in the variable you are defining.

There are also **operators**, or math symbols that allow Flash to compare elements, add to or subtract from, multiply, and so on.

Events refer to what a user might do while interacting with a published Flash move (press a button, for instance) and there are **event handlers** that respond to those user actions. In AS3, event handlers are called **event listeners**.

Functions are pieces of code that tell Flash what to do once the user does something, or when a specific event is encountered in the timeline. There are dozens of built in functions, like gotoAndPlay, stopAllSounds, and others. Capitalization (and spelling!) when specifying a function is key to making it work or not.

Properties are aspects of elements or objects within a Flash document, like the size and position of something, that you can change using ActionScript. In AS2, properties are always preceded by an underscore; _alpha, or _xscale, for example. Two of the more important properties are _root and _parent, which helps Flash figure out where things are, and how to get there.

_root refers to the main timeline

_parent refers to the next level "up" within a symbol

So if you have a button inside a movie clip the button's "parent" would be the movie clip, and the movie clip's "parent" would be the main timeline.

In ActionScript 3, the underscores are not needed when specifying a property.

ACTIONSCRIPT WORDS

When adding code to the Actions pane, you will note that some words turn different colors. This indicates that the word(s) are "reserved" AS words, and should never be used for naming instances or frames, or for variable or function names.

By the same token, if you are trying to type an AS word (like getURL, for instance) and it doesn't turn blue, then you know there's a typo somewhere. Check for your mistake immediately.

INSTANCE NAMES

There are several conventions you want to follow when working in Flash, and one of those is how things are named. Here are the basic rules:

Do not use any punctuation marks or special characters.

Start every name with a lower case letter (NOT a capital).

Never start an instance name with a number.

Use a descriptive name so you can remember it, and know what it is (after all, you will be referring to it in your code).

Use camelCase if you need two words for a label. camelCase is the practice of capitalizing the second word.

Or, use an _ (underscore) to separate words.

Avoid using ActionScript words (like stop, play, and so on).

Add _mc or _btn to the end of the name. This helps Flash recognize what kind of symbol it is.

These same rules apply when creating frame labels.

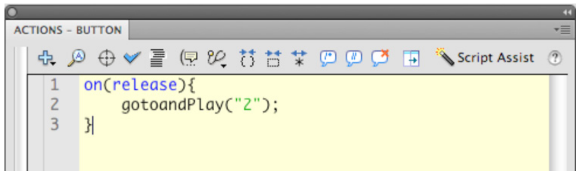
ERROR MESSAGES (AS2)

If you have made an error in your code, you may see a description of it in the output or compiler panel when you test the movie. Learning how to interpret those errors will go a long way in making your work in Flash less frustrating.

There are times when you won't get an error message, and your file will still not work. Learning what to look for in your code is also a huge help in getting things functioning.

Let's take a look at some of the more common mistakes.

This code will not generate an error message, but there are problems with it, so it will not play when you test the movie.

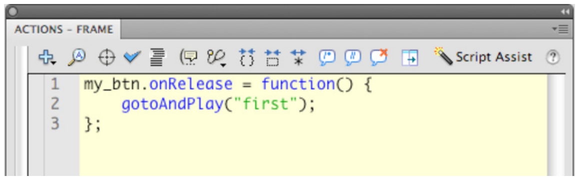


```
on(release){
    gotoandPlay("2");
}
```

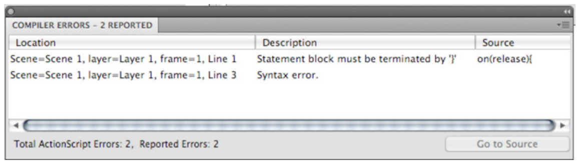
The first line, on(release) is fine, and it also contains the opening curly brace that defines the function.

The problem is a simple typo; gotoandPlay. As you may have already encountered, ActionScript is case sensitive, especially when it comes to reserved words or phrases like this one. The fix for this is to replace "gotoandPlay" with "gotoAndPlay" —a capital A for and, and a capital P for play. You'll know you've typed it correctly when it turns blue.

Here is an example of an error code that Flash will point out using the Compiler Errors panel.



```
my_btn.onRelease = function() {
    gotoAndPlay("first");
};
```



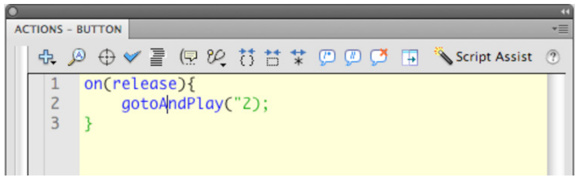
Location	Description	Source
Scene=Scene 1, layer=Layer 1, frame=1, Line 1	Statement block must be terminated by '}'	on(release){
Scene=Scene 1, layer=Layer 1, frame=1, Line 3	Syntax error.	

Total ActionScript Errors: 2, Reported Errors: 2

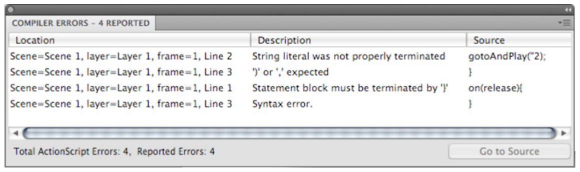
In analyzing the three areas of the Compiler Errors panel, you'll see that Flash tells you first why the error is occuring; there is something missing in this chunk of code.

The second line tells you where the missing curly brace needs to go. Very nice!

This code has a problem that will generate a Compiler Error.



```
1  on(release){
2       gotoAndPlay("2);
3  }
```



Location	Description	Source
Scene=Scene 1, layer=Layer 1, frame=1, Line 2	String literal was not properly terminated	gotoAndPlay("2);
Scene=Scene 1, layer=Layer 1, frame=1, Line 3	')' or ',' expected	}
Scene=Scene 1, layer=Layer 1, frame=1, Line 1	Statement block must be terminated by '}'	on(release){
Scene=Scene 1, layer=Layer 1, frame=1, Line 3	Syntax error.	}

Total ActionScript Errors: 4, Reported Errors: 4

When you are referring to a string in ActionScript, like a named frame (above), it must be in quotes. You can tell what the problem is, as all the code after the number 2 remains green—a good indication the quote wasn't closed.

In looking at the Compiler Error pane, you can see from the very first line what the issue is, and where within the Actions pane it exists: line 2 has a problem, and if you're on the ball, you'll see immediately that the string isn't closed.

The error on line two is a direct result of the first error; Flash is not seeing the closing curly brace as a closing curly brace at all, but part of the string:

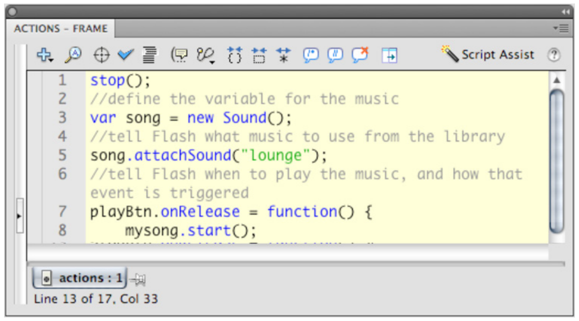
```
"2);{
```

Adding the closing quote after the number 2 will eliminate this message automatically.

The next two lines are reiterations of the problem, just phrased in a different way; Flash is still looking for that closing curly brace. Again, once you add the closing quote, this message will disappear.

Important note: Flash does not like curly quotes, or typographer's quotes. It does like inch marks, though. If you ever get an error message about the type of quote mark (usually because you pasted something from a Word doc), just delete them and retype them within the Actions pane in Flash.

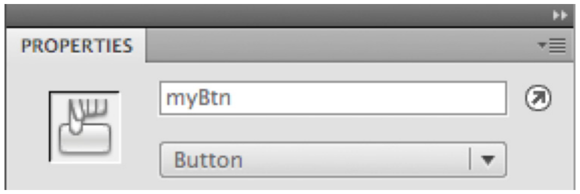
Another place where you can run into problems is a discrepancy between an instance name in the Properties panel and in the code.



```
stop();
//define the variable for the music
var song = new Sound();
//tell Flash what music to use from the library
song.attachSound("lounge");
//tell Flash when to play the music, and how that
event is triggered
playBtn.onRelease = function() {
    mysong.start();
```

Line 13 of 17, Col 33

This code looks great, and in fact it's correctly written, but it won't work because the instance name used in the code (my_btn) is not the same one used for the instance name in the Properties panel (below). If you know the code is right, double-check instance names. In fact, it's smart to keep a scratch pad nearby when developing a Flash file so you can make note of just those sorts of things, as well as unusual color hex numbers (for ones that are not included in the Swatches panel) and so on.



Always check instance names in the code against what you actually called something in the Properties panel.

Write down any variables you might define, too.

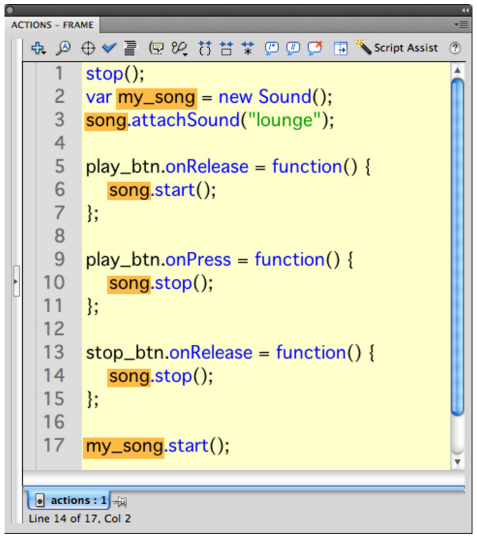
Aside from learning how to decode the error messages, another way to double-check problem code is to read it from bottom to top. That way, you'll be reading each word individually and should be able to spot typos faster.

The next step is to count the opening curly braces and make sure you have the same number of closing curly braces.

Watch out for green type where it doesn't belong; that usually signifies a forgotten closing quote.

Watch out for type that should be blue and isn't.

The huge advantage of knowing some of the more common mistakes, and learning how to correct them, is that you'll be less likely to make them in future projects, and if you do, spot and fix them faster.



```
1   stop();
2   var my_song = new Sound();
3   song.attachSound("lounge");
4
5   play_btn.onRelease = function() {
6       song.start();
7   };
8
9   play_btn.onPress = function() {
10      song.stop();
11  };
12
13  stop_btn.onRelease = function() {
14      song.stop();
15  };
16
17  my_song.start();
```

When defining a variable, make sure you use the same name and capitalization when you refer to it again within the code; the only thing this code will do is start the music playing right away (in line 17) as it's the only line that uses the correct variable name: my_song.

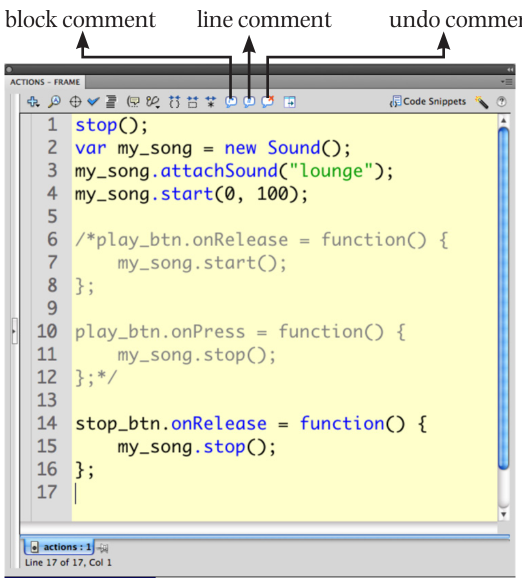
COMMENTS

You may have downloaded or watched tutorials online where some of the code is gray. These are comments created by the developer to alert you what's happening in the code.

Single line comments are defined with // at the start. These are generally used to tell someone who's using the code how it works. Because it's gray, Flash will ignore it.

Block comments are used to literally "block" code from running. It's a great way to test different ways to write code; you can try three different approaches, and block two at a time to run just one as a test.

Block comments begin with /* and end with */. They are used to prevent Flash from running the code between the /* and */ symbols. Block comments are a great way to test various bits of code to see which works, or which works better. You can also add multiple line comments using block code.



```
stop();
var my_song = new Sound();
my_song.attachSound("lounge");
my_song.start(0, 100);

/*play_btn.onRelease = function() {
    my_song.start();
};

play_btn.onPress = function() {
    my_song.stop();
};*/

stop_btn.onRelease = function() {
    my_song.stop();
};
```

Use block commments to test which portion of your code is unnecessary, or not functioning properly.

ActionScript 3 has it's own set of possible errors. The most common is that a function you've written just won't execute, or the file plays straight through without stopping where it should. As with AS2, check your spelling, capitalization, opening and closing brackets, opening and closing quotes, variable and function names, and so on.

Make sure you've assigned the correct function to the appropriate listener.

Make sure you're actually working on an AS3 file! You'd be surprised how often that can happen. Use File > Publish Settings and check the Flash tab to see, or change, the AS version.

Obviously we could spend a whole lot of time looking at the common and more esoteric errors that can occur within a Flash project. The key to getting started isn't about memorizing code as much as it's about knowing where to look by understanding the error messages in the Compiler Errors panel, and proofreading your code carefully. Once you're comfortable with that, writing your own code will get a lot easier.

The Flash Help menu is also a terrific resource ;-)

YOUR TURN

The game you are to develop is a combination of those flip books where you choose different eyes, noses, and mouths to make silly faces. Do a web search for “Flip-A-Face Book” to see an example of this in paper. In addition, you’ll make those features something the user can drag around to place where s/he chooses.

This is not about faces though! The topics for making a game like this are endless, from different color schemes or accessories for a room, to objects on a desk or workbench, or food on a plate. It’s a great opportunity to great something educational as well.

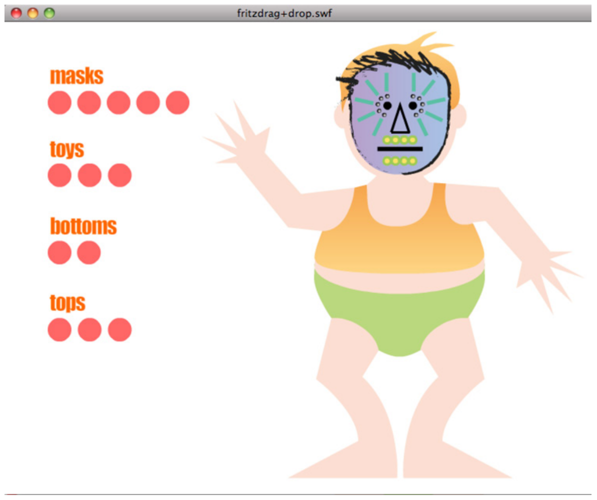
Use your imagination, draw everything in either Flash or Illustrator, and have fun! You need at least three elements that change (i.e. eyes, nose, mouth), and at least three variations of each (i.e. blue eyes, glasses, pirate patch). The variations will be placed in a movie clip (so you’ll have at least three movie clips), and buttons on the stage will be coded to play each variation within each movie clip (for a minimum of 9 buttons).

You could even go so far as to animate some of your elements. Think about it!

Use ActionScript 2 only, by naming frames and symbol instances and referring to them in code placed on the first frame of your file, rather than placing code directly on a symbol.

Publish your file, and edit the HTML document to center the SWF on the page, add an appropriate page title, and your copyright information.

Upload the HTML and SWF files to your web space; update your Flash index.html links, and upload the revised SWF for your homepage. Post your URL to Documents.


fritzdrag+drop.swf
masks
toys
bottoms
tops