

Tutorial – Using Sound in ActionScript 3

Flash Tutorial March 24th, 2008

This tutorial is Part 1 of a series of tutorials on creating a music player.

Anyways, in an era where melodic snobbishness is so prevalent, everyone with a website (or a MySpace account) wants to play their favorites for the whole world to hear, so if you're a Flash designer/developer, it might do you some good to learn how to import, play, and modify sound in ActionScript. In this tutorial, I'm going to introduce you to using sound in ActionScript 3, and in later posts, we'll start to get a little more in depth with the different things we can do with sound.

Note: If you're easily bored with tutorials and just want to look at some ActionScript, be sure to check out my recent post on an ActionScript 3 / XML Music Player I created. This post contains a link to an FLA that might have some of the ActionScript you're looking for.

In this tutorial, we're not going to talk about importing a sound into Flash. Rather, we're going to discuss accessing external sound files using ActionScript without ever actually importing them into the Flash authoring environment. 1. Save your Flash file. Before we do anything else, we need to save our Flash file, because we'll soon be pointing to an external sound file, and in order to point to the right location for the sound file, Flash will need to know where the FLA file will be saved as well. In this example, I'll be saving my FLA file in the same directory as my mp3 music file. 2. Create a Sound Object. So far, all we're trying to do is to get a sound file to play, so we don't really need to worry about putting anything on the stage. Instead, simply select "Layer 1" on your timeline, rename the layer "Actions", click on frame 1 of this layer, and hit Option+F9 (or just F9 on PC) to open up your Actions panel for this frame. The code for creating a new sound object is simple:



To create a sound object, we first created a variable called "music" and we set this variable equal to a new Sound object. Inside the parentheses for this new Sound object, we placed a new URLRequest object, which allows us to access the URL of an external file, as we see in line 1. This external file is the mp3 file we will be playing. In line 2, we point to our new sound object and tell it to play. And with only two lines of code, we're already playing music in our Flash file!

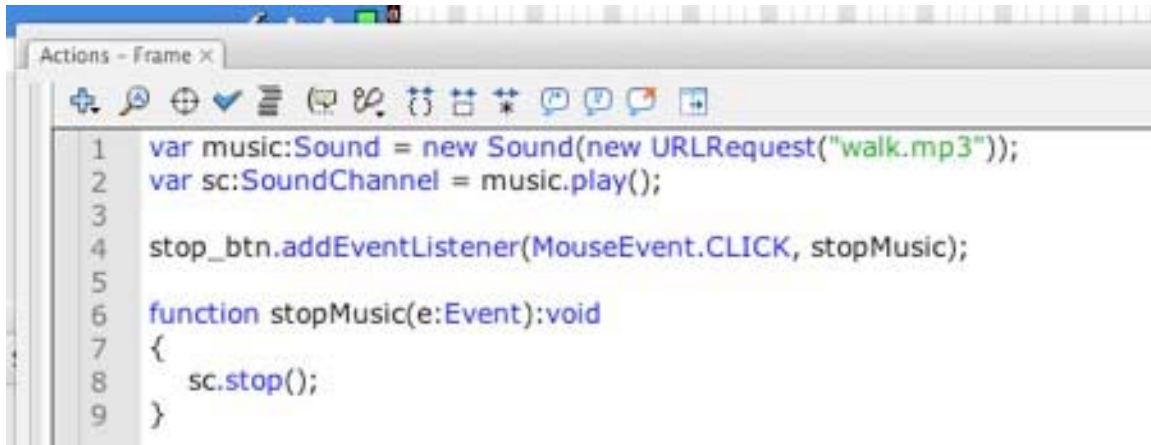
Phase Two: Shut that Racket Off!

If I were you, I wouldn't even play the music by default unless you're building a website for a band, or some other website where the user might expect to hear default music. But one thing's for sure--if you set up your music to play by default, you WILL lose some traffic to your site. Even if you provide controls to turn the music off, many people aren't going to take the time to look for the stop button. They're just going to leave. Create a stop button. It doesn't matter what the stop

button looks like, as long as it looks like a stop button. So create a new layer, draw your graphics, and then convert to a button symbol by hitting F8. Here's what my stop button looks like (in true, glossified, Web Two-Dot-Oh fashion):

Stop button

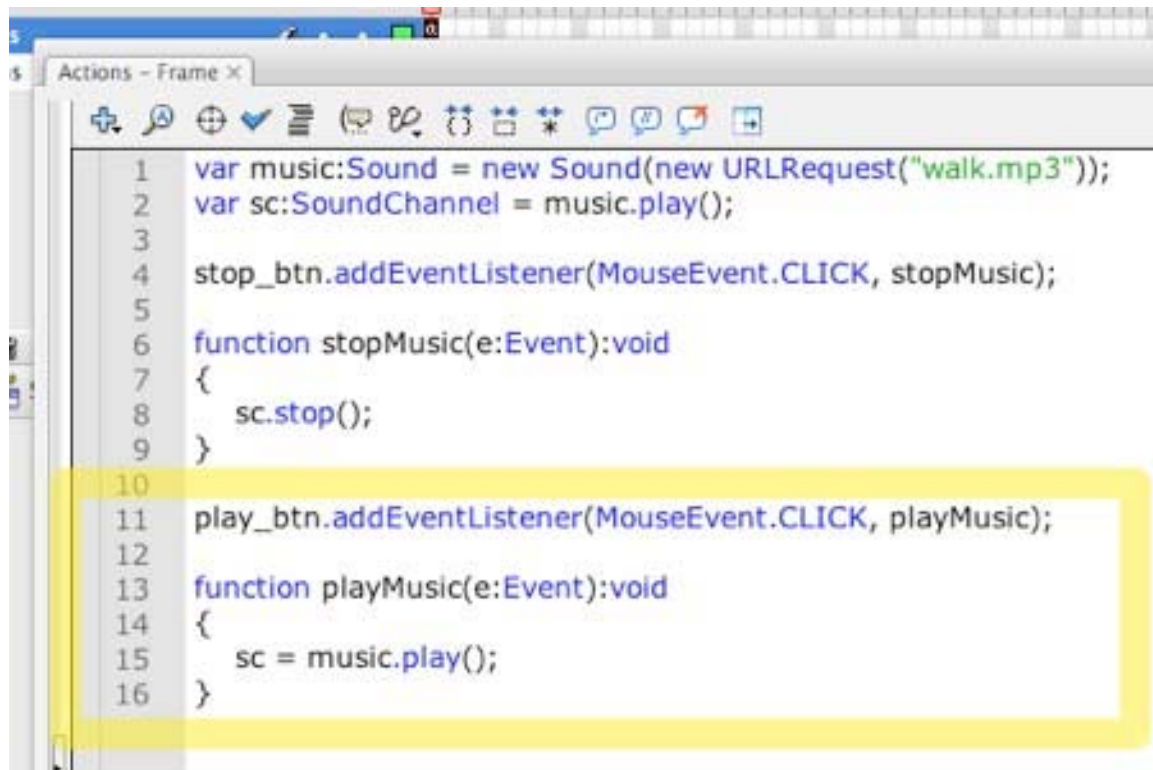
I've also given my button an Instance Name of `stop_btn`. Here's the code that causes our stop button to work (hang on to your seats, this one's complicated):



The first thing you should notice is that line 2 has changed a little. Unlike with ActionScript 2, in AS3, you can't control the playback, volume, etc. of a sound object using the Sound class. Instead, you have to assign the playback of your sound to a new instance of the SoundChannel class. And then, using your new SoundChannel object (which we simply called "sc"), you can tell your sound how to behave. In line 4, we created the typical event listener for our button so that when we CLICK on our stop button, it will trigger the "stopMusic" function, which contains the code for stopping the playback of our music. On line 8, inside the aforementioned "stopMusic" function, notice that the "stop()" method is attached to the SoundChannel object we created, which we called "sc." And now, if we test our file, we'll see that we have the ability to stop our sound.

So, How About a 'Play' Button?

Now that we've stopped the sound, how do start it back up again? EASY! Create your play button (I'm giving mine an Instance Name of "play_btn") and add the following code to your Actions layer:



```
1  var music:Sound = new Sound(new URLRequest("walk.mp3"));
2  var sc:SoundChannel = music.play();
3
4  stop_btn.addEventListener(MouseEvent.CLICK, stopMusic);
5
6  function stopMusic(e:Event):void
7  {
8      sc.stop();
9  }
10
11  play_btn.addEventListener(MouseEvent.CLICK, playMusic);
12
13  function playMusic(e:Event):void
14  {
15      sc = music.play();
16  }
```

At least that's the code you'd THINK you might add! And if your user presses the right buttons at the right time, everything will be dandy! But try testing your movie and pressing the play button after the movie has already started playing, and you'll discover that you've got the same sound playing on top of itself. And the only thing worse than The Carpenters playing on someone's website is The Carpenters playing on top of The Carpenters. So before we tell our music to start playing again, we need to make sure it isn't already playing. And we're going to do this by creating a variable to keep track of our status. Let's call our variable "isPlaying" and set it to a data type of Boolean (true/false). Here's what your code will look like:

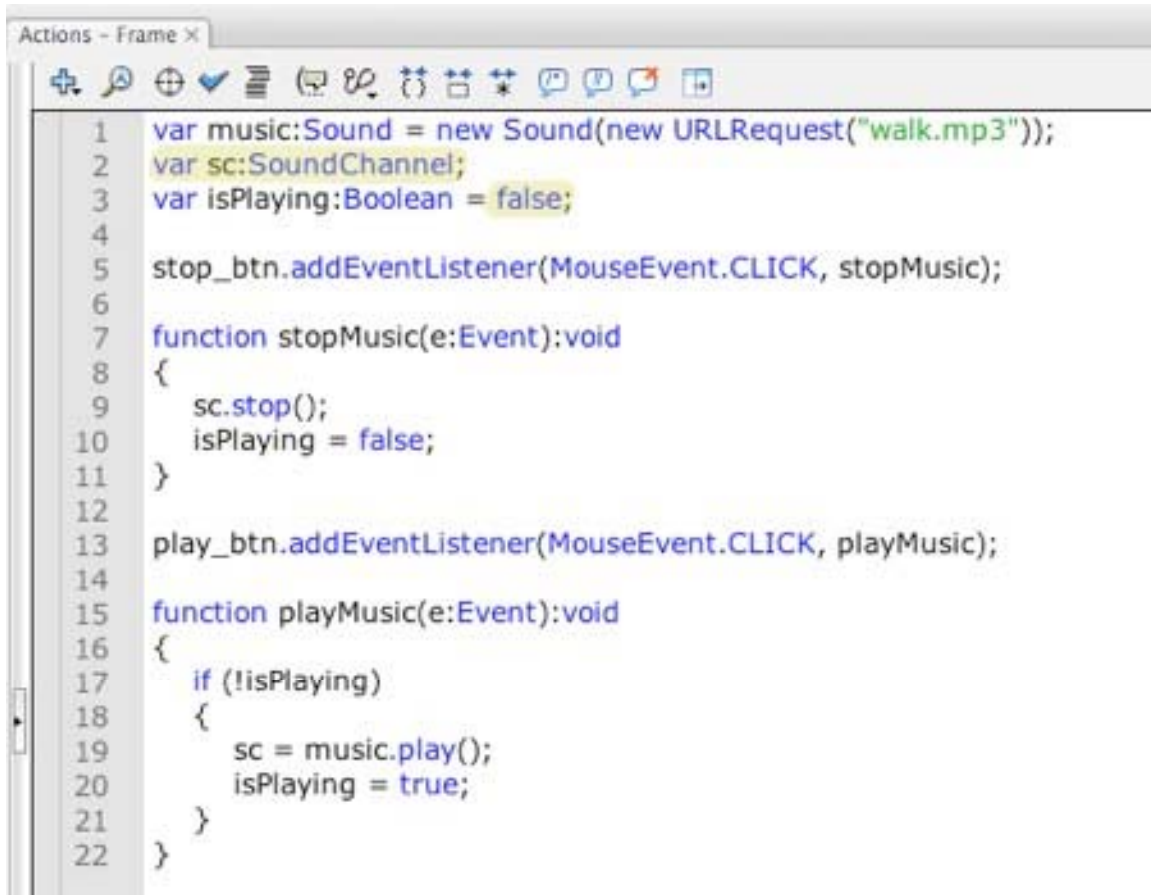
A screenshot of an IDE window titled "Actions - Frame x". The window contains a toolbar with various icons for editing and testing. Below the toolbar, there is a list of 22 lines of ActionScript code. The code defines a Sound object, a SoundChannel, and a Boolean variable 'isPlaying'. It also defines two event listener functions: 'stopMusic' and 'playMusic'. The 'stopMusic' function sets 'isPlaying' to false and stops the sound. The 'playMusic' function checks if 'isPlaying' is false; if so, it plays the sound and sets 'isPlaying' to true. The code is as follows:

```
1 var music:Sound = new Sound(new URLRequest("walk.mp3"));
2 var sc:SoundChannel = music.play();
3 var isPlaying:Boolean = true;
4
5 stop_btn.addEventListener(MouseEvent.CLICK, stopMusic);
6
7 function stopMusic(e:Event):void
8 {
9     sc.stop();
10    isPlaying = false;
11 }
12
13 play_btn.addEventListener(MouseEvent.CLICK, playMusic);
14
15 function playMusic(e:Event):void
16 {
17     if (!isPlaying)
18     {
19         sc = music.play();
20         isPlaying = true;
21     }
22 }
```

On line 3, we created our variable and set it equal to "true" since our music is playing by default. Then, inside the stopMusic function, we set our variable to false. Inside the playMusic function, instead of just allowing the music to start playing any time we hit the play button, we included an "if" statement that first checks to see if the music is already playing, and if it's NOT playing already (the exclamation point basically means "not"), then we allow it to play, and we set our "isPlaying" variable back to true. Now our music player functions beautifully!

One Last Thing

Remember earlier, when I mentioned that it's generally a bad idea to have your music playing by default when the page loads? Yeah, well I meant it! So let's talk about how we can set this up. Actually, you've probably already figured this part out for yourself. Remember that "music.play()" statement in line 2? Get rid of it! Wait until the user actually clicks on the play button before you start playing the music. Of course that also means you need to set the "isPlaying" variable to "false" in line 3. Here's your final code:



```
1 var music:Sound = new Sound(new URLRequest("walk.mp3"));
2 var sc:SoundChannel;
3 var isPlaying:Boolean = false;
4
5 stop_btn.addEventListener(MouseEvent.CLICK, stopMusic);
6
7 function stopMusic(e:Event):void
8 {
9     sc.stop();
10    isPlaying = false;
11 }
12
13 play_btn.addEventListener(MouseEvent.CLICK, playMusic);
14
15 function playMusic(e:Event):void
16 {
17     if (!isPlaying)
18     {
19         sc = music.play();
20         isPlaying = true;
21     }
22 }
```

Notice that in line 2, we still declared our "sc" variable and strict-typed it to SoundChannel, even though we didn't set it equal to anything. The reason for this is that if we try to declare our "sc" variable inside the "playMusic" function, then that variable won't be accessible outside of the function. By declaring the variable outside of the function, we're ensuring that it will be available anywhere within our code. Anyways, those are the basics of creating a simple music player. As time permits, I'll be adding more tutorials that explain how to create a "pause" button and a volume slider. Until then, keep practicing!

* Part 2 - Creating a Volume Slider

In our last tutorial, we talked about how to create a simple music player using ActionScript 3 with a play button and a stop button. In this tutorial, we'll take our music player one step further as we discuss how to add a volume slider to our Flash file. If you want to follow along, download the source file from the previous tutorial. This is the file we'll be starting with.

1. Draw Your Volume Slider. In this case, I'm starting with a simple black line, 2 pixels thick.

Volume Slider

With this black line selected, I'm going to hit F8 to convert it to a movie clip symbol. Then I'll give the symbol an instance name of "volume_mc" and double-click on the

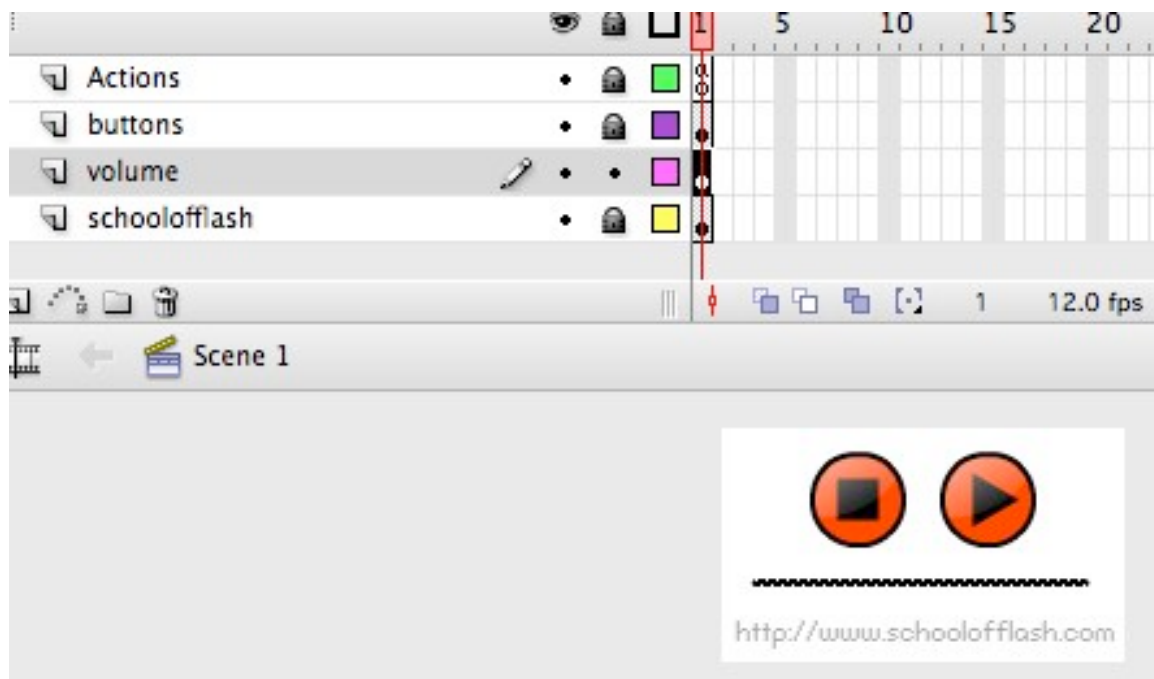
symbol to go into its timeline. Once I'm inside the volume slider timeline, I'll create another layer on top of the layer with the line in it, and in this new layer, I'll create the actual knob that we're eventually going to be able to drag back and forth. This knob is going to be a smaller version of the background of the stop and play buttons, so it will be easier if I just copy and paste the graphics from the buttons.

Once I've created the knob, I'm going to select it (and not the line) and convert it to a movie clip symbol (with the registration point in the center). This will give us a symbol within a symbol. Make sure you give this knob an instance name as well. I'm going to give it an instance name of "slider_mc".

2. Prepare the Slider. Once we have the volume movie clip set, we're going to line all of the graphics up inside the movie clip in a way that will make our ActionScript code a little bit easier. One thing that will make the coding easy is to make the length of the line exactly 100 pixels. This line represents the groove inside of which the slider is sliding, and if we make it exactly 100 pixels wide, it will be really easy to perform our volume calculations when we start moving our slider back and forth.

The second thing we need to do with our line is to place it at an x/y coordinate of 0/0. This way, when we slide our knob around inside the volume_mc movie clip, we'll be dragging between the x coordinates of 0 and 100. Again, this will make our volume level much easier to calculate when we jump into ActionScript.

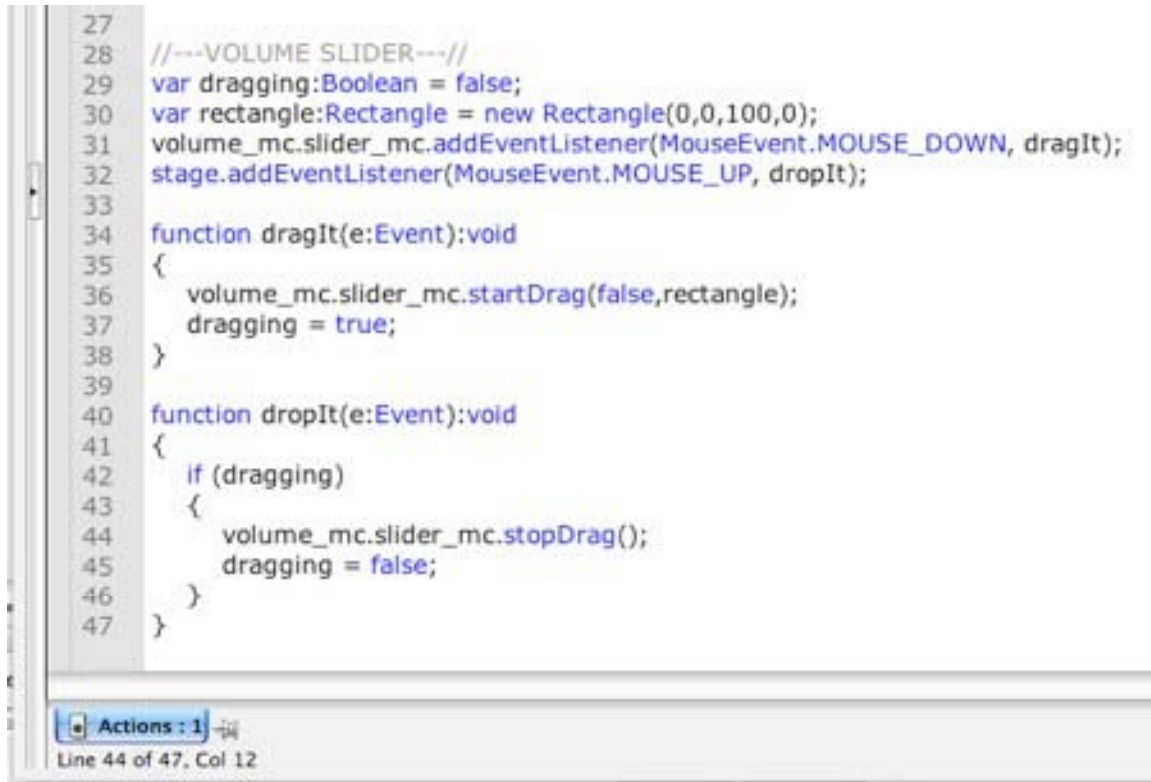
Finally, we're going to place the knob in it's initial position. By default, we want the volume turned up all the way, so we're going to place the knob at the far right end of the "groove" line. If you have the "Snap to Objects" magnet turned on at the bottom of your toolbar, it should be very easy to snap the knob to the right end of the line. Here's a zoomed-in view of our graphics as we should currently have them set up:



Remember that we have a movie clip within a movie clip here. The volume slider as a whole has been given an instance name of volume_mc, and inside this movie clip,

we have the knob set up as an individual movie clip with an instance name of slider_mc.

3. Make the Slider Draggable. Let's start by taking a look at the code. Select frame one of the Actions layer and open up your Actions Panel. Add the following code at the bottom of the code that's already there.

A screenshot of the Adobe Animate software interface, specifically the Actions Panel. The panel shows a list of actions for frame 1. The code is as follows:

```
27  
28 //---VOLUME SLIDER---//  
29 var dragging:Boolean = false;  
30 var rectangle:Rectangle = new Rectangle(0,0,100,0);  
31 volume_mc.slider_mc.addEventListener(MouseEvent.CLICK, dragIt);  
32 stage.addEventListener(MouseEvent.CLICK, dropIt);  
33  
34 function dragIt(e:Event):void  
35 {  
36     volume_mc.slider_mc.startDrag(false,rectangle);  
37     dragging = true;  
38 }  
39  
40 function dropIt(e:Event):void  
41 {  
42     if (dragging)  
43     {  
44         volume_mc.slider_mc.stopDrag();  
45         dragging = false;  
46     }  
47 }
```

The status bar at the bottom indicates "Line 44 of 47, Col 12".

You might notice right away, if you're relatively new to ActionScript 3, that making something draggable is a little more complex than it was in ActionScript 2. In ActionScript 3, the first thing we need to do is to create an Event Listener that will listen for the user to MOUSE_DOWN on the volume knob (line 31). This event listener triggers the "dragIt" function, in which we call on the "startDrag()" for the slider knob. This startDrag function works a little differently than in AS2. In AS2, it expected you to type in the limits of movement as 4 different numbers that represented the minimum and maximum x and y values. In AS3, it's still expecting the same thing, but now it's expecting it in the form of a Rectangle object, which we initially created on line 30.

When we created this new Rectangle, we put four numbers in the parentheses. These numbers represent the x coordinate, y coordinate, width, and height of the Rectangle object, in that order. And it's important to remember that since the knob is WITHIN the volume_mc movie clip, we need to enter in the x and y coordinates of the knob inside the context of that movie clip, which is why the first two numbers are set to zero. The third number is the width. And since our "groove" line is 100 pixels wide, and that's how far we want to be able to drag our knob left and right, then that's the number we need to use. Also notice that the height of this rectangle is zero. This is because we don't want the user to be able to drag the slider up and down--only left and right.

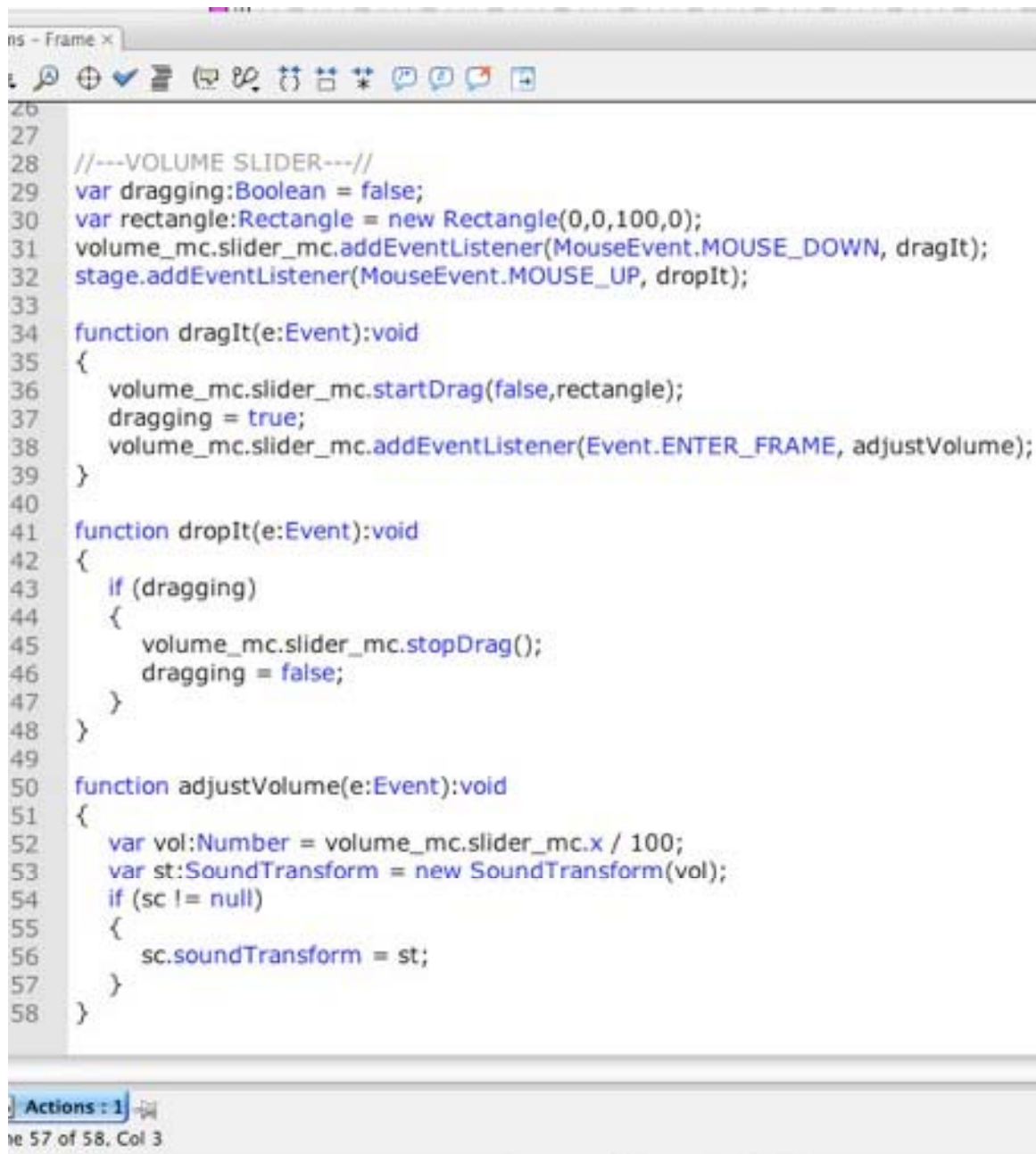
Anyways, since we've already created the Rectangle object in line 30, and we've stored it in a variable called rectangle, we can simple refer to that rectangle object inside our startDrag() method.

At this point, you might be wondering what the "dragging" variable is for. Well, in ActionScript 2, we had a handy little event handler called "onReleaseOutside". Without the "onReleaseOutside" event, if we dragged our knob over a little and then moved our mouse cursor up, away from the knob, before we released the mouse button, then the "onRelease" event wouldn't be triggered, and the "stopDrag()" method would never be called. So, in AS2, if we added the "onReleaseOutside" event to our code, then we could easily handle this.

Unfortunately, however, there is no longer an "onReleaseOutside" event, so we're forced to use a little bit of cunning trickery to get everything to work correctly. What we did was we created a MOUSE_UP event listener for the stage instead of for the slider knob. That way, no matter where the mouse cursor is when the user releases, the "dropIt" function will still be triggered. However, we don't want this to happen ANY time we click on the stage and release. We only want it to happen if we're currently dragging the slider. So we handle this by creating a Boolean (true/false) variable called "dragging" that keeps track of whether we're currently dragging the slider. Obviously, this would be set to "false" by default. And then, when we click on the slider and start dragging it, we switch it to "true" (line 37).

And then, when we run the "dropIt" function, we first check to see if we're currently dragging the slider. If not, then nothing happens. But if we are, it runs the "stopDrag" function, and everything is peachy! So, at this point, be sure to test your file to make sure that you can drag your slider within the restricted bounds that we set up in our Rectangle object.

4. Adjust the volume. When we start dragging, we not only want the slider knob to follow our cursor around, but we also want our volume to adjust as we move it. And the way we're going to achieve this is we're going to create an ENTER_FRAME event inside our "dragIt" function. This ENTER_FRAME event will cause a function to run over and over again at the current frame rate. In this function, we want to check the position of our slider and then use a little hocus pocus (a.k.a., math) to adjust the volume of our sound file accordingly.



As mentioned, we added our ENTER_FRAME event inside the dragIt function on line 38. This event listener triggers a function called adjustVolume. The adjustVolume function, which starts on line 50, contains all the code that examines the location of the slider knob and adjusts the volume accordingly. The first thing we need to do in order to adjust the volume is to calculate the number we want to use. In ActionScript 3, the number representing the volume is going to be a number between zero and one. Zero represents complete silence, 0.5 represents 50% volume, and 1 represents full volume. Since our slider is exactly 100 pixels wide, it's very easy to calculate what the volume needs to be. Simply find the x coordinate of the knob within the volume_mc movie clip and divide by 100 to get the adjusted volume level.

Note: If your slider is not 100 pixels wide, then you will need to use some fancier magic (I mean, math) in order to come up with this number. In order to accomplish

this, it's easiest to think in percentages. If your slider knob is 50% of the way across the slider area, then the volume needs to be adjusted to 0.5. If the slider area was 250 pixels wide, for example, the calculation would look like this:

```
var vol:Num = volume_mc.slider_mc.x / 250;
```

Easy enough, right?

Once the necessary volume level has been calculated, this number needs to be placed inside a new instance of the SoundTransform class. This SoundTransform object is then assigned to the "soundTransform" property of our SoundChannel object that we created to control the music. (For more information on the SoundChannel class, view the last tutorial.) This SoundChannel class is stored in a variable called "sc", and our SoundTransform class is stored in "st".

You'll notice that the SoundTransform object is only applied to "sc" IF "sc" is not null. If we don't include this "if" statement, then when we test our movie, we'll get an error if we try to adjust the volume before we actually hit the play button. This error occurs because before we hit "play", our SoundChannel object still contains a null value. It isn't assigned a value until we actually start playing the music. Anyways, if you test your movie now, you'll find that you now have fully functioning sound controls for your file.

* Part 3 - Creating a Pause Button

The Pause Button

Creating the code for the pause button is going to be similar to the code for the stop button, but it's going to take a little bit of extra tweaking in order to get it to work properly. With our stop button, we simply stopped the sound from playing so that when we clicked on the play button again, we were starting the song over from the beginning. When we hit the pause button, however, we want to be able to resume the playback from the point in the song where we were when clicked the pause button. So, in order to do this, we need to store the current position of the song whenever we click on the pause button.

1. Create the pause button. Naturally, before you can start adding code for your pause button, you need to actually create it. I created mine to go along with the theme I had already developed for my play and stop buttons, and then I gave the pause button an instance name of pause_btn.

2. Create a variable called "pos." In this variable, we'll keep track of the current playing position of our song. As you'll notice in the image below, I declared the pos variable just below the rest of my initial variable declarations, on line 7.

```

3
4 var music:Sound = new Sound(new URLRequest("walk.mp3"));
5 var sc:SoundChannel;
6 var isPlaying:Boolean = false;
7 var pos:Number = 0;
8
9 stop_btn.addEventListener(MouseEvent.CLICK, stopMusic);
10
11 function stopMusic(e:Event):void

```

I also set the pos variable equal to a default value of zero, because--naturally--when our file first opens, we're going to be at the very beginning of the song. This number, by the way, represents how far we are into the song in milliseconds. The reason for this is that when we use the "play()" method of the Sound class, we can enter in a number within the parentheses in order to tell Flash how far into the song (in milliseconds) we would like to start playing. Also, there is a property of the SoundChannel class called "position" which gives us the current position of the song (again--you guessed it--in milliseconds).

3. Write the code for your pause button. This code will consist of the usual event listener and corresponding function, as seen below:

```

8
9 pause_btn.addEventListener(MouseEvent.CLICK, pauseMusic);
10
11 function pauseMusic(e:Event):void
12 {
13     if (isPlaying)
14     {
15         pos = sc.position;
16         sc.stop();
17         isPlaying = false;
18     }
19 }
20

```

In line 9, we add our event listener, which causes our "pauseMusic" function to execute when triggered. Within this function, we're checking the "isPlaying" variable to see if the music player is currently playing music (this variable was created in a previous tutorial), and if the music IS currently playing, then 3 things need to occur:


- * We need to store the current position of the song, as seen in line 15. The current position can be accessed with the "position" property of our SoundChannel object, which we've called "sc." In this example, we're storing this position in the "pos" variable that we created back in step 2.

- * We need to stop the sound (line 16). The code for doing this is nothing new. It's the same code we used for the stop button.

- * We need to change the "isPlaying" variable to false. Since the music will no longer be playing once we hit the pause button, the "isPlaying" variable needs to reflect that.

4. Update the code within the "playMusic" function. We already created this function in the first tutorial of this series, but we need to add one more thing to it.


```
33
34 function playMusic(e:Event):void
35 {
36     if (!isPlaying)
37     {
38         sc = music.play(pos);
39         isPlaying = true;
40     }
41 }
```



Before we altered the code, the parentheses for the "music.play()" method were empty. If we were to leave it empty, then Flash would assume that there was a zero in the parentheses, which would cause the song to play from the beginning. But if we've only hit the pause button, then we don't want to start over from the beginning when we hit play again. We want to start from where we were when we hit pause. In order to do this, we simply put the desired position, which is stored in the "pos" variable, in the parentheses.

5. Update the code within the "stopMusic" function. Because the "music.play()" method is now using the "pos" variable to point to a specific position within the music file (instead of pointing to the default of zero), we now need to update the stopMusic function so that when we hit the stop button, the "pos" variable is reset to zero.

```
22
23 function stopMusic(e:Event):void
24 {
25     if(sc != null)
26     {
27         sc.stop();
28         pos = 0;
29         isPlaying = false;
30     }
31 }
```



Now, when we hit the play button after hitting the stop button, the music will once again start from the beginning. And that about wraps it up for our music player tutorials. I hope you've learned a lot from these. Be sure to keep me updated with the types of tutorials you'd like to see on the site, and if you haven't subscribed to the site yet, you can do so from the subscription options in the left column of the website.