Sound Controls Exercises

Adobe Flash CS4/CS5 and ActionScript 3.0

These exercises show you how to use and control external MP3 files.

Part 1/ Introduction

1. Open the FLA file named *sound_controls_1.fla*.

2. Open the Actions panel. Read the instructions there and complete the exercises suggested in the commented lines. Although those exercises are not essential to using sound controls, it is hoped that you will understand the usefulness of the *trace* function. The results of *trace* always appear in the Output panel.

3. Close the SWF window.

4. Open the Library and examine the objects there. You'll see three buttons and nothing else. (Note that these buttons are already on the Stage and already have instance names assigned to them.)

5. Outside Flash, open the folder named "audio" and note that there are three MP3 files inside. The *relationship* of this folder and the SWF are very important, because the ActionScript you are going to write will look for a folder with this **name** at this **location**, *relative to* the SWF. (Move the folder, or rename it, and the AS will not work!)

Part 2/ Simple Play and Stop buttons

1. *Save As:* Rename the FLA to *sound_controls_2.fla*

2. In the Actions panel, delete everything from Line 6 to the end.

3. Below the three listeners, write:

```
function stopSound(myEvent:MouseEvent):void {
        myChannel.stop();
}
function playSound(myEvent:MouseEvent):void {
        myChannel = mySound.play();
}
```

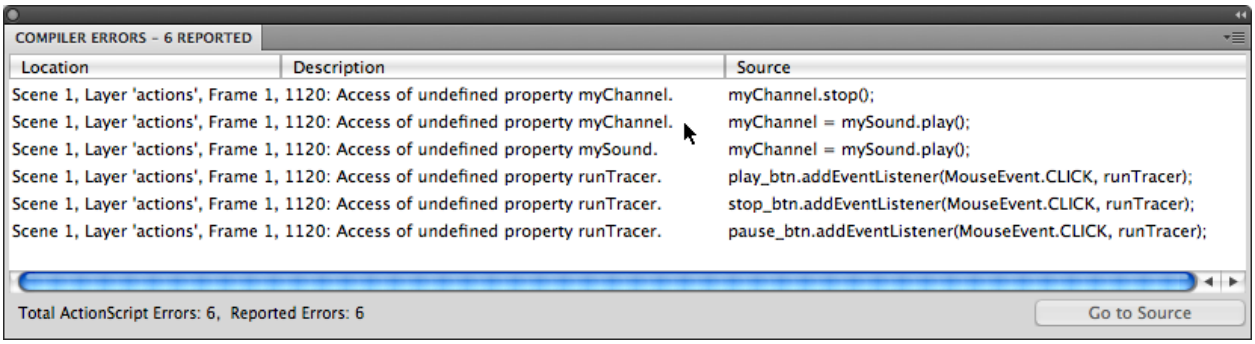
4. Look at the two functions and try to take them apart. That is, *analyze* what you see in the functions.

a. Each function has its own unique name. One is named *stopSound*. The other is named *playSound*.
b. Each function contains a command that should be familiar to you. One command is *stop();* — the other command is *play();*
c. Two things are not accounted for—they are new. One is *myChannel* and the other is *mySound*. These are **variable names**.

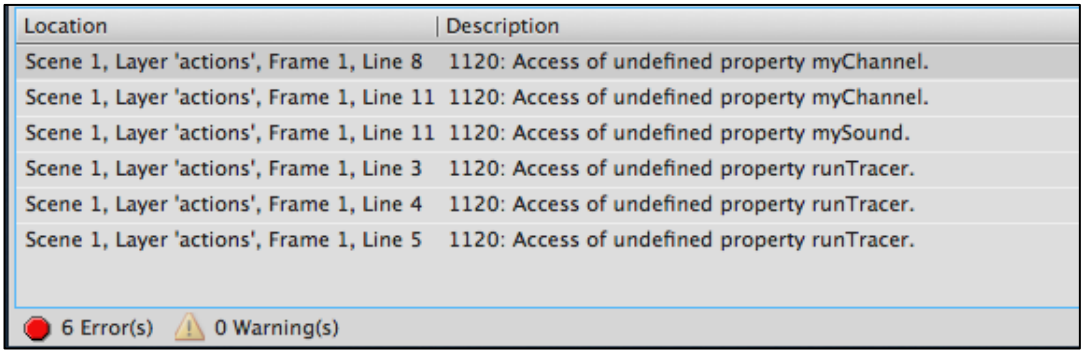
Variables are created by you, the creator of the FLA. Each variable is assigned a unique **name** as soon as it is created. Many programmers start their variable names with either *the* or *my* to make it easy to recognize, later, which things in the script are the variables. This is just a convention, not a requirement. However, it does make it easy to avoid using ActionScript's reserved words (such as *event,* or *name,* or *source*) and thus causing errors.

See pages 208–211 in *Adobe Flash CS5 Professional Classroom in a Book* for more information about variables.

5. Save and test the movie. You will see some errors:



Above: CS4 error outputs



Above: CS5 error outputs

The first three errors concern *the missing variables*. Flash even calls them "undefined" to give you a big hint that you are trying to use a variable that does not exist.

The other three errors all refer to the function that you deleted. The listeners need the function that they are scripted to look for (in this case, *runTracer*).

6. First you will fix the three function errors:
    a. For the listener associated with the Play button, replace *runTracer* with the name of your new function *playSound*.
    b. For the listener associated with the Stop button, replace *runTracer* with the name of your new function *stopSound*.
    c. For the listener associated with the Pause button, simply *comment it out* for now. That means: Type two slashes // at the very start of the line. The whole line of ActionScript will become gray, meaning that Flash will not read it.

7. Second, to fix the variable errors, you will create two new variables named *mySound* and *myChannel*. These will be used to control the external MP3 files.

    This is how you declare new variables:

    Variables are usually declared at the *top* of the script. Add these lines (ABOVE the listeners):

```
var mySound:Sound;
var myChannel:SoundChannel;
```

    The term that appears after the colon in both lines above (*Sound* and *SoundChannel*) are the **data types** used by ActionScript. When you declare a variable, you also specify its data type, as shown here. For a list of data types in AS3, see: http://en.wikipedia.org/wiki/ActionScript#Data_types

8. Save and test the movie. Click the Play and Stop buttons. You will see some errors:

    TypeError: Error #1009: Cannot access a property or method of a null object reference.
      at sound_controls_2_fla::MainTimeline/playSound()

    TypeError: Error #1009: Cannot access a property or method of a null object reference.
      at sound_controls_2_fla::MainTimeline/stopSound()

    A **null object** is an empty object, an object that contains nothing. A function can't operate on an empty object.

9. To fix the null object reference, you will instruct Flash to **load** one of the external MP3 files into the Sound object. Add these two lines BELOW the two variable declarations:

```
mySound = new Sound;
mySound.load(new URLRequest("audio/river.mp3"));
```

Note that you must have an MP3 file named *river.mp3* inside the folder named *audio* for this to work. Note also that the quotation marks are *required*.

If you get an error message like the one below, it means that either the FLA and SWF are not in the same folder along with the folder named *audio,* or the file named *river.mp3* does not exist in that folder.

```
Error #2044: Unhandled IOErrorEvent:. text=
Error #2032: Stream Error.
   at sound_controls_2_fla::MainTimeline/frame1()
```

10. Save and test the movie. Click the Play button, then the Stop button, then the Play button. You should hear the music playing, etc. However, there's a problem. Click the Play button, *then click it again.* Uh-oh.

11. To fix this sound overlap problem, you need to create *a new variable*. (This might seem annoying, but it will be necessary for making the pause button work too, so it's really not a wasted effort.) Type this line below your other variable declarations:

```
var isPlaying:Boolean = false;
```

For more about Boolean variables:
http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/Boolean.html

12. Edit your two functions to include the new true/false "flag" to tell Flash whether the audio is playing or not:

```
function stopSound(myEvent:MouseEvent):void {
	myChannel.stop();
	isPlaying = false;
}
function playSound(myEvent:MouseEvent):void {
	myChannel = mySound.play();
	isPlaying = true;
}
```

13. It would be nice if that were the end of it, but it's not. Flash does not know what to do with these true/false values unless you write instructions. You need to tell Flash what to do *if* the audio is playing, or what to do *if* the audio is not playing, or both. We try to think of the most minimalist way to tell Flash what it needs to know. So please consider the possibilities:
    a. Do you need to tell Flash any conditions for stopping the audio?
    b. Do you need to tell Flash any conditions for playing the audio?
    c. What does Flash need to know to prevent overlapping audio?

    The answer to *a.* is no. The answer to *b.* is yes. **The answer to *c.* is the key.**

14. *If* the audio is playing, *do not* make it play (again). But we need to write this instruction in the function that makes it play. So you need to set up a condition that permits the audio to play only *if it is not playing* already.

```
function playSound(myEvent:MouseEvent):void {
        if (!isPlaying) {
              myChannel = mySound.play();
              isPlaying = true;
        }
}
```

Using the exclamation point in this way (!isPlaying) is the same as saying "not"; the Boolean way of saying "not equal to," for example, is: !=

The four lines that begin with *if* and end with a curly brace are called an **if statement**. By nesting the *play();* command (and the Boolean flag) *inside* the if statement, you ensure that those things happen *only if* the conditions set in the first line apply. What are the conditions? In this case: *If* isPlaying is false (!isPlaying) … then the rest will happen. And if not? Then nothing (new) happens.

15. Save and test your movie. The Play and Stop buttons should work perfectly, no matter how you click them.

Your complete script now should look like this:

```
var mySound:Sound;
var myChannel:SoundChannel;
var isPlaying:Boolean = false;

mySound = new Sound;
mySound.load(new URLRequest("audio/river.mp3"));

play_btn.addEventListener(MouseEvent.CLICK, playSound);
stop_btn.addEventListener(MouseEvent.CLICK, stopSound);
// pause_btn.addEventListener(MouseEvent.CLICK, runTracer);

function stopSound(myEvent:MouseEvent):void {
        myChannel.stop();
        isPlaying = false;
}
function playSound(myEvent:MouseEvent):void {
        if (!isPlaying) {
              myChannel = mySound.play();
              isPlaying = true;
        }
}
```

3/ Adding script for a Pause button

You could operate most audio applications for journalism with only a Play and a Stop button, but it's nice for the users if you also provide a Pause button—even if it does require some extra effort.

1. *Save As:* Rename your FLA to *sound_controls_3.fla* (so that you preserve the simpler script for Play and Stop).

2. Write a new function below your other functions:

```
function pauseSound(myEvent:MouseEvent):void {
       if (isPlaying) {
            p = Math.floor(myChannel.position);
            myChannel.stop();
            isPlaying = false;
       }
}
```

   Note that a Pause button should work *only if* the audio is playing (if isPlaying is *true*).

3. Examine your new function and look for what you recognize. Everything in this function has been covered earlier in this exercise—*except* the line that begins with *p*. What is *p*? It is a new **variable**.

   So the first step is to *add another variable declaration* after all the others:

```
var p:uint = 0;
```

   The variable *p* is going to hold the **position** of the audio file. The position is represented in *milliseconds.* For example, 29808.004535147393 is a reading of *SoundChannel.position* at the end of a (supposedly) 30-second audio file.

   The **data type** in this case tells Flash to expect an unsigned integer (*uint*), which is a positive number. This is different from and preferable to using a number (*Number*). If you need negative integers, use *int* instead. For more information, see: http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/Number.html

4. Now, about what this line actually *does* (it is in your new *pauseSound* function):

```
p = Math.floor(myChannel.position);
```

   When you click the Pause button, this bit of script tells Flash to capture the position of the SoundChannel object; then Flash takes the number and runs it through the Math.floor method, which essentially *rounds it down,* or cuts off all the data after the

decimal point. See:
http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/Math.html
This done, Flash sticks the result into the variable *p,* meaning that *p* now contains that number—so Flash can use it when you click the Play button again.

5. Once you have *p,* what are you going to do with it? You need to add it to both the *playSound* function and the *stopSound* function. The *playSound* function needs to use *p* to know where to restart the audio file:

```
function playSound(myEvent:MouseEvent):void {
	if (!isPlaying) {
		myChannel = mySound.play(p);
		isPlaying = true;
	}
}
```

The *stopSound* function needs to reset *p* to zero:

```
function stopSound(myEvent:MouseEvent):void {
	myChannel.stop();
	p = 0;
	isPlaying = false;
}
```

6. What's left to do? Well … does your Pause button work yet? No, because it's still commented out. So, delete the two slash marks. You will also need to replace *runTracer* with the name of your new function *pauseSound*.

```
pause_btn.addEventListener(MouseEvent.CLICK, pauseSound);
```

7. Save and test your movie. Everything should work.

Your complete script now should look like this:

```
var mySound:Sound;
var myChannel:SoundChannel;
var isPlaying:Boolean = false;
var p:uint = 0;

mySound = new Sound;
mySound.load(new URLRequest("audio/river.mp3"));

play_btn.addEventListener(MouseEvent.CLICK, playSound);
stop_btn.addEventListener(MouseEvent.CLICK, stopSound);
pause_btn.addEventListener(MouseEvent.CLICK, pauseSound);
```

```
function stopSound(myEvent:MouseEvent):void {
   myChannel.stop();
   p = 0;
   isPlaying = false;
}
function playSound(myEvent:MouseEvent):void {
   if (!isPlaying) {
        myChannel = mySound.play(p);
        isPlaying = true;
   }
}
function pauseSound(myEvent:MouseEvent):void {
   if (isPlaying) {
        p = Math.floor(myChannel.position);
        myChannel.stop();
        isPlaying = false;
   }
}
```

4/ Playing more than one audio file

A common audio application for journalism is to allow users to choose among several audio files (for example, interviews with various people). Here you'll build on the previous scripts by adding two more MP3s and making sure only one audio file can play at a time.

1. Open the FLA named *sound_controls_4.fla* and check the Library. Nothing new has been added there. However, two more instances of the single Play button have been dragged onto the Stage.

2. Unlock the buttons layer and check the instance name of each button. You'll see that the three Play buttons are now named *song1_btn, song2_btn,* and *song3_btn*. The Stop and Pause buttons have the same instance names as before.

3. Lock the buttons layer again. You're not going to change anything on the Stage.

4. Unlock the dynamic text layer and click the text field (under the words *Now Playing*). In the Properties panel, notice that its instance name is *title_txt*.

5. Lock the dynamic text layer again.

6. Open the Actions panel. All the script is the same as before, with one exception: There is a listener for *song1_btn*. (This replaced the listener for *play_btn,* which is gone.) So you are starting with script you already understand.

7. You're going to modify the script. First you'll write a new function:

```
function song1_data(myEvent:MouseEvent):void {
	songfile = "audio/river.mp3";
	songtitle = "The River";
	playSound(null);
}
```

This function refers to two new **variables** that don't exist yet, so you'll need to *add them* (below the other variable declarations):

```
var songfile:String;
var songtitle:String;
```

The **data type** in this case (String) indicates text. For more information, see: http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/String.html

8. Your new function is doing something very cool—it is calling another function! What does that mean? You already have a function named *playSound*; you used it in the earlier exercises. Now your new function is going to set *playSound* into action. You do not have to write the lines of *playSound* into the new function; it can just use what's already there in your script.

   However, if we *call a function* (that is, set it into action) from inside another function, it is not triggered by a MouseEvent. This requires a small change in the existing *playSound* function:

```
function playSound(myEvent:Event):void {
```

   (Where it used to read *MouseEvent,* it now reads only *Event*.)

9. Next you will move two lines of script into the *playSound* function. First *cut* these two lines:

```
mySound = new Sound;
mySound.load(new URLRequest("audio/river.mp3"));
```

   Then paste them into the *playSound* function:

```
function playSound(myEvent:Event):void {
	mySound = new Sound;
	mySound.load(new URLRequest("audio/river.mp3"));
```

Continued …


```
        if (!isPlaying) {
            myChannel = mySound.play(p);
            isPlaying = true;
        }
    }
```

10. There's one more change to make to the *playSound* function. What you need to ensure is that the function can be used by all three Play buttons. But each Play button must play a different song. How can that happen when the song file is specifically named in the *playSound* function?

    That's where the variable *songfile* comes into use. By storing the folder and filename of the MP3 file in this variable (see step 7, above), you made it possible to swap the hard-coded folder/filename (*audio/river.mp3*) with the more flexible **variable name**:

    ```
    mySound.load(new URLRequest(songfile));
    ```

    Note that when you're using a **variable name,** there are NO quotation marks!

11. Save and test the movie. You will get an error when you click the Song 1 button:

    ```
    TypeError: Error #2007: Parameter url must be non-null.
      at flash.media::Sound/_load()
      at flash.media::Sound/load()
      at sound_controls_4_fla::MainTimeline/playSound()
    ```

    You might associate this with the word *null* used in the *song1_data* function (step 7), but that's NOT the problem. The problem is *in the listener* for the Song 1 button:

    ```
    song1_btn.addEventListener(MouseEvent.CLICK, playSound);
    ```

    You rewrote the *playSound* function in a way that now makes it dependent on the new *song1_data* function—for that reason, when the listener calls the *playSound* function, it does not work! What you need to do, instead, is call the new function:

    ```
    song1_btn.addEventListener(MouseEvent.CLICK, song1_data);
    ```

12. Save and test the movie. The Song 1 button, the Stop button, and the Pause button should all work correctly. (The Song 2 and Song 3 buttons don't work yet.)

13. Now everything is in place for you to do some easy copying and pasting that will allow the other two buttons to work! First, copy the whole *song1_data* function and paste it twice. Change the function names to *song2_data* and *song3_data,* then add the MP3 file and song title information:

```
function song2_data(myEvent:MouseEvent):void {
	songfile = "audio/jerseygirl.mp3";
	songtitle = "Jersey Girl";
	playSound(null);
}
function song3_data(myEvent:MouseEvent):void {
	songfile = "audio/thunderroad.mp3";
	songtitle = "Thunder Road";
	playSound(null);
}
```

(You can use different MP3 files if you have them on hand.)

14. Finally, copy the *song1_btn* listener and paste it twice, then modify it:

```
song1_btn.addEventListener(MouseEvent.CLICK, song1_data);
song2_btn.addEventListener(MouseEvent.CLICK, song2_data);
song3_btn.addEventListener(MouseEvent.CLICK, song3_data);
```

Don't forget to change BOTH the button name AND the function name to match!

15. What about the variable named *songtitle*? You have not used it yet. All that's needed is to add one line to the *playSound* function:

```
function playSound(myEvent:Event):void {
	mySound = new Sound;
	mySound.load(new URLRequest(songfile));
	title_txt.text = songtitle;
	if (!isPlaying) {
		myChannel = mySound.play(p);
		isPlaying = true;
	}
}
```

This line of script speaks to the *text* **attribute** of the **dynamic text field** named *title_txt* (see step 4, above). It tells that text field to contain whatever is contained in the **variable** named *songtitle*. Each song data function writes a different title into that variable (*songtitle*), so the text in the movie will always show the title of the song that's playing. (Another example of how useful variables can be!)

16. Save and test your movie. All works great if you simply play and stop, play and stop. But using the Pause button, or clicking a different Song button without *first* clicking Stop, shows you have problems in this script.

This illustrates that **whenever you add interactions or new functions** to a script, it is very likely that something will break. This is normal. It always takes patience and perseverance to troubleshoot and debug a new script.

17. **Debugging:** The Play and Stop buttons work, but without clicking Stop first, another Play button will not work properly. (Task 1: Make other songs stop playing when you click a different Play button.) Pause works, but clearly it keeps the same position (*p*) even after you have started playing a different song. (Task 2: Tell *p* to return to zero when a new song is selected.)

    In fact there's nothing that needs to change in the *pauseSound* function.

    Everything that's not working happens because of the way the *playSound* function is written. You need to change it around so that it takes into account whether another song is *already* playing.

    Well, think about that. Look at this part of the *playSound* function:

```
if (!isPlaying) {
     myChannel = mySound.play(p);
     isPlaying = true;
}
```

    Right now, it only does something if the Boolean *isPlaying* is false. That worked when you had only one audio file. Now you have many.

    The Boolean *isPlaying* is true when any song is playing. So you can check and see if a song is playing. This is what you want: *If* anything is playing right now, make it stop. Then play the *songfile* starting at 0 (*not* at whatever value is stored in *p*). No need to change the value of *isPlaying* to true—it was already true.

```
if (isPlaying) {
     myChannel.stop();
     myChannel = mySound.play(0);
```

    That takes care of changing to a new song. What if no song is playing? Above, you have *If anything is playing …* Now you need what happens *otherwise*.

    Otherwise (*else*), do the same thing your script was doing before. That is, if nothing is playing right now, then play *songfile* starting at the value stored in *p*. Change the value of *isPlaying* to true (because it had been false).

```
} else {
     myChannel = mySound.play(p);
     isPlaying = true;
```

Here is the complete newly rewritten *playSound* function:

```
function playSound(myEvent:Event):void {
       mySound = new Sound;
       mySound.load(new URLRequest(songfile));
       title_txt.text = songtitle;
       if (isPlaying) {
            myChannel.stop();
            myChannel = mySound.play(0);
       } else {
            myChannel = mySound.play(p);
            isPlaying = true;
       }
}
```

18. Save and test your movie. The three Song buttons work great now. But there's a problem if you pause one song and then play another. Why? Because when a song is paused, it's *not playing*. Look at the *playSound* function. What does it do when no song is playing (*else*)?

    This is how you troubleshoot and build a script. One thing at a time, with lots of testing in between. Always click all of your buttons lots of times. Try to break things. That is how you learn what needs to be fixed.

19. To fix this pausing problem, you're going to change the *playSound* function a little and the *pauseSound* function a lot.

```
function playSound(myEvent:Event):void {
       mySound = new Sound;
       mySound.load(new URLRequest(songfile));
       title_txt.text = songtitle;
       if (isPlaying) {
            myChannel.stop();
            myChannel = mySound.play(0);
       } else {
            myChannel = mySound.play(0);
            isPlaying = true;
       }
}
```

Take the variable p out of the *playSound* function altogether, as shown above.

20. Change the *pauseSound* function as shown below:

```
function pauseSound(myEvent:MouseEvent):void {
       if (isPlaying) {
            p = Math.floor(myChannel.position);
            myChannel.stop();
            isPlaying = false;
            isPaused = true;
       } else if (isPaused) {
            myChannel = mySound.play(p);
            isPlaying = true;
            isPaused = false;
       }
}
```

You are adding a new condition that causes the Pause button to act as a *toggle*. The Song buttons no longer resume a paused track—because you removed the *p* from the *playSound* function (step 19).

21. Add the new Boolean variable to the list of variables near the top of your script:

```
var isPaused:Boolean = false;
```

22. Save and test your movie. Try out the newly rescripted Pause button!

23. Your script has one last problem. If you click the Stop button *before you have played any song,* Flash throws an error:

TypeError: Error #1009: Cannot access a property or method of a null object reference.
  at sound_controls_4_fla::MainTimeline/stopSound()

That's because *nothing is playing*.

How would you ever discover that? Who would click the Stop button when nothing is playing? You just don't know—that's what I meant about troubleshooting (step 18). You have to try everything, even if it is illogical.

To fix the stopping problem, you need another if statement to force the *stopSound* script to behave nicely (next page):

```
function stopSound(myEvent:MouseEvent):void {
        if (isPlaying) {
             myChannel.stop();
             p = 0;
             isPlaying = false;
             isPaused = false;
        }
}
```

Note that I also sneaked in a new line to prevent the Stop button from messing up the *pauseSound* function.

The script is now bulletproof. No matter what the user does, everything works, and nothing ever breaks.

Your complete script now should look like this:

```
var mySound:Sound;
var myChannel:SoundChannel;
var isPlaying:Boolean = false;
var isPaused:Boolean = false;
var p:uint = 0;
var songfile:String;
var songtitle:String;

song1_btn.addEventListener(MouseEvent.CLICK, song1_data);
song2_btn.addEventListener(MouseEvent.CLICK, song2_data);
song3_btn.addEventListener(MouseEvent.CLICK, song3_data);
stop_btn.addEventListener(MouseEvent.CLICK, stopSound);
pause_btn.addEventListener(MouseEvent.CLICK, pauseSound);

function song1_data(myEvent:MouseEvent):void {
        songfile = "audio/river.mp3";
        songtitle = "The River";
        playSound(null);
}
function song2_data(myEvent:MouseEvent):void {
        songfile = "audio/jerseygirl.mp3";
        songtitle = "Jersey Girl";
        playSound(null);
}
```

Continued ...


```
function song3_data(myEvent:MouseEvent):void {
        songfile = "audio/thunderroad.mp3";
        songtitle = "Thunder Road";
        playSound(null);
}
function stopSound(myEvent:MouseEvent):void {
   if (isPlaying) {
        myChannel.stop();
        p = 0;
        isPlaying = false;
        isPaused = false;
   }
}
function playSound(myEvent:Event):void {
   mySound = new Sound;
   mySound.load(new URLRequest(songfile));
   title_txt.text = songtitle;
   if (isPlaying) {
        myChannel.stop();
        myChannel = mySound.play(0);
   } else {
        myChannel = mySound.play(0);
        isPlaying = true;
   }
}
function pauseSound(myEvent:MouseEvent):void {
   if (isPlaying) {
        p = Math.floor(myChannel.position);
        myChannel.stop();
        isPlaying = false;
        isPaused = true;
   } else if (isPaused) {
        myChannel = mySound.play(p);
        isPlaying = true;
        isPaused = false;
   }
}
```

This probably seems like a lot of work to do, and it is. However, if you have never done any programming or scripting before, you have now experienced a pretty normal workflow of testing and fixing, testing and fixing—one function at a time.

If you review what you've done, you'll see that the basic Play and Stop for a single audio file (pages 1–5) were pretty straightforward. If you do not need to add more complexity, then just don't do it!