CHAPTER 18

Sound Lingo

In this chapter

- Triggering and controlling sound with Lingo
- Setting the volume level of sound
- Preloading digital sounds
- Using the Beatnik Lite Xtra
- Sound channel objects

Using the sound-related Lingo commands frees you from the limitations of just two audio channels in the Score. In this chapter, you learn to play linked and embedded sound files, start and stop the playback of a sound, trigger an event upon the completion of an event, and control sound volume. Lastly, we cover the new sound channel objects and how they enable you to play back sounds much faster, play parts of sound members, and other controls.

Controlling Sound with Lingo

You can use digital audio to add narration, special effects, and background music to an interactive media project. Sound in your movie can be triggered by a variety of events, including these:

- When the user clicks a button or other sprite
- When the mouse pointer rolls over a specific sprite
- When the playback head plays a specific series of frames
- While the playback head loops through a single frame

There are three Lingo commands that you can use to play a sound file in your movie. You can use the `puppetSound` command to play an embedded sound cast member, or you can play an external (linked) sound file by using the `sound playFile` command. The third method is through using sound references.

We look at puppet sounds in the first part of this chapter. A later section, “Playing Linked Files with Lingo,” discusses linked sound files. “Sound Channel Objects” the end of the chapter teaches you to use this new type of object.

Using puppet sounds

The most common method of adding sound to your movie is to use the `puppetSound` command. The command's syntax is very simple:

```
puppetSound castSound
puppetSound channelNumber, castSound
```

The `channelNumber` parameter determines which channel the sound plays in. When you have only one parameter, it is assumed to be the name or number of a sound cast member. Although the Score window has only two sound channels, you can access additional sound channels by using Lingo. In the following statement, the sound `"musicBed"` is played in channel 4:

```
puppetSound 4, "musicBed"
```

The `puppetSound` command can support a maximum of eight sound channels on a Macintosh system, or four on a Windows system.

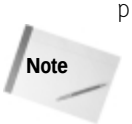
The `castSound` parameter is the cast member name or number of the sound cast member. If you use the cast member name, it must be enclosed within quotation marks. For instance, the following instruction plays a sound in the cast named siren.wav or siren.aiff, in channel 1:

```
puppetSound 1, "Siren"
```

Note that you don't need to include the sound file's extension when using the `puppetSound` command.

If the "Siren" cast member is in cast position 5, you can use the following command form:

```
puppetSound 1, 5
```



When you issue the `puppetSound` command, the sound takes precedence over any sounds in the Score's two sound channels.

Mechanically speaking, sounds in Director play only at these times:

- When the playback head moves, as in a `go to the frame` handler, or when the playback head moves through a series of frames, each of which has the sound in the sound channel
- Whenever the `updateStage` command is issued

When a sound fails to play, the most common cause is the programmer's failure to follow the `puppetSound` command with the `updateStage` command.

Adding music and special effects to a multimedia presentation does not require that you create the sound files from scratch. A large body of clip media exists that you can purchase and use royalty-free in your productions.

Always check the licensing agreement that accompanies any clip media you use. You can use some clips only for personal use or for internal use within your company. Others are royalty-free in all cases. Almost every publisher of clip media, however, prohibits the user from repackaging some or all of the media clips for resale as a clip media collection.

Triggering sound by using Lingo

In the following steps, you modify the sound 1.dir movie to play sounds when the user clicks a sprite on the Stage (see Figure 18-1).

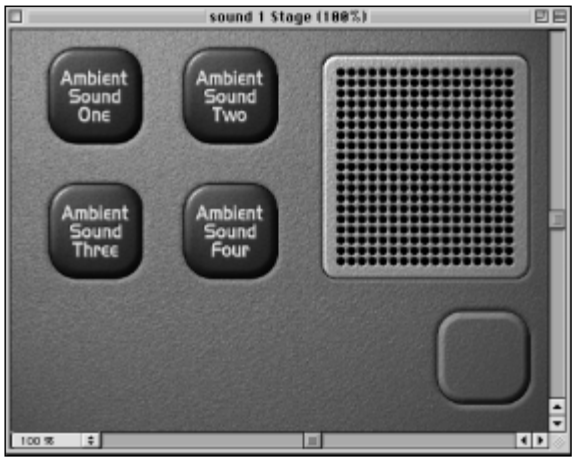


Figure 18-1: The sound 1.dir movie

To complete the movie, you add behaviors (using the `puppetSound` command) to play the appropriate sound file when the user clicks an object. All the sounds are internal to the movie; that is, they are embedded and not linked external files.

You can find sound 1.dir on the companion CD-ROM in the EXERCISE:CH18 (EXERCISE\CH18) folder.

Using Lingo to Add Sounds to noise.dir

1. Open the sound 1.dir movie in Director.
2. Select the Ambient Sound One button sprite.
3. Ctrl+click (right-click) on it and choose Script to create a new behavior for that sprite.

4. Modify the script to match Figure 18-2.

Figure 18-2: This script plays "sound 1" in the first sound channel.

5. Name the script **B_sound button**.
6. Close the Script window and save the movie.
7. Play the movie and test it by clicking the Ambient Sound One button. The sound plays for a few seconds and then stops.
8. We have three buttons still without scripts. This would be a good opportunity to make the script a little more generic. Open the Script window B_sound button.
9. Create an `on getPropertyDescriptionList` handler, as shown in Figure 18-3. This script uses two properties: `pChannel` to hold the channel number the sound plays in, and `pSound` to hold the name of the sound member.

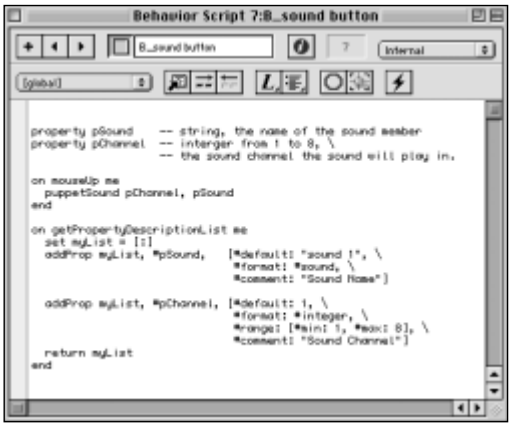


Figure 18-3: Make a more generic script for playing sounds.

10. Drag the behavior onto one of the other sprites. When you do, a dialog box pops up (see Figure 18-4).

Figure 18-4: The dialog box created from the getPropertyDescriptionList handler.

11. If you dragged it onto Ambient Sound Two, then choose sound 2 for Sound Name and move the Sound Channel slider to number 2.

12. Repeat Step 11 for the remaining buttons, applying the appropriate sound and channel for each.

13. Save the movie again and play it. Test each object to be sure that each sound plays. Because they are playing in different channels, the sounds can play at the same time.

Note that we gave the sound cast members different names from those of the bitmap cast members. If they had the same names, Director would play or use the lowest-numbered cast member when the cast member's name is referenced in a Lingo command, with the result that the sound would not play when you click the button.

Attaching sound to sprites and buttons

It is a good idea to give some visual feedback, to go along with the sound, when a user clicks the button. Create another behavior, as shown in Figure 18-5.

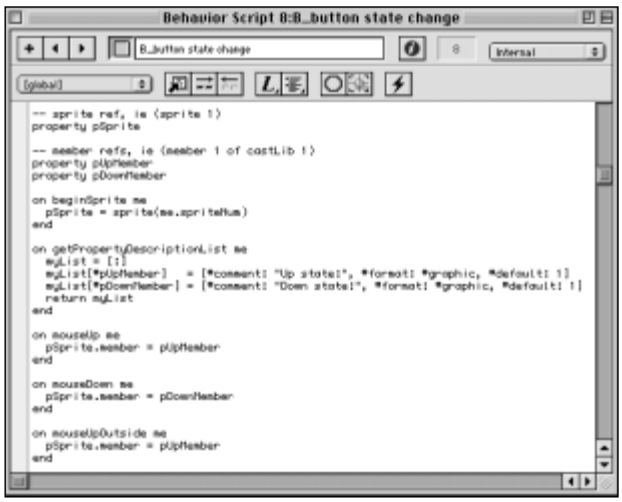


Figure 18-5: A simple behavior to handle up and down states

If you studied Chapter 14, then this handler looks basic to you. If you wanted, you could use the behavior created in Chapter 14 instead. This one is straightforward—it sets the button graphic to the down state that you specify when you apply the behavior. The dialog box is shown in Figure 18-6.

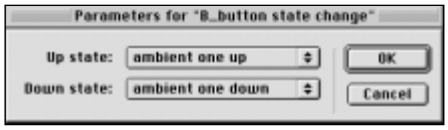


Figure 18-6: Choosing the up and down states for the buttons

To make sound play continuously (or until another sound or action interrupts it), select the sound cast member in the Cast window (not the sprite that you click to play the sound). Then click the Cast Member Properties button, and be sure that the Loop check box is checked. You need to do this for each button.

Finally, if you set a sound file to loop, be sure that the clip's beginning and end blend well. Sometimes just adding a fade to a pause at the end and a fade-in does the trick. Fade-in and fade-out are discussed in the section "Controlling Fade-in/Fade-out." Save this movie as **sound 2.dir**.

Stopping sound

After a sound has started, you might not always want it to finish playing. Lingo provides at least two methods for terminating the playback of a sound. First, you can issue the `puppetSound` command with the 0 (zero) parameter, as follows:

```
puppetSound 0
```

This command causes Lingo to relinquish control of the sound channels and return that control to the Score. The syntax `puppetSound 0` will kill sound in channel 1. To specify a specific channel, such as channel 2, write the following code:

```
puppetSound 2, 0
```

Another method of stopping one sound is to begin playing another sound in the same sound channel. A Sound channel can play only one sound at a time.

In the next series of steps, you provide a mechanism for the user to stop the audio. As the sound2.dir movie is currently constructed, every sound that is played continues to loop. By modifying the noise1.dir movie as outlined in the exercise, you enable the following actions:

- ✦ Clicking the Stop Sound button terminates all sounds.
- ✦ Clicking the Stop Sound button again just turns that sound off.

Adding a Button to Stop the Sound

1. Open the sound 2.dir movie.
2. Select the gray plug in the lower-right corner, and then select cast member `"stopSound up" (member 21)` and choose Edit➪Exchange Cast Members. Your Stage looks like that in Figure 18-7.

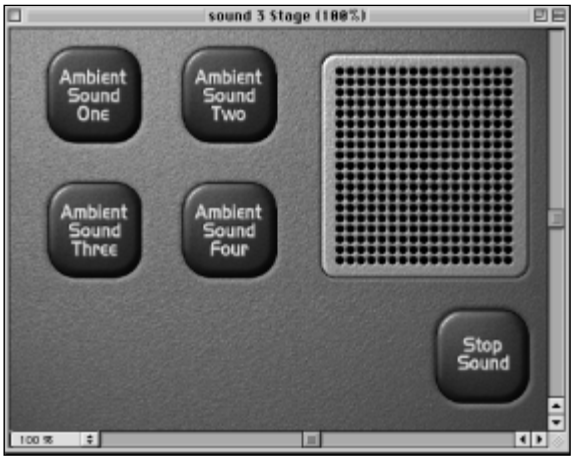


Figure 18-7: The Stop Sound button replaces the gray plug.

3. Now, drag the B_button state change behavior onto it. When the dialog box opens, choose `stopSound up` for the up state and `stopSound down` for the down state.
4. Create a new behavior, as shown in Figure 18-8, and apply this behavior to the Stop Sound button. This behavior loops through all the sound channels and turns the puppeting off, killing the sound.

Figure 18-8: The violently named KillSound handler

5. Play the movie and click several of the sound buttons, and then click the Stop Sound button to kill all the sounds at once. Save this movie as **sound 3.dir**.
6. Now open the B_sound button script and make the changes to the `mouseUp` handler, as shown in Figure 18-9. This makes the buttons toggle between playing and stopping the sound. It tests to see if the sound is playing. If it is, then it unpuppets the sound; otherwise it plays the sound. Save this movie as **sound 4.dir**.

Figure 18-9: Turning the sound buttons into toggles

Additional Lingo controls for digital sound

With Lingo, you can do far more than just play and stop sounds. You can determine whether a specific sound channel is busy, enable and disable sounds, control the sound's volume, and cause a sound to fade in or fade out. Let's take a look at how you accomplish each of these controls.

Determining whether a sound channel is busy

The `soundBusy` function determines whether a sound channel is busy or not. We used it in the preceding exercise. If the sound channel specified is busy, the function returns 1. If the sound channel is not busy, the function returns the 0.

Before you check whether a sound is playing, give Director sufficient time (a delay) to begin playing the sound, or else the `soundBusy` function might return 0.

Practical uses of the `soundBusy` function include:

- Looping within a frame until a sound finishes and then enabling the playback head to move to the next frame. This works very effectively for a self-advancing linear slideshow of sounds and images within your movie.
- Using a single object to toggle on and off a sound that is playing.

Using soundBusy to advance to the next frame

You can use the silence that follows the completion of a sound as the trigger to advance the playback head to the next frame. Such an approach is useful when you create a linear slideshow of images that are synchronized to specific sounds or breaks in narration. You can effectively use this technique in the following situations:

- ✦ When sounds or narration correspond to specific frames of a Director movie — that is, when a specific frame has an image on the Stage that corresponds to a given segment of digital sound
- ✦ When you check for the completion of a sound by means of the `soundBusy` command
- ✦ For sounds that have natural breakpoints that you can edit into separate sound cast members, which then can play and complete (triggering the next frame and the next sound)

You can also use a single sound track with cue points, and then have the playback head advance to the next frame as each cue point is encountered. We discuss cue points in a later section.

The handler in Figure 18-10 demonstrates the `soundBusy` technique. It does not matter whether the sound was puppeted or is actually in the Score.

Figure 18-10: Looping on a frame until the sound in channel 1 has stopped

The net effect is to loop the playback head within the current frame (showing the desired image and playing the desired sound or narration). When the sound finishes, the playback head moves to the next frame. If you repeat this script over a series of frames, you create a slideshow of images that advance whenever the narration for the current slide finishes.

Adding a film loop and playing sound

In the next exercise, you add visual interest to the movie by switching a film loop for a stationary image. A film loop and sound together add a little more complexity to the project.

The best approach is to list what you want to occur in the movie and then convert the list into Lingo scripts. Here are the actions that we want to occur:

- ✦ Start a new sound as soon as the user clicks the Play Sound button.
- ✦ When the sound starts, the sprite for the levels indicator should switch members with the film loop.
- ✦ As long as the sound plays, the film loop is on the Stage.
- ✦ If the user clicks again, the film loop is swapped with the cast member named level indicators off.

To accomplish all this, apply the behaviors that you created in an earlier exercise. The behaviors are already in the movie, so you don't have to copy and paste them from the other movie. Then you need to create a new behavior that keeps track of the sound channel.

For the following exercise, use filmloop.dir in the EXERCISE:CH18 (EXERCISE\CH18) folder on the CD-ROM that accompanies this book.

Adding a Film Loop and Playing Sound

1. Open filmloop.dir (see Figure 18-11).

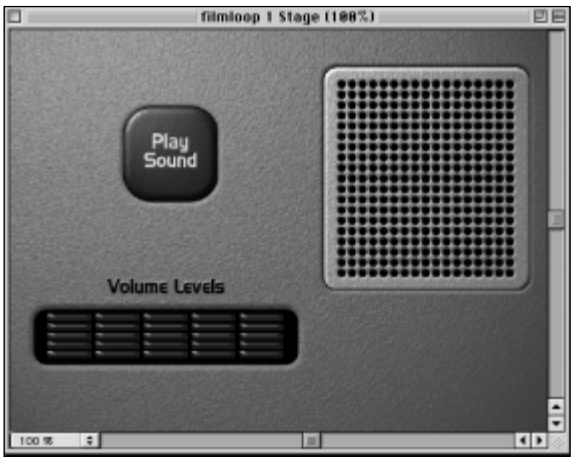


Figure 18-11: The filmloop.dir movie

2. Apply the behaviors B_sound button and B_button state change to the Play Sound button. Have the sound play in channel 1, and choose the appropriate up and down states, as you did earlier.

3. Ctrl+click (right-click) the level indicators sprite and choose "Script" to create a new behavior for this sprite.

4. Create the script shown in Figure 18-12. This script has three properties. The first property, `pSprite`, you are familiar with. The other two are references to the on and off states for the level indicators. The on state is a `filmloop`. In the `exitFrame` handler, sound channel 1 is checked with the `soundBusy()` function to see whether a sound is playing. If 1 is returned from the test (of sound channel 1), the `filmloop` is put into the member for its sprite; otherwise, the bitmap "level indicators off" is put in the member.

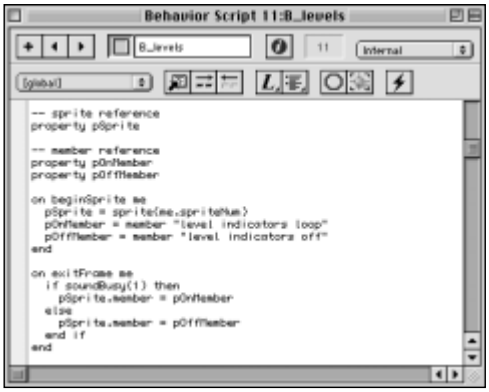


Figure 18-12: A behavior for the level indicators

5. Play the movie and turn the sound on and off. The level indicators go on and off as you would expect if they were responding to the sound playing. Save this movie as filmloop 2.dir.

Playing sound while the mouse is down

Using Lingo-related sound commands, the methods available to control sound playback are almost as endless as the digital sounds available. So far, you have learned to turn sound off and on by using a button or object. You can also cause a sound to play as soon as the mouse button is pressed or for as long as the mouse button is pressed down.

To have a sound play as soon as the mouse button is pressed, you need to add an `updateStage` following the statement with the `puppetSound` command.

In the handler in Figure 18-13, the sound cast member "LLOOP09" plays as long as the mouse button is pressed down. As soon as the user releases the mouse button, the `puppetSound 1, 0` command is executed, which stops the music. If the update Stage were not there, the sound would never start; it would be turned off before it began. The repeat loop has one statement, the keyword `nothing`. `nothing` in Lingo does exactly that — nothing. It is not necessary, but it adds some clarity to the repeat loop: You know the intent is to do nothing while the mouse button is pressed down.

On the CD-ROM

You can see the script from Figure 18-13 in mousedown.dir.

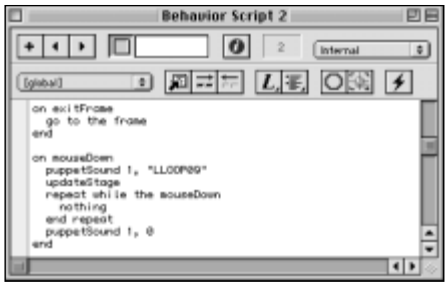


Figure 18-13: This handler plays a sound until the mouse button is released.

Playing sound on a rollover

Playing a sound when rolling the cursor over an element is often used. Having the element change in response to the mouse being over it is another common effect. In the next exercise, you do both.

On the CD-ROM

You can find rollovers 1.dir on the companion CD-ROM, in the EXERCISE:CH18 (EXERCISE\CH18) folder.

Activating Sound When the Mouse Rolls Over a Sprite

1. Open the movie rollovers 1.dir.
2. Create a new behavior in member 7, as shown in Figure 18-14. This behavior is very similar to the behavior for clicking a button to hear a sound. In fact, the properties and the `getPropertyDescriptionList` are the same. The difference is the `mouseEnter` and `mouseLeave` handlers. The `mouseEnter` puppets the sound when the cursor enters the bounds of the sprite; when it goes outside the bounds, it turns the sound off.
3. Apply this behavior to each of the four buttons. Make the first button play sound 1 in channel 1, make the second button play sound 2 in channel 2, and so on. Play the movie to make sure it works.
4. A nice effect is to have the state of the button change as well as the audio feedback. Create a new behavior in member 8, as shown in Figure 18-15. This behavior should also look familiar to the one in Figure 18-5. Instead of swapping members on the `mouseDown`, it swaps them when the cursor enters the sprite.

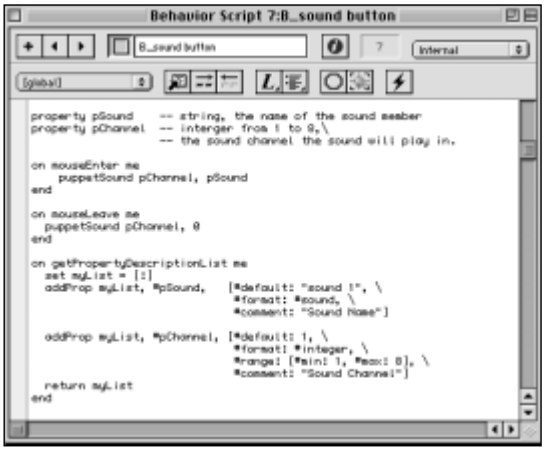


Figure 18-14: The behavior to activate the sound when the cursor rolls over a button.

Figure 18-15: This behavior changes the button sprite's member when the cursor rolls over it.

5. Apply this behavior to each of the buttons. Make sure you specify the appropriate up and down states.
6. Play the movie to test the behaviors. When you're finished, save the movie as **rollovers 2.dir**.

Playing random sounds

Whenever possible, you should add variety and a little friendly unpredictability to your movies. Moreover, although you want navigational buttons to always behave in the way the user expects them to, it doesn't hurt to add some surprises to the animation that occurs on the Stage or to the sounds generated by specific objects.

You can use the `random()` function to randomly choose various sounds to play when the user clicks the sound button. The handler in Figure 18-16 has a list of sounds hard-coded in it. The list is assigned to the variable `soundList`. Then `r` is assigned a random number from 1 to the number of items in the `soundList`. We could have made the code simpler by writing `r = random(3)`, but then each time you added or removed an item from the `soundList` you would also have to remember to change the number in the following line of code. Using the `count` function to count the number of items enables us not to have to worry about that. Additionally, it makes it more obvious in the code that `r` is related to the `soundList`. Lastly, the r-th item is retrieved from the list and that sound is puppeted.

Figure 18-16: A simple behavior to play a random sound on a mouseUp.

The handler in Figure 18-16 is fine in most cases, but it never hurts to do some error checking, especially when you are working on large projects with others who might be modifying the movie elements or code. Figure 18-17 adds some error checking. The first two statements are the same. After that, you test to determine whether the member actually exists. The number of member returns a –1 when a member is not found. If the member is not found, an alert pops up.

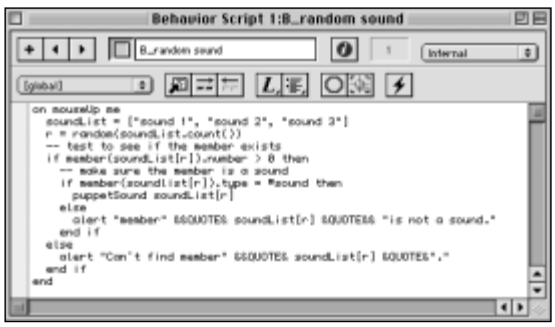


Figure 18-17: Before attempting to puppet a sound, add error checking.

Note in the alert statements that we used &&QUOTE&. You can write it all squished together or you can write it && QUOTE &. The double ampersands, as you might recall, concatenate two strings with a space between them. QUOTE is a Lingo constant that enables you to put a double quotation mark into a string. The single ampersand just concatenates two strings. This is true of any constants that you use in a string (such as &RETURN&). Sometimes the code is more readable when you chunk items together in this way.

Playing a list of sounds in one frame

Another way to use sound is to play a series of sound clips sequentially in a single frame. Conceivably, you could combine the multiple sound clips into a single clip and play it in one frame or over several consecutive frames. Because smaller sound clips load more quickly, however, it's often a good plan to play several sounds sequentially in a single channel. The following exercise is one approach to the task.

You can use the movie soundlist 1.dir from the EXERCISE:CH18 (EXERCISE\CH18) folder on the accompanying CD-ROM for the following exercise.

Playing Three Sounds Sequentially in One Frame

1. Open a new movie by choosing File ➪ New ➪ Movie, or press Command+N (Ctrl+N), or open soundlist 1.dir on the CD-ROM.
2. If you created a new movie, import any four sound files. (It doesn't have to be four, that's just a suggestion; it can be more or less.)
3. Double-click the Script channel in frame 1, and type the script shown in Figure 18-18. If you imported your own sounds, then `soundList` should be assigned their names instead of the ones shown. You can put as many or as few items as you like in this list.

   This behavior checks to see whether sound channel `pChannel` is busy; if it is not, it calls the `PlayNextSound` handler. `PlayNextSound` plays the sound indicated by `pTrack` and then increments `pTrack`. If `pTrack` exceeds the number of items in `soundList`, it is set back to 1.
4. Save the movie as **soundlist 2.dir**, and then rewind and play it. Note that nothing appears on the Stage when you play the movie, but the three sound clips play sequentially repeatedly. (They won't stop until you stop the movie.)
5. To make this behavior more flexible, you can add a `getPropertyDescriptionList` to it so that when you drag it into the Score, it asks you which channel you want to play the sound in (see Figure 18-19). This way, you won't need to go back into the code to change `pChannel`'s value. Figure 18-20 shows the changes necessary to do this.
6. Save this movie as **soundlist 3.dir**.

Figure 18-18: A frame script behavior that plays one sound after another.

Figure 18-19: A slider is created so that the user can choose a sound channel from 1 to 8.

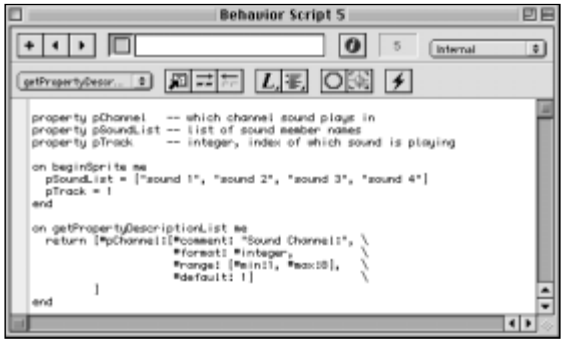


Figure 18-20: Adding a getPropertyDescriptionList handler

Playing linked files with Lingo

Director has several ways to incorporate sound into a movie. You can import, insert, or link sounds. *Importing* embeds the sound in the movie, and *inserting* links the sound. There is an important difference in the functionality of an imported/linked versus inserted sound. Director can only insert Shockwave Audio (SWA) files; when you do that, you have all of the Shockwave audio commands available to you, as shown in Figure 18-21. Director can now import Shockwave Audio (as well as other sound formats, such as WAV, AU, and AIFF). When a Shockwave Audio file is imported, it is treated just like other imported sounds: You can no longer use the Shockwave Audio Lingo on it. If you test the type of the member, it returns `#sound` (as opposed to `#swa`, for an *inserted* Shockwave Audio).

Figure 18-21: The Sound and Sound Members/Shockwave Audio Lingo menus

Another way to play an external (linked) sound file is by using the `sound playFile` Lingo command. Director can work with AIFF, SWA, AU, or WAV files on either a Macintosh or a Windows system. A nice aspect of using the `sound playFile` command is that the sound does not have to be in the movie. The syntax is:

```
sound playFile channel, pathToSound
```

You replace the `channel` parameter with a channel number, and the path to the file to be played replaces the `pathToSound` parameter. If the path is a URL, it's not a bad idea to use `preloadNetThing` or `downloadNetThing` to download the file before playing it. You cannot use the `sound playFile` command to play an embedded cast member.

Using the `puppetSound` command loads the sound file into RAM and then begins playing it, but the `sound playFile` command streams the audio from disk and begins playing it immediately. Remember that your computer cannot read two disk files simultaneously. If a Director movie is reading a sound file using the `sound playFile` command, the movie cannot concurrently play digital video, load cast members into memory, or conduct any other disk read/write activity.

For example, the following command plays the sound file "jazz" from an external sound file in channel 2:

```
sound playFile 2, "jazz"
```

If you don't place the external sound file in the same folder or directory as the movie, you must include the file's path preceding its filename. For example, if the "jazz" sound file is in a folder called SOUNDS on your local hard drive, the `sound playFile` command requires one of the following pathname formats:

- On a Macintosh system, assuming the local hard drive is Macintosh HD, the pathname would be indicated like this:

  ```
  sound playFile 2, "Macintosh HD:sounds:jazz"
  ```

- On a Windows system (assuming the local hard drive is C:), the pathname would be indicated like this:

  ```
  sound playFile 2, "c:\sounds\jazz"
  ```

- If the sound is a linked file, you can use its `fileName` property to play it, as follows:

  ```
  sound playFile 2, the fileName of member "jazz"
  ```

Controlling the Sound Volume

You can control the sound in a movie in one of three ways:

- By adjusting the sound level for all sounds playing (using the `soundLevel` property)
- By adjusting the volume of a specific sound in a specific channel (using the `volume of sound` property)
- By causing a sound to fade in or fade out over a specific time period (using the `sound fadeIn` and `sound fadeOut` commands)

Using the soundLevel property

The `soundLevel` property enables access to the sound hardware on your computer to adjust its overall volume. With the `soundLevel` property, you can set or determine the volume of the sound that is played through your computer's speakers, from a value of 0 (muted) to 7 (maximum volume).

The syntax for setting the sound level (volume) is as follows:

```
set the soundLevel to n
```

You replace `n` with an integer value from 0 to 7 representing the sound volume.

For example, to set the volume to the maximum level, you can use the following instruction:

```
set the soundLevel to 7
```

In the following steps, you modify the soundlevel 1.dir movie to set the volume of a sound cast member played through your computer's speakers.

The file soundlevel 1.dir is on the companion CD-ROM in the EXERCISE:CH18 (EXERCISE\CH18) folder.

Setting the Volume Level of Sound

1. Open the movie soundlevel 1.dir. Figure 18-22 shows soundlevel 1.dir. You need to add three scripts to the movie to make it functional. There are already three scripts in the movie that should be familiar to you in members 2 through 4. They are the behaviors for the sound button, the button state change, and a `go to the frame`. They have already been applied to the appropriate sprites.
2. The up and down buttons already have the "B_button state change" applied to them, so they highlight when you click them, but they do not do anything. Create a new behavior in member 5, as shown in Figure 18-23. This behavior increases the sound by one increment each time the button is clicked. If the `soundLevel` is already at 7, it has no effect.
3. Apply this behavior to the Increase button.
4. Create another behavior in member 6, as shown in Figure 18-24. This is the behavior for decreasing the sound. It is not as simple as the behavior to increase sound. When you try to set the `soundLevel` to a number less than zero, Director sets the `soundLevel` to 7, which is probably not a desirable effect — lowering the sound and it suddenly blares it. That's why you need to test the `soundLevel` before setting it. If it is above 0, lower it; otherwise, do nothing.

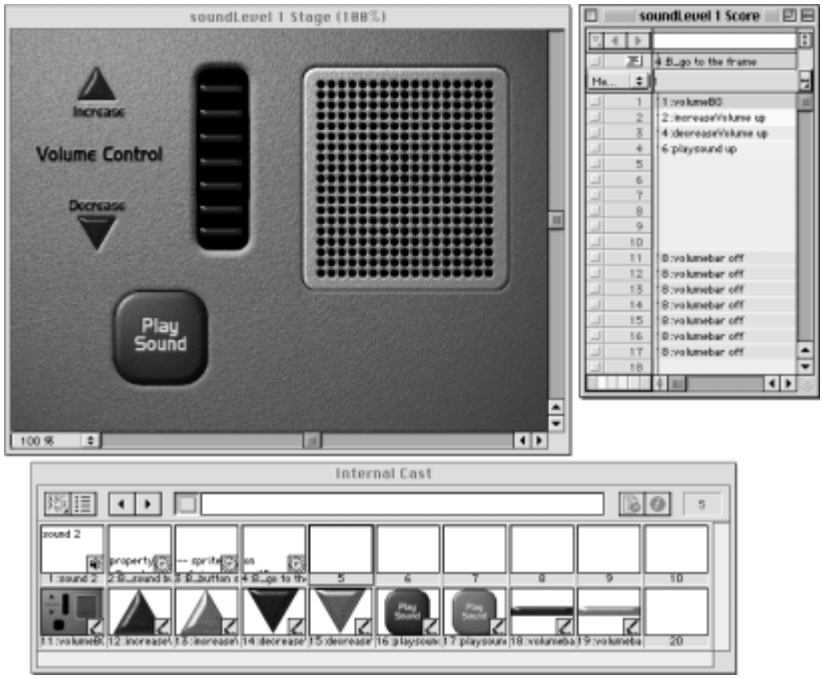


Figure 18-22: The soundlevel 1.dir movie

Figure 18-23: Increasing the soundLevel

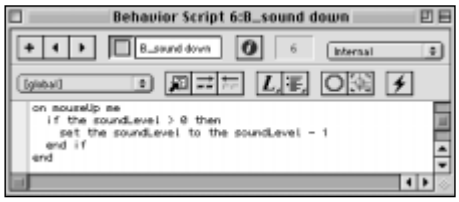


Figure 18-24: Decreasing the soundLevel

5. Apply the behavior to the Decrease button.

6. Now for the fun part! Create a new behavior in member 7, as shown in Figure 18-25. This behavior is for the volume bars. In the `exitFrame`, it checks to see what the `soundLevel` is. If it is over a certain amount, it switches the member of the sprite to which it is applied. The number of the sprite minus an offset determines the amount. We are taking advantage of the fact that all seven bars are laid out in a group in the Score (see Figure 18-22). Because the sprites begin in channel 11, if you subtract 10 from the first one in channel 12 you get 2. If the `soundLevel` is greater than or equal to 2, then this member is switched on. The sprite in channel 11 is also on because 2 > 11 – 10 (`the soundLevel >= (me.spriteNum - pOffset)`), but the sprites in channels 13 to 17 are not on.

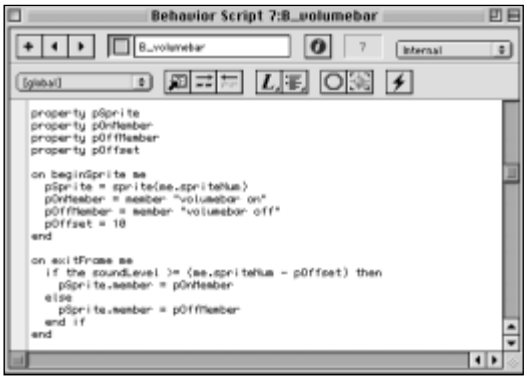


Figure 18-25: Creating a behavior for the volume bars

7. Save the movie as **soundlevel 2.dir**. Play it to test. The volume bars work whether or not a sound is playing, because they detect the `soundLevel`, not whether sounds are playing.

Using the volume of sound property

Like the `soundLevel` property, the `volume of sound` property enables you to control volume — in this case, the volume of a specific sound channel. Further, you can control the volume with a greater degree of precision because you have a wider range of acceptable values for this property: from 0 (muted) to 255 (the maximum volume). The volume of sound is relative to the `soundLevel`, so if channel 1 gets set to maximum volume and the `soundLevel` is at 0, you won't hear anything.

In the following statement, the sound in channel 1 is set to the maximum volume:

```
set the volume of sound 1 to 255
```

The following statement mutes the sound that is playing in channel 2:

```
set the volume of sound 2 to 0
```

Because this property can be determined or set by the user, it is often adjusted by using a slider to control the volume of sound in a movie. In the following steps, you modify the volume of sound 1.dir movie so that the user can control the volume of sound played in a specific sound channel.

In the completed slider movie, shown in Figure 18-26, you are able to click the Channel One button and manipulate the volume of that channel with the Volume Control slider. The text members below the Channel buttons give feedback as you move the slider. When you click the Play Sound button, it plays one of two sounds, one for sound channel 1, and another for sound channel 2. It requires the creation of four behaviors and one movie script. Because there is only one slider and two channels being modified in this movie, the slider needs to know which channel has the focus (which one is the target). For this we create a global variable called `gTargetChannel`, which is accessed by many of the behaviors.

Figure 18-26: The slider movie

The file slider 1.dir is on the companion CD-ROM in the EXERCISE:CH18 (EXERCISE\CH18) folder.

Setting the Volume Level for a Sound Channel

1. Open slider 1.dir.

2. Create a new behavior in member 3, as shown in Figure 18-27.

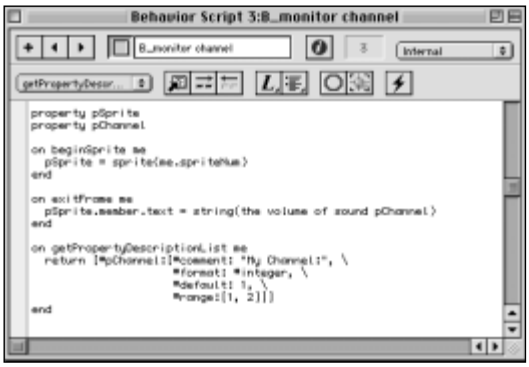


Figure 18-27: A behavior for the text members to monitor the sound channels

3. Apply the behavior to both text member sprites (6 and 7), which are below the Channel One and Channel Two buttons, respectively. This behavior monitors the volume of a sound channel. When you apply the behavior to a sprite, the Parameters dialog box pops up and you can choose between channel 1 or 2. If you wanted more options in the My Channel pop-up dialog box, then you would add those to the `#range` list in the `getPropertyDescriptionList()` function. Each time the `exitFrame` handler executes, it sets the text of the member in the sprite to the value of `the volume of sound` for the channel it monitors. The text of a member must be a string, so the `string()` function is used to convert the integer to a string.

4. Create a new behavior in member 3, as shown in Figure 18-28.

Figure 18-28: The behavior of the Channel One and Channel Two buttons

5. Apply this script to the Channel One and Channel Two button sprites (sprites 3 and 4, respectively). This is the longest script in the movie, but several of the elements should be familiar to you by now. The purpose of this script is to set the `global gTargetChannel` when the button is clicked. It then sends a message to the slider sprite so that it can deal with the change of channels.

   The `beginSprite` handler initializes all of the properties except `pChannel`. `pChannel` is given a value when you apply the behavior to a sprite. `pSprite` is the reference to the sprite. `pOnMember` and `pOffMember` are references to members for on and off, respectively. `pSliderSprNum` is the number of the sprite channel that the slider is in. This behavior needs it so that it can communicate with it. You could put `pSliderSprNum` in the `getPropertyDescriptionList()` function and set the value when you apply the behavior. We chose to put it in the code so that if we need to change its value, we only have to make the change in one place.

   The `mouseDown` handler sets the `global gTargetChannel` to its channel (held in `pChannel`). Then the slider sprite is sent a message to let it know the target has changed. We'll explain why this is necessary in the slider script.

   The `exitFrame` just tests the value of `gTargetChannel`. If `pChannel` is the same as `gTargetChannel`, the button lights up; otherwise, it stays dim.

6. Create a new behavior in member 5, as shown in Figure 18-29.

Figure 18-29: The behavior for the slider

7. Apply this behavior to the slider sprite (sprite 10). This behavior makes the sprite move like a sliding switch. It uses another invisible sprite to constrain its movement; the invisible sprite is just a rectangle shape with no stroke or fill. It assumes the constraining sprite is in the channel just below its sprite channel. Another way to constrain movement is to do it through code and not use another sprite. That way has advantages in not relying on two sprites, but it makes for code that is more complex.

The `beginSprite` handler sets `pSprite` and then sets the `moveableSprite` property of `pSprite` to TRUE. This enables the user to click and move the sprite at will. You don't want a user moving it all over the screen, so it is constrained to the bounds of the sprite right below it.

The `exitFrame` handler has one long statement. It sets the volume of a sound channel based on `gTargetChannel`'s value. The level the sound is set to is based on the location of the sprite minus the constraining sprite's location, multiplied by 255.0 (the maximum sound level) divided by the width of the constraining sprite. 255 is made a float (255.0); otherwise, the division would be integer division. So, every time the `exitFrame` is called, the volume of the current sound channel is changed.

The `ChangeTarget` handler is also a single long statement. It takes the value of the current channel and sets the sprite's locH. The calculation is the reverse of what is done in the `exitFrame`. `ChangeTarget` is called from "B_channel buttons" when the button is clicked. Why is this necessary when the `exitFrame` is always monitoring the value of `gTargetChannel`? When you clicked the other Channel button, the value of the field would be set to the value of the other button. Try commenting out the `sendSprite(pSliderSprNum, #ChangeTarget)` in the "B_channel buttons" behavior when the movie is finished for a clearer idea of what happens.

8. Create a new behavior in member 6, as shown in Figure 18-30.

Figure 18-30: An additional behavior for the Play Sound button

9. Apply this behavior to the Play Sound button (sprite 12). This behavior plays the sound based on the value of `gTargetChannel`. The sounds are named "sound 1" and "sound 2". This enables you to simply concatenate "sound" with `gTargetChannel`. The double-ampersand is used to concatenate with a space in between the two strings. Sprite 12 already has one script, "B_button state change", applied to it.
10. Last, but not least, create a movie script (not behavior) in member 7, as shown in Figure 18-31. All this script does is give the `global gTargetChannel` an initial value.
11. Save the movie as **slider 2.dir** and play it.

Figure 18-31: The global gTargetChannel is initialized in the startMovie handler.

Controlling fade-in/fade-out

The third way to control sound volume is to cause the sounds to fade in and fade out as needed. This feature is useful when you want to smooth an otherwise abrupt beginning of a musical clip, or to fade out background music at a specific point in your movie. Fade-outs can also reduce the volume in one channel while the volume of narration or sound effects increases in another channel.

You use two companion commands to handle the fade-in and fade-out of sound in a specific sound channel: `sound fadeIn` and `sound fadeOut`.

The `sound fadeIn` command increases the volume of a sound until it reaches an optimum level over a specified period. The time specified is measured in ticks, with 60 ticks equal to a second. Any fade-in you specify continues at the predetermined rate until the specified time has elapsed, or until the sound in the specified channel changes or stops.

To fade in a sound in channel 2 over a five-second period, for example, you use the following instruction:

```
sound fadeIn 2, 300
```

Here's another version of this instruction:

```
sound fadeIn 2, 5*60
```

The advantage of the second method is that you can easily (without any mathematical effort) determine the number of seconds (in this case, 5) that the sound uses to fade in.

When you specify the number of ticks, the fade-in occurs smoothly over the specified period. If you do not specify the number of ticks, however, Director calculates the default number of ticks as follows:

15 × 60 ticks / the Tempo setting in the first frame of the fade-in

The `sound fadeOut` command decreases the volume of a sound in a specified channel over a specified number of ticks. Similar to the `sound fadeIn` command, `sound fadeOut` fades the sound smoothly over the period you specify. If you do not specify the number of ticks, Director calculates the default number of ticks as follows:

15 × 60 ticks / the Tempo setting in the first frame of the fade-out

In the following example, the sound in channel 2 fades out over 3 seconds:

```
sound fadeOut 2, 180
```

Note that the `sound fadeIn` command should precede the `puppetSound` command; otherwise, you hear the initial sound at the default volume, then hear it drop to a muted level, and then hear it fade in.

On the other hand, the `sound fadeOut` command should follow the `puppetSound` command, so the initial sound is played at the current volume, which then drops (over the specified number of ticks) to a muted level.

Preloading and Unloading Cast Members

Playing sound from large sound files can create delays. Director enables you to minimize (somewhat) these delays by doing the following:

- ✦ Preloading one or more specific cast members
- ✦ Preloading all cast members required in one or more frames

Preloading cast members still takes time, and the amount of time required is contingent on the amount of RAM available, plus the size and number of the cast members that you want to preload. Nevertheless, there are advantages to using a preload of assets, because you can schedule the preload:

- ✦ While the system is normally idle
- ✦ When user interaction is not expected, such as while an instruction screen is displayed for the reader
- ✦ While a previously loaded animation is running

You can also add a screen message that alerts the user that work is going on behind the scenes.

You never want to leave the user trying to figure out whether a mouse click worked or the program has locked up and gone into cyberspace.

Preloading specific cast members used in one or more frames

You can use the `preLoad` command to load all the cast members required for one or more frames in your movie. For example:

- You can preload all cast members used in the current frame to the end of the movie by using the `preLoad` command without any parameters, as in the following:

  ```
  preLoad
  ```

- To load all cast members used in the current frame, through and including cast members used in frame 10, use the following command:

  ```
  preLoad 10
  ```

  When you include a single parameter, the `preLoad` command preloads frames from the current frame through and including the frame designated by the parameter.

- To load all cast members used in frames 10 through 15, use the following command:

  ```
  preLoad 10, 15
  ```

This command does not preload *only* the cast members in frames 10 and 15. It preloads all cast members in frames 10 *through* 15.

The preloading process terminates when your system's RAM (working memory) is full or when the specified cast members are loaded.

You can use label names (markers) with the `preLoad` command. If you use one label, Director preloads from the current frame through the frame indicated by the marker. If you use two label names, all cast members between the two label names are preloaded. In the following example, all cast members in the frames labeled "Start" (cast slot 1) and "Midway" (cast slot 40) are preloaded into memory:

```
preLoad "Start", "Midway"
```

Another variation is to have the `preLoad` command load all cast members from the current frame to the next frame with a marker. The command form is entered as follows:

```
preLoad marker (1)
```

Preloading specific cast members

You use the `preLoadMember` command to load one or more cast members that you specify. Unlike the `preLoad` command, `preLoadMember` gives you greater control over selecting specific cast members that you want to place in RAM before they are normally loaded. Use this technique to load a cast member and then gain quicker display or playback of the member (sound, film loops, animation, and so on) at a specific point in the movie — without pausing to load the cast member. For example, you can do the following:

- You can preload all cast members in the entire movie by using the `preLoadMember` command without any parameters, as in the following:

  ```
  preLoadMember
  ```

- To load only cast member 20, use this command:

  ```
  preLoadMember 20
  ```

- To load all cast members (in the Cast window) beginning with cast member number 20 and up through 30, use this command:

  ```
  preLoadMember 20, 30
  ```

  Note that this command form does not preload only the cast members 20 and 30. It preloads all cast members 20 through 30.

The preloading process terminates when your system's RAM (working memory) is full or when the specified cast members are loaded.

You can use cast member names, as well, with the `preLoadMember` command. If you name a specific cast member, it alone is preloaded into memory. If you use two cast member names, all cast members between and including the first and last named cast members are preloaded.

In the following example, all cast members between "Circus" and "Reg1" are preloaded into memory.

```
preLoad "Circus" "Reg1"
```

Unloading cast members

Preloading cast members can help eliminate loading delays and provide smoother playback of a movie, but eventually every computer system runs out of memory. To free up memory for different cast members, use the `unLoad` command.

- You can unload all cast members in all frames except the current one by executing the following instruction:

  ```
  unload
  ```

- You can also instruct Director to unload files from a specific frame (frame 25, in this example), by using the following instruction:

  ```
  unLoad 25
  ```

- Similar to the `preLoad` command, you can unload all cast members in a range of files (say, 25 to 35), by using this instruction:

  ```
  unLoad 25, 35
  ```

You can use cast member names, as well, with the `unLoad` command. If you name a specific cast member, it alone is unloaded from memory. If you use two cast member names, all cast members between the first and last named cast members are unloaded. In the following example, all cast members between and including "Circus" (cast slot 15) and "Reg1" (cast slot 18) are unloaded from memory:

```
unLoad "Circus", "Reg1"
```

You do not need to unload memory routinely, because Director automatically unloads the least recently used cast members based on the cast member's Unload setting. The `unLoad` command is intended for special circumstances in which there is a lag in the startup of an animation, the redrawing of the Stage, the playback of a digital video, or the playback of a sound, due to the lack of sufficient working memory.

Director Does Beatnik

Director now supports Beatnik via the Beatnik Lite Xtra (you can upgrade to the Beatnik Pro Xtra). Director 7 automatically installed it, but Director 8 requires you to install it manually from the Director 8 CD-ROM.

Creating a Beatnik object

Creating a Beatnik object is just like creating an object with other Xtras. You can follow along in the Message window if you like. Make sure you first open the groovoid 1.dir movie, which you can find in the EXERCISE:CH18 (EXERCISE\CH18) folder on the CD.

```
gMusic = new(xtra "Beatnik")
```

That's it. You've created an object. By default, the object plays direct-to-soundcard (DTC). DTC mode plays very fast and uses very little CPU time. If you need to play other sounds in Director simultaneously, then you need to use the Macromedia Open Architecture (MOA). This mode is not as responsive. You cannot have two instances of different types; all objects at any one time must be either DTC or MOA. To initialize the object to use MOA, write the following code:

```
gMusic = new(xtra "Beatnik", TRUE)
```

After creating the instance (the object), you can set the slice size of the rendering buffers. If you do not, defaults are used based on the mode you are in. Larger `setSliceSize` values slow the response from the Beatnik Xtra, but they give you audio that does not break up. The default for the MOA mode is 12. It needs to be larger, because you are running two audio mixers (Beatnik's and Director's). If you are running in DTC mode, you get better performance and can set the slice to 1 or 2. Of course, you won't be able to use other forms of sound in Director. For best performance, use Beatnik instances as you need them in DTC mode and then destroy them by setting the variable to VOID, before you use another type of sound:

```
setSliceSize(gMusic, 1) --for DTC mode try values around 1 to 2
setSliceSize(gMusic, 12) --for MOA mode try values 11 to 13
```

Next, you need to tell the object the location of the sound bank it is going to use. In this case, use the one that comes with the Xtra, "patches.hsb" (which has a General Midi sound bank).

```
setSampleLibrary(gMusic, the moviePath & "Patches.hsb")
```

Now you need to initialize the Beatnik, which you do with the `setReady` command, as follows:

```
setReady(gMusic)
```

At this point, you are ready to play some sounds. If you're using a Mac, you might notice that this action takes a few seconds; this delay is reportedly being fixed for the Pro version and probably the next Lite version of the software.

Groovoids

Groovoids are sound effects built into the sound bank. There are 72 available for Beatnik Lite, and even more in the Pro version. You can get a complete list from the documentation.

Continuing with the instance created earlier (`gMusic`), here are a few Groovoids that you can try:

```
playGroovoid(gMusic, FALSE, "UI-Chimes")
playGroovoid(gMusic, FALSE, "UI-Click2")
playGroovoid(gMusic, FALSE, "Background-InfoPulse")
playGroovoid(gMusic, FALSE, "Fanfare-Sports")
playGroovoid(gMusic, FALSE, "Misc-CashRegister")
```

The first parameter is the object. The second is a Boolean value indicating whether the sound should loop. The third is the name of the Groovoid to play. Go ahead and play the groovoid 1.dir movie and try the different Groovoids. Click the Loop Sound? check box to have the Groovoid loop continuously (see Figure 18-32).

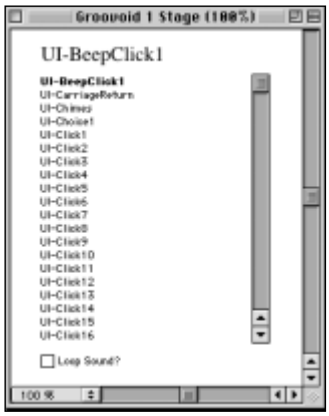


Figure 18-32: Give the Groovoids a try in the groovoid 1.dir movie.

Playing RMF Files

You can also play Rich Music Format (RMF) files. To use the idunno.rmf in the same folder as the groove 1.dir, type the following:

```
play(gMusic, FALSE, the moviePath & "idunno.rmf")
```

Just as with the `playGroovoid` command, the second parameter is a Boolean value indicating whether the sound should loop. The third is a path to the file. There are quite a few methods available for manipulating Beatnik sounds in real time. We'll touch upon some of the basics, and you can read more about them in the Beatnik Director Xtra — musicObject Lingo Handler Reference.

To adjust the volume of the sound, just type the following:

```
setVolume(gMusic, 50)
```

The volume can be set from 0 to 100. To add some reverb, you can type the following code:

```
setReverbType(gMusic, 5)
```

Reverb can be set from 1 to 6. To change the tempo, use the following syntax:

```
setTempo(gMusic, 60)
```

Tempo has a range of 0 to 499.

With Beatnik, you can make MIDI Controller Changes on the fly. You can control certain synthesizer settings of an RMF or MIDI file as the file is playing back:

```
setController(gMusic, 10, 7, 64)
```

The preceding code sets the volume of the drums in channel 10 to 64 (possible values 0 to 127). The number 7 indicates the controller, which, in this case, is the volume. The movie beatnik 1.dir puts all of these elements together. Play the movie and try the different buttons (see Figure 18-33).

Figure 18-33: The beatnik 1.dir movie

Click the Play idunno.rmf button. After a few seconds, you hear drums in the music. Try the Fade Drums Out and then the Fade Drums In buttons. The speed at which the drums fade is based on the movie's tempo (not the sound's tempo). Try speeding up or slowing down the sound's tempo. Clicking pause pauses the music. Clicking it again makes it resume from where it left off. Clicking Stop stops the music.

Another fun thing to try, while the music is playing, is to change the program. While the music plays, type the following into the Message window:

```
setProgram(gMusic, 0, 11) -- 11 is Vibraphone
setProgram(gMusic, 0, 25) -- 25 is Steel Guitar
setProgram(gMusic, 0, 14) -- 14 is Tubular Bell
```

Using this same command, you can also switch from the General Midi bank (0) of instruments to the Special Bank (1). You need to add another parameter to tell which bank to use. This additional parameter goes before the instrument parameter, as follows:

```
setProgram(GetMusic(),0, 1, 24) -- 24 is Acoustic Bass Gtr in
Special Bank
```

If we used the earlier `setProgram` commands, specifying the Special Bank would get the Special Bank instruments, as follows:

```
setProgram(gMusic, 0, 1, 11) -- 11 is Vibraphone 2
setProgram(gMusic, 0, 1, 25) -- 25 is Mute Guitar 2
setProgram(gMusic, 0, 1, 14) -- 14 is Reverse Bell
```

To switch back to the General Midi bank, playing the Steel Guitar, you write the following code:

```
setProgram(gMusic, 0, 0, 25)
```

Sound Channel Objects

Director 8 adds a new sound channel object and a great deal of functionality. The only price to pay is a little bit of complexity. The sound channel object actually comes in the form of a reference object, very similar to the object returned when you get the "ref" of a chunk of text (for a discussion regarding using the "ref" keyword, see Chapter 16). Instead of a reference to a chunk of text in a text member, with the sound channel object you have a reference to a sound channel. Try the following in the Message window:

```
put sound(1)
-- <Prop Ref 3 b423878>
```

The last chunk of data, `b423878`, will be different on your machine; in fact, it will be different for each movie. Fortunately, you don't need to concern yourself with it!

Open the movie sound_channel.dir in the EXERCISE:CH18 (EXERCISE\CH18) folder on the CD-ROM so you can follow along, trying the new commands from the Message window.

You can place a sound channel object reference into a variable to shorten your code and make it more understandable, as follows:

```
myChan = sound(1)
```

If you want to play a sound using the channel object, write the following code:

```
sound(1).play(member(1))
```

You should hear the sound start playing. This command is analogous to typing the following:

```
puppetSound 1, member 1
```

So, which seems friendlier? On one hand, you have a more intuitive sounding command, `play()`. On the other hand, you have much simpler syntax. You can use standard syntax to simplify things. Although this is a new command, it can still be constructed by using old syntax:

```
play sound 1, member 1
```

Because `play` is not a function, the parentheses are not necessary. Many of the help examples do not use this syntax, so we'll stick to dot syntax because it seems the way of the future. It is more likely that other folks will write code similar to the online help examples, not even aware of the old syntax. The new sound object has quite a few new commands and functions (see the table below). Of these, many should be familiar, such as `fadeIn()` and `fadeOut()`. These handlers have been in Director for years and now work with the sound channel object. Some, such as `queue()` and `setPlayList()`, are new and we'll talk about those next.

Sound Channel Object Handlers
sound(*chan*).breakLoop( )
sound(*chan*).fadeIn(< *milliseconds* >)
sound(*chan*).fadeOut(< *milliseconds* >)
sound(*chan*).fadeTo(*volume* < , *milliseconds* >)
sound(*chan*).getPlayList( )
sound(*chan*).isBusy( )
sound(*chan*).isPastCuePoint(*inOrString*)
sound(*chan*).pause( )
sound(*chan*).play( )
sound(*chan*).playNext( )
sound(*chan*).queue(*memberOrPlayList*)
sound(*chan*).rewind( )
sound(*chan*).setPlayList(*playList*)
sound(*chan*).showProps( )
sound(*chan*).stop( )

Queuing sounds

If you're an experienced Director developer, you'll have experienced many times sounds not playing as promptly as you had expected, especially on Windows machines. To play sounds quickly, you can *queue* them. A queued sound responds much more quickly. To queue member 1 and then play it, type the following:

```
sound(1).queue(member(1))
sound(1).play()
```

As the command name suggests, `queue` enables you to queue more than one sound. You can queue several sounds and when you tell the object to play, the sounds play consecutively:

```
sound(1).queue(member(1))
sound(1).queue(member(2))
sound(1).queue(member(3))
sound(1).play()
```

Perhaps at this point the complexity is starting to seem worth the extra work.

Creating a playList

As you may have noticed from the table, the `queue()` command also takes a list as a parameter. This list is a linear list of property lists (another reason we spent so much time on lists in Chapter 12!).

The sound channel object has many properties. You can view them in the Message window by typing the following:

```
put sound(1).showProps()
currentTime:          0.0000
elapsedTime:          0.0000
startTime:            0.0000
loopStartTime:        0.0000
loopEndTime:          0.0000
endTime:              0.0000
loopCount:            1
loopsRemaining:       0
member:
preLoadTime:          1500
pan:                  0
volume:               255
currentChannel:       1
channelCount:         0
sampleCount:          0
sampleRate:           0
sampleSize:           0
mostRecentCuePoint:   0
status:               0
```

Of these properties, the following can be used in the playList:

- `#member`: This property is the only mandatory one. It is a reference to a member, such as `member(1)` or `member("memName")`.
- `#startTime`: This is the time the sound will begin to play. The default is 0.
- `#endTime`: This is the time the sound will end. The default is the length of the sound.
- `#loopCount`: This is the number of times you want to play the sound.
- `#loopStartTime`: This is the time the sound will start when it loops. For example, if you specified that a sound should loop three times, and set the `#loopStartTime` to 1000, the first time the sound plays it will play all the way through, and the subsequent times it will start playing at 1000. The time is measured in milliseconds. If you want all iterations to begin at the same point, you need to set the `#startTime` to the same value.

- #loopEndTime: This is the time within the sound at which a loop ends, in milliseconds. It works the same way as #loopStartTime, except that it specifies the time the sound ends.
- #preloadTime: This is the amount of the sound that you want buffered before playback.
- #rateShift: This property is not officially supported, but it seems to work well. By using a negative number, you can slow down a sound, lowering its pitch. Entering a positive number speeds up the sound, raising its pitch.

These properties you can use in the playList. Before we get to setting the playList, take a look at another way to play a single sound by using a property list. The following sound will be looped three times:

```
sound(1).queue([#member: member 1, #loopCount: 3])
sound(1).play()
```

The preceding example simplifies our code by not having to queue member 1 three times. If you want to queue multiple sounds in one line of code, you need to set the playList property. The playList property is a linear list composed of property lists like the preceding one.

```
sound(1).setPlayList([[#member: member 1, #loopCount: 3],
[#member: member 2]])
sound(1).play()
```

Using setPlayList() is the only way to set the playList property. There is a corresponding function called getPlayList(), which returns the value of the playList. The playList returned will *not* include the currently playing sound. The following code shows setting the playList from the Message window and then getting it:

```
sound(1).setPlayList([[#member: member 2], [#member: member 3],
[#member: member 4]])

put sound(1).getPlayList()
-- [[#member: (member 2 of castLib 1)], [#member: (member 3 of
castLib 1)], [#member: (member 4 of castLib 1)]]
```

Panning sounds

Panning a sound is another of the cool new things you can do with a sound channel object. The pan property can be set with a value from –100 to 100 and changed while the sound is playing (unless the channel is doing a fade). A value of –100 plays the sound only from the left speaker, and a value of 100 plays the sound only from the right speaker. A value of 0 is a balance between the two.

Figure 18-34 shows a script that pans `sound(1)` based on the mouse location. For this example, we start `sound(1)` playing member 3 and loop it 3 times. The closer you move the mouse to the right side of the Stage, the more the sound only plays from the right speaker. As you move the mouse to the center, you hear it from both speakers. As you move the mouse toward the left of the Stage, the sound plays more and more from the left speaker.

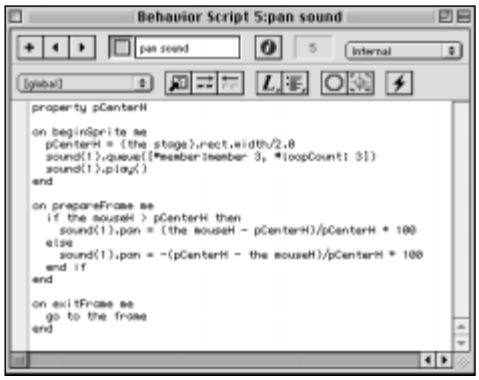


Figure 18-34: Panning a sound based on the mouse location

Summary

Among the things you learned in this chapter are the following:

- You can embed sound files in a Director movie and control the sound's playback by using the Score or the `puppetSound` Lingo command.
- You can also link an external sound file to a Director movie and play back the sound by using the `sound playFile` command.
- You can embed or link either AIFF (.AIF on the Windows platform) sound files or wave (.WAV) files.
- Lingo can control the fade-in and fade-out of a sound, adjust the volume, or preload and unload a sound.
- Director 8 comes with the Beatnik Lite Xtra, but you must install it manually.
- You can queue sounds for fast playback by using the new sound channel object.
- The sound channel object enables you to play all or part of sounds.
- You can pan a sound between speakers.

Our next chapter discusses the control of digital video via Lingo. It includes discussion of QuickTime VR commands.

✦ ✦ ✦