CHAPTER 19

Video Lingo

Digital video is one of the many media types that you can import into Director. Unlike static media types, such as text and bitmaps, digital media have the dimension of time. Within Director, a digital video movie need not just play back from beginning to end. Using Lingo, you can make the movie speed up, slow down, go backward, and more.

Toward the end of this chapter we'll approach how the subject of integrating QuickTime VR (QTVR) movies into your Director movie. With Lingo, you can couple the interactivity of a QTVR with your own Director-based interactivity.

✦ ✦ ✦ ✦

In This Chapter

Using Lingo to control digital video

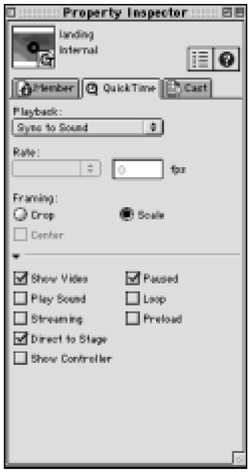
Creating your own movie controls

Using QTVR commands

✦ ✦ ✦ ✦

Using Lingo to Control Digital Video Properties

Through Lingo, you can get and set the properties of a Quick-Time cast member, just as you would in the QuickTime Xtra Properties (see Figure 19-1). If you want to try setting some of these properties, you can open the landingqt.dir movie in the EXERCISE:CH19 (EXERCISE\CH19) folder on the CD-ROM that accompanies this book. Many of the properties apply to the QuickTime member, but some apply to the actual sprite. As you work through this chapter, *make sure that the Director movie is playing when you manipulate the settings from the Message window*.



landing
Internal

Playback: Sync to Sound

Rate: 0 fps

Framing: Crop [✓ Scale]

☐ Center

☑ Show Video
☑ Paused
☐ Play Sound
☐ Loop
☐ Streaming
☐ Preload
☑ Direct to Stage
☐ Show Controller

Figure 19-1: The properties of a QuickTime cast member

Setting the video of member property

Remember that, even though you use a digital video clip, you don't *have* to display the video portion of the file. Each digital video member has a `video` property that can be set to 1 or 0 (TRUE or FALSE). Setting the property to 0 disables the display of video for that member. This can be helpful when you want to use a sound-only QuickTime movie in your project. To disable the display of video, use a statement similar to this:

```
set the video of member "landing" to 0
-- or dot notation
member("landing").video = 0
```

To enable the display of video from a digital video cast member, use a statement similar to this:

```
set the video of member "landing" to 1
-- or dot notation
member("landing").video = 1
```

The 1 and 0 values in the preceding examples can be replaced with TRUE and FALSE, respectively. This convention is consistent throughout Lingo. Using numeric values requires less typing, but the TRUE and FALSE keywords are more English-like and clearer to the casual Lingo reader. In the next two examples, we'll use the TRUE/FALSE keywords.

If you have a QuickTime movie sprite and you set the video to FALSE and the `directToStage` property is TRUE, it appears that the movie has stopped, but in actuality, the screen just needs to be refreshed. To do this, use the following after setting the video to FALSE:

```
set the stageColor to the stageColor
-- or
the stageColor = the stageColor
```

Note

If `directToStage` is set to FALSE, this refreshing is not necessary.

Setting the sound of member property

Just as you might want to play a digital video clip without the video, so might you want to skip the sound portion of the file and play only the video. Digital video has a sound property, which can be set to 1 or 0 (TRUE or FALSE). Setting the property to 0 will disable the sound for a video member. The following statement disables the sound in a digital video cast member:

```
set the sound of member "landing" to FALSE
-- or dot notation
member("landing").sound = FALSE
```

This code enables the sound in a digital video cast member:

```
set the sound of member "landing" to TRUE
-- or dot notation
member("landing").sound = TRUE
```

Pausing at the beginning of a video using Lingo

You can pause a digital video cast member on its first frame (which means you must use Lingo to play, rewind, and stop the video clip). This has the same effect as selecting the Paused check box option in the Properties dialog box. The following statement sets this property:

```
set the pausedAtStart of member "landing" = TRUE
-- or dot notation
member("landing").pausedAtStart = TRUE
```

This statement turns off the property:

```
set the pausedAtStart of member "landing" = FALSE
-- or dot notation
member("landing").pausedAtStart = FALSE
```

Setting the loop of member property

The following example statements show you how to cause a video clip to automatically loop and replay:

```
set the loop of member "landing" to TRUE
-- or dot notation
member("landing").loop = TRUE
```

Here’s how to turn off the automatic looping of a video clip:

```
set the loop of member "landing" to FALSE
-- or dot notation
member("landing").loop = FALSE
```

Cropping and centering video using Lingo

Setting the crop of a member to TRUE means that if the rect of the sprite is smaller than the rect of the member, it does not scale it to fit. The movie only shows what fits within the rect. If the `crop` is set to FALSE, and the sprite’s rect is smaller than the movie, then the image is scaled to fit the rect of the sprite.

The following statements show you how to enable a resized video clip to be cropped:

```
set the crop of member "landing" to TRUE
-- or dot notation
member("landing").crop = TRUE
```

To disable the cropping of a resized video clip, use this code:

```
set the crop of member "landing" to FALSE
-- or dot notation
member("landing").crop = FALSE
```

To center a cropped video clip within its resized bounding box, use a statement similar to the following:

```
set the center of member "landing" to TRUE
-- or dot notation
member("landing").center = TRUE
```

To disable this centering for the cropped video clip, use this type of statement:

```
set the center of member "landing" to FALSE
-- or dot notation
member("landing").center = FALSE
```

Playing a video direct to the Stage using Lingo

When working with QuickTime video on a Macintosh computer system, you can cause the digital video to be displayed direct to the Stage. In the following statement, we do this with a QuickTime video named landing:

```
set the directToStage of member "landing" to TRUE
-- or dot notation
member("landing").directToStage = TRUE
```

Here's the statement to turn off this property:

```
set the directToStage of member "landing" to FALSE
-- or dot notation
member("landing").directToStage = FALSE
```

Remember that `directToStage` being set to TRUE draws the movie over any other sprite, even ones in higher sprite channels. This gives the best performance for the movie. If you need sprites to appear over the movie, set `directToStage` to FALSE. This setting reduces the performance of the movie's playback and may adversely affect how smoothly the movie plays, especially on slower computers.

Displaying a controller using Lingo

You can use Lingo to display or hide a controller when using QuickTime movies on either the Windows or Macintosh operating system. To display the digital video controller, use the following code:

```
set the controller of member "landing" to TRUE
-- or dot notation
member("landing").controller = TRUE
```

To hide the digital video controller, do the following:

```
set the controller of member "landing" to FALSE
-- or dot notation
member("landing").controller = FALSE
```

Establishing the frame rate for digital video using Lingo

The `frameRate of member` property establishes the frame rate at which a digital video cast member plays. The settings you establish with this property correspond to the settings in the Rate and fps boxes in the Properties dialog box. These settings are available only if you elect to play every frame, as opposed to selecting the Sync to SoundTrack option. When the Rate field is set to Fixed, Director uses the value in the fps field to establish the frame rate.

To set these values using Lingo, you can use a statement similar to this:

```
set the frameRate of member "landing" to 30
-- or dot notation
member("landing").frameRate = 30
```

When you set the frame rate to a value between 0 and 255, Director equates this with a Fixed setting in the Rate field and assumes that the numeric value is for frames per second. Although the `frameRate of member` property can range up to 255, in most cases, you only use values in the 10 to 30 fps range.

You can also use the `frameRate of member` property to set the selection to the Normal rate setting in the Properties dialog box by using this statement:

```
set the frameRate of member "landing" to -1
-- or dot notation
member("landing").frameRate = -1
```

Another option is to use the `frameRate of member` property set to the Maximum rate setting in the Properties dialog box, which plays every frame as fast as possible, by using the following statement:

```
set the frameRate of member "landing" to -2
-- or dot notation
member("landing").frameRate = -2
```

Enabling the preload of video using Lingo

The following statement enables a video clip to be preloaded into memory:

```
set the preLoad of member "landing" to TRUE
-- or dot notation
member("landing").preLoad = TRUE
```

This statement disables the preload:

```
set the preLoad of member "landing" to FALSE
-- or dot notation
member("landing").preload = FALSE
```

Using Lingo to Create Other Controls

Although you can show the QuickTime controller, there may be times when you want to create your own controls. Creating your own controls enables you to decide what the interface looks like. It also enables you to decide what elements you want in your interface. More importantly, you can add controls that are not in the QuickTime controller's interface. If you didn't open it earlier, open the movie landingqt.dir. You can add functionality to the movie's buttons, which are shown in Figure 19-2.

Before leaping into the code, let's consider the problem. There are eight buttons. What should they do?

- **Rewind:** Rewinds the video
- **Backward:** Plays the video quickly in reverse
- **Stop:** Stops the video
- **Play:** Plays the video
- **Forward:** Plays the video quickly forward

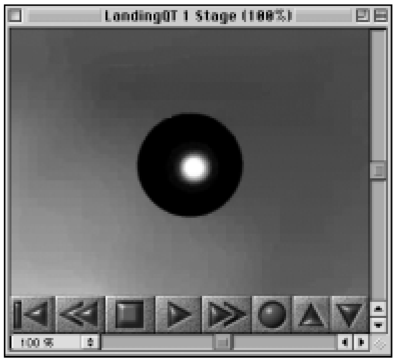


Figure 19-2: Creating your own interface for a QuickTime sprite

These first five buttons all work together. If one is activated, the others are deactivated. You need to consider this in your code. They should stay deactivated until one of the others is clicked.

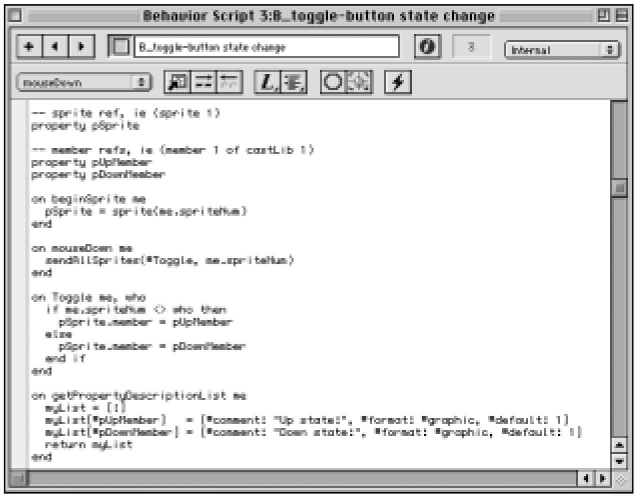
The next button is the Sound button. It should light up when the sound is on. It should also toggle the state of the sound when it is clicked.

Finally, there are the Volume up and Volume down buttons. They need to increase and decrease the sound, respectively. They should check when they are within the 0 to 255 bounds of the volume, to prevent it from being distorted[GSL1]. These buttons only become highlighted when clicked.

Creating the toggle behavior

Already included in the movie is a `go to the frame` behavior and the B_button state change behavior from Chapter 18. Toggling the button state is very similar, so it is easier to just duplicate the B_button state change behavior and modify it by following these steps:

1. Select member 2 (B_button state change) and choose Edit ➪ Duplicate.
2. Open the duplicated member (member 3 in the cast).
3. Delete the `mouseUp` and `mouseUpOutside` handlers.
4. Change the `mouseDown` handler and add the `Toggle` handler, as shown in Figure 19-3.


Behavior Script 3:B_toggle-button state change
B_toggle-button state change
3
Internal
mouseDown

```
-- sprite ref, ie (sprite 1)
property pSprite

-- member refs, ie (member 1 of castLib 1)
property pUpMember
property pDownMember

on beginSprite me
  pSprite = sprite(me.spriteNum)
end

on mouseDown me
  sendAllSprites(#Toggle, me.spriteNum)
end

on Toggle me, who
  if me.spriteNum <> who then
    pSprite.member = pUpMember
  else
    pSprite.member = pDownMember
  end if
end

on getPropertyDescriptionList me
  myList = [:]
  myList[#pUpMember]   = [#comment: "Up state:", #format: #graphic, #default: 1]
  myList[#pDownMember] = [#comment: "Down state:", #format: #graphic, #default: 1]
  return myList
end
```

Figure 19-3: Making a toggle behavior

The Toggle message is sent when the sprite is clicked. It is sent to all sprites, along with the sprite number of the sprite clicked. The `Toggle` handler compares the `who` parameter with its `spriteNum`. If the sprite is clicked, it sets its member to the down state. If it is not, it sets its member to the up state. Apply this behavior to the first five buttons.

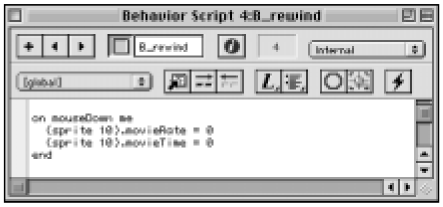
Rewinding the video

To rewind a QuickTime sprite, you need to change two properties. The first is the `movieRate`; the `movieRate` is how fast the movie is going. Because we want the movie to stop after it is rewound, it needs to be set to 0. The `movieTime` of a QuickTime sprite is the time index of where it is in the movie. We want it to be at the beginning, so it needs to be set to 0 as well.

Create a new behavior, as shown in Figure 19-4. The QuickTime video is in sprite 10. Apply this behavior to the Rewind button.

Tip

For the sake of clarity, we use the literal 10 in all of the behaviors that follow. If we were doing this in a real project, we'd probably use a movie script function to return the value instead of explicitly putting the number in ten different scripts. This would make changes easier. Using a global is another option, but then you also need to declare it wherever you use it.

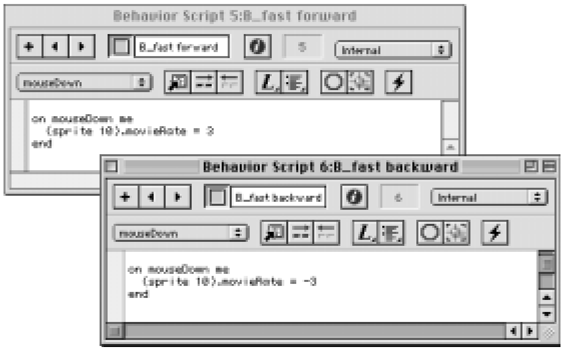


```
on mouseDown me
  (sprite 10).movieRate = 0
  (sprite 10).movieTime = 0
end
```

Figure 19-4: The Rewind behavior

Fast Forward and Reverse

Fast forward and reverse are essentially the same behavior; the only difference is whether you are setting the `movieRate` to a positive or a negative. Setting the `movieRate` to 3, as shown in Figure 19-5, makes the movie play at three times normal speed.


Behavior Script 5:B_fast forward
B_fast forward
5
Internal
mouseDown
Behavior Script 6:B_fast backward
B_fast backward
6
Internal
mouseDown

```
on mouseDown me
  (sprite 10).movieRate = 3
end
```

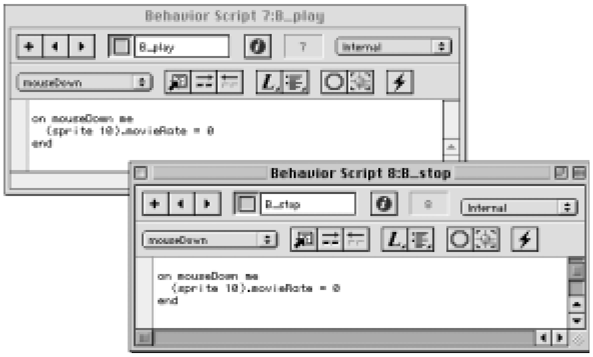
```
on mouseDown me
  (sprite 10).movieRate = -3
end
```

Figure 19-5: The Fast Forward and Reverse behaviors

Create the behaviors as shown in Figure 19-5. Apply B_fast forward to the Fast Forward button. Apply B_fast reverse to the reverse button.

Playing and stopping the video

Playing and stopping a video sprite is a simple matter of setting the `movieRate` to 1 or 0. Create two behaviors as shown in Figure 19-6. Apply B_play to the Play button and B_stop to the Stop button.



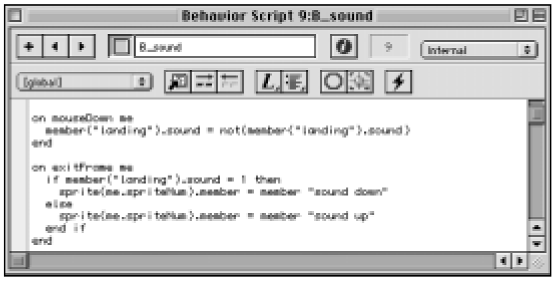
```
on mouseDown me
  (sprite 10).movieRate = 0
end
```

```
on mouseDown me
  (sprite 10).movieRate = 0
end
```

Figure 19-6: The Play and Stop behaviors

Turning the sound on and off

The green circle on the interface indicates whether the sound is ON or OFF. Create the behavior shown in Figure 19-7 and apply it to the Sound button. This script constantly polls the sound property of member "`landing`" and sets the member accordingly. The `mouseDown` simply toggles the value returned by the `sound of member` property by using the `not` keyword.

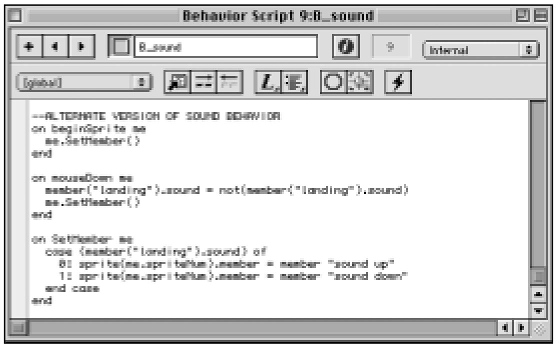


```
on mouseDown me
  member("landing").sound = not(member("landing").sound)
end

on exitFrame me
  if member("landing").sound = 1 then
    sprite(me.spriteNum).member = member "sound down"
  else
    sprite(me.spriteNum).member = member "sound up"
  end if
end
```

Figure 19-7: The Sound button behavior

Another way to achieve the same result is to change the member only once, on the `mouseDown`, as shown in Figure 19-8. This might seem more efficient than Figure 19-7, but you won't notice any speed difference. Director is very smart in setting the member of a sprite; if the member is already there, you get no speed hit by trying to set it again. The other thing you need with this method is a `beginSprite` handler to set the initial value. A beneficial side effect of the version in Figure 19-7 is that if some other script changes the sound of the member, the button still lights up. The script in Figure 19-8 would not be aware of the change.



```
--ALTERNATE VERSION OF SOUND BEHAVIOR
on beginSprite me
  me.SetMember()
end

on mouseDown me
  member("landing").sound = not(member("landing").sound)
  me.SetMember()
end

on SetMember me
  case (member("landing").sound) of
    0: sprite(me.spriteNum).member = member "sound up"
    1: sprite(me.spriteNum).member = member "sound down"
  end case
end
```

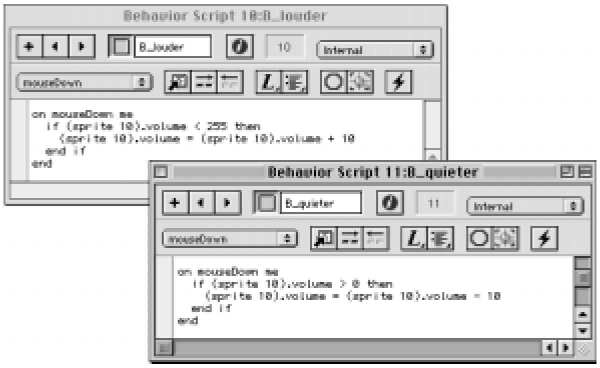
Figure 19-8: Changing the member of the sprite in the mouseDown handler

Changing the volume of a QuickTime sprite

Digital video uses `the volume of sprite` property to control the sound level of the sound track. You can set `the volume of sprite` to any value between 0 and 255, but any value lower than 1 mutes the sound. The following statement sets the volume at its highest level (you can set the volume higher than 255, but doing so distorts the sound):

```
Set the volume of sprite 10 to 255
-- or dot syntax
sprite(10).volume = 255
```

Create the behaviors shown in Figure 19-9. Apply each behavior to the appropriate button. You also need to apply the B_button state change behavior to each of these buttons so that they are highlighted when clicked.



```
on mouseDown me
  if (sprite 10).volume < 255 then
    (sprite 10).volume = (sprite 10).volume + 10
  end if
end
```

```
on mouseDown me
  if (sprite 10).volume > 0 then
    (sprite 10).volume = (sprite 10).volume - 10
  end if
end
```

Figure 19-9: The volume-changing behaviors

That about does it! Now save the movie as **landingqt2.dir** and play it.

Track-related Lingo

QuickTime movies are made up of different tracks. Those tracks could be `#video`, `#sound`, `#sprite`, `#music`, and `#text`. To count the number of tracks in a movie, use the `trackCount()` function, as follows:

```
put trackCount (member "landing")
-- 3

-- or dot notation
put member("landing").trackCount
-- 3
```

After knowing the number of tracks, you can learn what type they are, as follows:

```
put trackType(member "landing", 1)
-- #video
put trackType(member "landing", 3)
-- #sound

-- or dot notation
put member("landing").trackType(1)
-- #video
put member("landing").trackType(3)
-- #sound
```

The `trackCount()` and `trackType()` functions work for both members and sprites.

As shown earlier, you are able to turn the video and sound of a member ON and OFF. Having knowledge of the tracks gives you even more control. Using `setTrackEnabled`, you can turn individual tracks ON and OFF, as shown here:

```
setTrackEnabled(sprite 10, 1, FALSE)
-- or dot notation
(sprite 10).setTrackEnabled(1, FALSE)
```

This code turns the video track off for the QuickTime movie in sprite 10. This property has the same effect as setting the video of a QuickTime member to FALSE, for that particular track.

Tracks can be varying lengths of time. Using `trackStartTime()` and `trackStopTime()`, you can get this information from either sprites or members, as follows:

```
put trackStopTime(member "landing", 1)
-- 1788
put trackStopTime(sprite 10, 2)
-- 2072

-- or dot notation
put member("landing").trackStopTime(1)
-- 1788
put (sprite 10).trackStopTime(10, 2)
-- 2072
```

The `trackNextKeyTime()` and `trackPrevKeyTime()` functions return a value indicating the next and previous keyframes for a sprite. The `trackNextSampleTime()` and `trackPrevSampleTime()` functions return the next and previous samples of a track. The return values for all four of these functions are determined by the `digitalVideoTimeScale`—a system property used to measure digital video members. The value for this property is the number of units per second, the default being 60.

Masking a QuickTime sprite

A QuickTime member can have a 1-bit mask applied to it. This enables you to have the speed benefit of running the movie direct-to-Stage, yet have a nonrectangular shape. When creating a mask bitmap, it is important to note that a QuickTime member has its registration point in the upper-left corner. Just as setting `the video of member` to FALSE can leave an artifact on the screen, setting the mask while a sprite of the member is on the screen can have a similar effect. The mask should be set before the sprite appears on the Stage or, if after it is on the Stage, setting `the stageColor` to `the stageColor` refreshes the Stage. To set the mask, type the following:

```
set the mask of member "landing" to member "qtmask"
set the stageColor to the stageColor
```

```
-- or dot notation
member("landing").mask = member("qtmask")
the stageColor = the stageColor
```

To clear it, just set it to 0, as shown here:

```
set the mask of member "landing" to 0
-- or dot notation
member("landing").mask = 0
```

You can invert the mask by setting the `invertMask` property to TRUE, as in the following:

```
set the invertMask of member "landing" to TRUE
-- or dot notation
member("landing").invertMask = TRUE
```

QuickTime VR

You can control QuickTime VRs through Lingo, just as you can control normal QuickTime videos. Open the movie qtvr.dir, shown in Figure 19-10, to experiment with the commands as you work through the chapter. It is in the EXERCISE:CH19 (EXERCISE\CH19) folder on the accompanying CD-ROM.

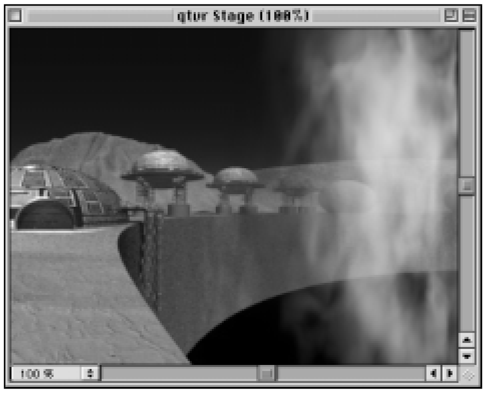


Figure 19-10: The qtvr.dir movie displaying a QuickTime VR sprite

The first thing to try is testing whether the member is a QTVR. A QuickTime member might or might not be a QTVR. You can use the `isVRMovie` property to test this, as follows:

```
put the type of member 1
-- #quickTimeMedia
```

```
put the isVRMovie of member 1
-- 1

-- or dot notation
put member(1).type
-- #quickTimeMedia
put member(1).isVRMovie
-- 1
```

As was shown with the QuickTime movie earlier, an interface can be created for a QTVR as well. Unlike a QuickTime movie, a QTVR expects you to interact with it by using the mouse. QTVRs already have nicely designed cursor feedback, but if you want to override that control, you can set the `mouseLevel` property of the sprite. The `mouseLevel` has four possible values:

- `#shared`: The default. Passes all mouse clicks to QuickTime and Director (in that order).
- `#all`: All mouse clicks are passed to QuickTime.
- `#none`: None of the mouse clicks are passed to QuickTime.
- `#controller`: Only mouse clicks on the sprite's controller are not passed to Director.

If you want to take control of the interaction with Lingo, you probably would want to set the `mouseLevel` of the sprite to `#none`. Then you would need to write the code to enable interaction with the QTVR.

Viewing commands

The `pan`, `tilt`, and `fieldOfView` properties enable you to change your view of the VR.

- Setting `pan` moves your view left or right.
- Setting `tilt` moves your view up and down.
- Setting `fieldOfView` moves your view in and out.

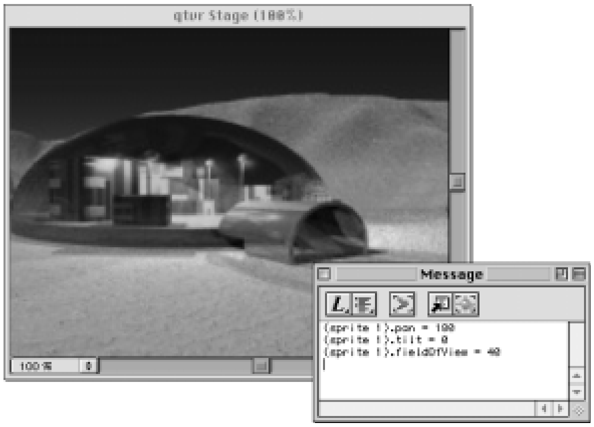
The value of each is in degrees. In the Message window, put each of these properties:

```
put the pan of sprite 1
-- 90.0000
put the tilt of sprite 1
-- 2.7000
put the fieldOfView of sprite 1
-- 64.2813

-- or dot notation
put (sprite 1).pan
```

```
  -- 90.0000
  put (sprite 1).tilt
  -- 2.7000
  put (sprite 1).fieldOfView
  -- 64.2813
```

These are the default values for this particular QTVR. Now try changing them as shown in Figure 19-11.



```
(sprite 1).pan = 180
(sprite 1).tilt = 0
(sprite 1).fieldOfView = 40
```

Figure 19-11: The view after the pan, tilt, and fieldOfView properties were changed.

Rewind the movie so that the sprite returns to its defaults. Play the movie, and in the Message window, type the following:

```
  swing(sprite 1, 180, 0, 40, 5)
  -- or dot notation
  (sprite 1).swing(180, 0, 40, 5)
```

Cool, huh? The view automatically "swings" to the position you specified. The first four arguments to the preceding `swing()` command you should recognize as the sprite, `pan`, `tilt`, and `fieldOfView` that you manually set moments ago in the Message window. The fifth argument is the speed it moves to the new position. Possible speed values are 1 to 10.

If you need to just move the image slightly, you can use the `nudge` command. You can nudge in eight directions, as in the following example:

```
  nudge(sprite 1, #up)
  -- or dot notation
  (sprite 1).nudge(#up)
```

Values for nudging are `#up`, `#down`, `#downLeft`, `#downRight`, `#left`, `#right`, `#upLeft`, and `#upRight`.

Image quality

You can change the quality of the sprite for the QTVR sprite while it is moving or while it is still. The `motionQuality` property can be set to `#minQuality`, `#maxQuality`, or `#normalQuality`. Lower qualities animate more quickly.

You can change the quality of the image while it is not moving by using the `static Quality` property. It has the same values as `motionQuality`: `#minQuality`, `#max Quality`, and `#normalQuality`.

Have you ever noticed how QTVRs antialias when you stop dragging them? It seems like that is a common default. You can force the QTVR to antialias as it is dragged by setting `motionQuality` to `#maxQuality`. Or you can have it not be antialiased when it is stopped by setting the `staticQuality` to `#minQuality`.

You can change the amount of warping by setting the `warpMode` property. Possible values are `#full`, `#partial`, and `#none`.

Nodal VRs

Several QTVRs can be put together into one QuickTime movie. These movies are linked together. Each one is referred to as a *node*. To get from one node to another, the author of the VR defines *hotspots.*

Open the movie nodal.dir (shown in Figure 19-12). You can find it in the EXERCISE: CH19 (EXERCISE\CH19) folder on the accompanying CD-ROM. This movie is four QTVRs bundled together. By clicking one of the three screens, you enter one of three QTVRs.

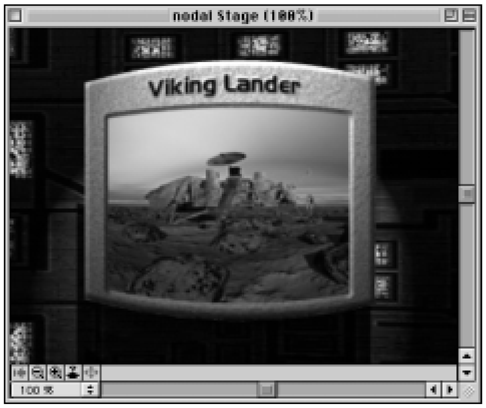


Figure 19-12: A nodal QuickTimeVR movie

The sprite in nodal.dir movie has four nodes numbered 127, 128, 129, and 130. You can get the value of the node you are in with the `node` property, but you cannot set it:

```
put the node of sprite 1
-- 127

-- or dot notation
put (sprite 1).node
-- 127
```

The numbers represent these nodes: 127 is the main node; 128 is Viking Lander; 129 is Terraforming; and 130 is Olympus Mons. You cannot get the names of nodes. We hope this will change in a future version of Director.

You can determine the type of node with the `nodeType` property. Possible values are `#panorama`, `#object`, and `#unknown`. `#unknown` means that the sprite isn't a QTVR. Use this property as follows:

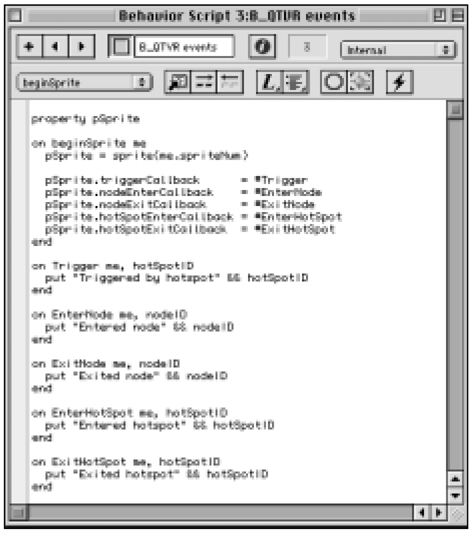
```
put the nodeType of sprite 1
-- #panorama

-- or dot notation
put (sprite 1).nodeType
-- # panorama
```

You can have a handler execute when you enter or exit a node, enter or exit a hotspot, or when you click a hotspot. Look at the behavior in Figure 19-13. The behavior sets the properties: the `triggerCallback`, the `nodeEnterCallback`, the `nodeExitCallback`, the `hotSpotEnterCallback`, and the `hotSpotExitCallback`. The values for these properties are symbols that correspond to the names of handlers in the behavior.

Through the use of callbacks, you can respond to the user's interactions with the QTVR, or you can just track their movements through it. Here is the output for a short session we just did:

```
-- "Entered hotspot 218"
-- "Triggered by hotspot 218"
-- "Exited node 127"
-- "Entered node 130"
-- "Exited hotspot 218"
-- "Entered hotspot 183"
-- "Triggered by hotspot 183"
-- "Exited node 130"
```



```
property pSprite

on beginSprite me
  pSprite = sprite(me.spriteNum)

  pSprite.triggerCallback       = #Trigger
  pSprite.nodeEnterCallback     = #EnterNode
  pSprite.nodeExitCallback      = #ExitNode
  pSprite.hotSpotEnterCallback = #EnterHotSpot
  pSprite.hotSpotExitCallback  = #ExitHotSpot
end

on Trigger me, hotSpotID
  put "Triggered by hotspot" && hotSpotID
end

on EnterNode me, nodeID
  put "Entered node" && nodeID
end

on ExitNode me, nodeID
  put "Exited node" && nodeID
end

on EnterHotSpot me, hotSpotID
  put "Entered hotspot" && hotSpotID
end

on ExitHotSpot me, hotSpotID
  put "Exited hotspot" && hotSpotID
end
```

Figure 19-13: A behavior to demonstrate QTVR callbacks

With this information, you know exactly which nodes we visited. If there were multiple paths to the nodes, you would know which path we used, based on the hotspots we triggered.

More hotSpot commands

If you've created a game using QTVRs, you might want to selectively enable and disable hotspots. You can do this with the `enableHotSpot` command, as shown here:

```
  enableHotSpot(sprite 1, 59, FALSE)
  -- or dot notation
  (sprite 1).enableHotSpot(59, FALSE)
```

HotSpot 59 is the one that leads to the Viking Lander. When you roll over that monitor, your cursor no longer changes; you are not able to enter that node.

You can also get the rect of a hotspot. The rect is the coordinates of where that hotspot is on the Stage at the moment in time when you check it. Rewind the movie and type the following:

```
  put getHotSpotRect(sprite 1, 59)
  -- rect(86, 77, 299, 234)

  -- or dot notation
```

```
-- (sprite 1).getHotSpotRect(59)
-- rect(86, 77, 299, 234)
```

Move down and to the right and get the hotspot's rect again, as follows:

```
put getHotSpotRect(sprite 1, 59)
-- rect(0, 28, 154, 184)

-- or dot notation
put (sprite 1).getHotSpotRect(59)
-- rect(0, 28, 154, 184)
```

If you had a sprite interacting with the hotspot, you would know exactly where to put it with this command.

A convenient function is `ptToHotSpotID`. With this function, you can find out whether a point is within a hotspot. Although you could easily write a function to do this, given the `getHotSpotRect`, it is nice that you don't have to. Write the following code:

```
put ptToHotSpotID(sprite 1, the mouseLoc)
-- 59

-- or dot notation
put (sprite 1).ptToHotSpotID(the mouseLoc)
-- 59
```

If it returns 0, then the point (in this case, the `mouseLoc`) is not within a hotspot.

Summary

In this chapter, you learned the following:

- You can import QuickTime movies into Director.
- Lingo can control the playback of a QuickTime movie.
- QTVRs are another type of QuickTime movie that can be manipulated through Lingo.
- You can assign functions that execute as the user interacts with a QTVR.

In the next chapter, we discuss troubleshooting your code.

✦ ✦ ✦