Getting started: Overview

Macromedia Director 8 Shockwave Studio is the world's foremost authoring tool for creating interactive multimedia. Developers rely on Director to create attention-grabbing business presentations, advertising kiosks, interactive entertainment and educational products. To see some of the exciting and varied ways in which developers use Director, visit Macromedia's Gallery at www.macromedia.com. You can see great examples of Shockwave at www.shockwave.com.

Your users can view your completed Director movie over the Internet, either in a Web browser or independent of a browser, or as a stand-alone projector suitable for LANs and distribution through CD-ROM and DVD-ROM.

For an animated introduction to Director, watch the **Guided Tour**.

What's new in Director 8

One of the most important changes in Director 8 is a Property Inspector that automatically switches context to match the current selection. The Property Inspector is referred to throughout this book. For basic information about it, see **The Property Inspector**.

Other new authoring features in Director 8 include the following:

Zoomable stage allows shrinking or expanding of the Stage window during authoring without affecting logical sprite sizes and positions. See **Increasing or decreasing your view of the Stage**.

Cast window List view provides a new way to display cast members, and provides the ability to sort and change member properties. See **Using Cast List view**.

Asset management fields on the Cast window, including a comments field and source control fields, are customizable for each cast member. **Using Cast List view**

Linked scripts let you store scripts in external text files that can be edited separately from a Director movie. See **Using linked scripts**.

Bitmap compression allows JPEG compression for bitmap members in a DCR. You can specify compression for individual bitmaps, or at the movie level for all bitmaps in your DCR. An optimize in fireworks option lets you preview the JPEG image at various quality settings. Bitmap compression offers a compression strategy for 32-bit cast members with alpha channel data. **Compressing bitmaps**.

Lockable sprites help prevent unintentional modifications during authoring. See **Locking and unlocking sprites**.

Guides on the Stage (in addition to the existing grid) help you place elements precisely. See **Positioning sprites using guides, the grid, or the Align window**.

Publish command lets you create a Shockwave movie, in your choice of HTML templates, by simply choosing File > Publish. A Publish Settings dialog box lets you configure how your want your Shockwave movie to appear in a browser. See **Creating Shockwave movies**.

Scalable Shockwave lets Shockwave movies stretch to fit the browser window while (optionally) preserving the original aspect ratio. See **Changing Publish settings**.

Multiple curve vectors offers the ability to create and edit vector cast members with more than one curve segment. **Drawing vector shapes**.

Inline IME, available for Japanese operating systems, supports direct entry of double-byte Japanese text in Shockwave and projectors.

Enhanced Lingo performance and new parent-child scripting functionality. See **Parent script and child object basics**.

Imaging Lingo lets you create and manipulate bitmap images entirely in Lingo. See **Controlling bitmap images with Lingo**.

Sound control Lingo allows precise, professional quality control of sound playback. See **Playing sounds with Lingo**.

New Lingo terms

Director 8 includes the following new Lingo terms.

allowZooming
activateApplication
bitmapSizes
bitOr()
breakLoop()
comments
createMask()
creationDate
deactivateApplication
duplicate() (image function)
elapsedTime
extractAlpha()
fadeOut()
fill()
getFlashProperty()
getPlaylist()
handler()
image
imageCompression
inlineImeEnabled
on isOKToAttach
loopCount
loopsRemaining
the markerList
modifiedBy
movieFileVersion
movieImageQuality
newCurve()
pan (sound property)
period
playNext()
rawNew()
regPointVertex
sampleCount
appMinimize
bitAnd()
bitNot()
bitXor()
characterSet
copyPixels()
createMatte()
curve
draw()
editShortCutsEnabled
endTime
fadeIn()
fadeTo()
flushInputEvents
getPixel()
getVariable()
handlers()
image()
imageQuality
isBusy()
linkAs()
loopEndTime
loopStartTime
member (sound property)
modifiedDate
movieImageCompression
name (timeout property)
originalFont
pause() (sound playback)
persistent
queue()
rect (image)
rewind()
seconds

setAlpha()
setFlashProperty()
setPixel()
setPlaylist()
setScriptList()
setVariable()
the soundMixMedia
status
stop() (sound)
target
time()
timeout()
timeoutHandler
timeoutList
trimWhitespace()
useFastQuads

Resources for learning Director

The Director package contains a variety of media to help you learn the program quickly and become proficient in creating multimedia—including online help, a multimedia guided tour, a tutorial, integrated tooltips, printed books, and a regularly updated Web site.

Director includes the following main instructional components.

Director Help and the Guided Tour

Director Help is the comprehensive information source for all Director features. The help includes complete conceptual overviews of all features, animated examples, descriptions of all interface elements, and a reference of all Lingo commands and elements. They are extensively cross-referenced and indexed to make finding information and jumping to related topics quick and easy.

The best place to start learning Director is the Guided Tour included with Director Help. The Guided Tour provides a quick conceptual overview of how to use key features to create and distribute a movie.

Click the Help button in any dialog box to open the relevant help topic.

Director Tutorial

When you're ready to actually start working in Director, proceed to the Director Tutorial. The tutorial shows you how to create a basic movie with some of Director's most useful and powerful features. The tutorial appears in Director Help and in Chapter 1 of *Using Director*.

Using Director book

Using Director is a printed excerpt of Director Help. It includes all the main topics in Director Help, but omits some topics that are less frequently used or becoming obsolete as Director evolves.

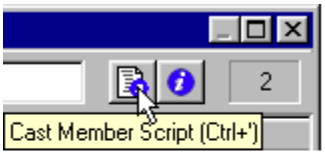
Lingo Dictionary

The *Lingo Dictionary* is a printed version of all the Lingo topics in Director Help.

Tooltips

When you place the pointer over a Director tool for a few seconds, a small tooltip appears that explains the function of the tool. When a keyboard shortcut is available, it is included in the tooltip.

In the following illustrations, Director is displaying tooltips for two different tools in the Cast window.



Keyboard shortcuts

Many commands that are available from Director menus are also accessible through the use of keyboard shortcuts. When you display a menu or submenu, the appropriate key combinations are shown next to the commands for which keyboard shortcuts are available.

Director Support Center

The **Director Support Center Web site** contains the latest information on Director, plus additional topics, examples, tips, and updates. Check the Web site often for the latest news on Director and how to get the most out of the program.

For example, you can visit the Director Support Center for additional information about these topics:

- **Using Director 8 behaviors**
- **Working with Multiuser behaviors**
- **Using the Shockwave Multiuser Server**
- **Using the XML parser**
- **Troubleshooting Lingo**
- **Authoring from Lingo**
- **Controlling vector shapes with Lingo**
- **Specifying chunk expressions with dot syntax**
- **Optimizing bitmaps in Fireworks**
- **Creating Java applets with Director**
- **Creating dialog boxes from the MUI Xtra**
- **Director 8 keyboard shortcuts**

Conventions used in Director Help and printed books

The help system and printed books use the following conventions:

- The terms *Lingo* and *Director* refer to version 8 of Director.
- Within the text and in Lingo examples, Lingo elements and parts of actual code are shown in `this` font. For example, `set answer = 2 + 2` is a sample Lingo statement.
- Quotation marks that are part of Lingo statements are shown in the text and Lingo code examples as straight quotation marks (") rather than as curly quotation marks (”).
- The continuation symbol (¬), which you enter by pressing Alt+Enter (Windows) or Option+Return (Macintosh), indicates that a long line of Lingo has been broken onto two or more lines. Lines of Lingo that are broken this way are not separate lines of code. When you see the continuation symbol in this book, type the lines as one line when you enter them in the Script window.
- Variables used to represent parameters in Lingo appear in *`italics`*. For example, *`whichCastMember`* is commonly used to indicate where you insert the name of a cast member in Lingo.
- Text that you should type is shown in **this** font.

Director 8 tutorial: Overview

You're about to see how easy it is to master basic tasks necessary to create a movie in Director 8. With a few more simple steps, you can add multiuser functionality to a movie and export the entire project for distribution. By completing this tutorial, you'll learn Director fundamentals and acquire a basis for exploring more advanced Director features.

For the tutorial, you'll create a movie that plays in the Web page of an organization called GardenChat. You'll also take advantage of Director's multiuser behaviors to add chat functionality to the site, allowing members of the organization to discuss gardening tips with each other in real time.

The tutorial assumes no prior knowledge of Director other than the information provided in the **Guided Tour**. You should, however, be familiar with basic computer operations such as using menus and selecting and dragging objects. The tutorial takes approximately two hours to complete, and it focuses on many Director processes, including the following:

- Creating a new movie, cast members, and sprites
- Using inks
- Creating animation using tweening, frame-by-frame animation, and blends
- Importing media
- Synchronizing sound
- Attaching behaviors
- Controlling streaming over the Web
- Publishing your movie for Web playback

View the completed Shockwave version of GardenChat

The tutorial takes you through the steps of creating an animated sequence that plays in a Web browser. When you publish a movie for Web playback, you create a Shockwave version of the movie with the .dcr extension. Your original Director movie remains unchanged.

1. Open your browser.
2. In your Director application folder, open the Learning folder and the Completed_Tutorials subfolder.
3. Drag the file Completed_Tutorial.html to your browser window.

   The completed GardenChat movie plays in your browser in the Shockwave movie format.

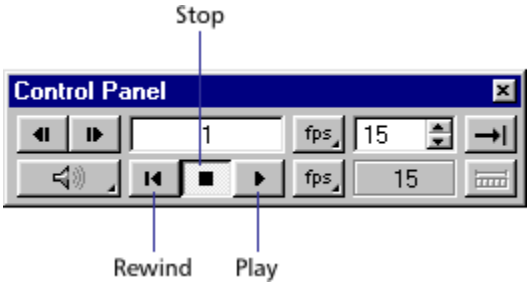
View the completed DIR version of GardenChat

When you work on a Director movie, you use the authoring environment. Director movies saved in this environment have the .dir file extension. (These movies are not yet prepared for distribution.) Now view the completed DIR version of the tutorial movie to understand how the assets work together on the Stage and in the Score to create the movie.

Note: The DIR version of the completed tutorial movie does not include the chat component.

1. Launch Director and then choose File > Open.
2. Browse to your Director application folder, open the Learning folder and the Completed_Tutorials folder, and then open Fun.dir.

3 To play the movie, click Play on the Control Panel or the toolbar along the top of the screen.



If the Control Panel and toolbar are not visible, you can select them from the Window menu.

4 After viewing the movie, click Stop and look at the Stage, Score, Cast window, and Property Inspector to get a sense of how the Director application and movie are organized.

If the Stage, Score, Cast window, and Property Inspector are not visible, you can select them from the Window menu.

Set up the movie

To begin your own version of GardenChat, you'll create a new movie and set the size of the Stage. You'll also select an appropriate color palette.

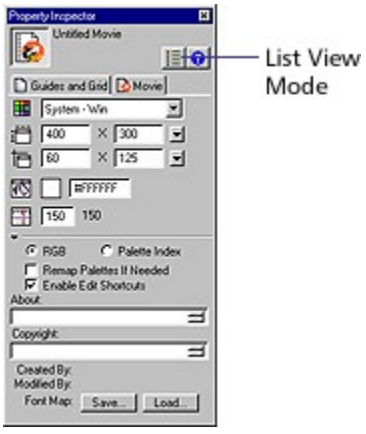
1 Choose File > New > Movie.

2 If you've made changes to the Fun.dir movie, Director prompts you to save them. Choose Don't Save.

Note that the default Stage is a different size than the Stage in the completed GardenChat movie.

3 To change your Stage size, click the Movie tab of the Property Inspector.

If the Property Inspector is not open, choose Window > Inspectors > Property. You should be in the default Graphical view, with the List View Mode icon deselected.



4 To specify a new Stage size in pixels, enter **450** in the first Stage Size field (width) and **500** in the second Stage Size field (height). After entering data in a field, click either the Stage or Property Inspector and the Stage resizes.

Because you are creating this movie for playback on the Web, you want to use a palette of Web-safe colors to ensure proper display. Director has a Web palette that you can select for your movie.

5 On the Movie tab of the Property Inspector, click the Movie Palette pop-up menu and select Web 216.

6 Choose File > Save. You can also press Control-S (Windows) or Command-S (Macintosh).

7 Name the movie **GardenChat1**.

8 Browse to the Learning folder within the Director application folder, and then open the My Tutorial folder; then save your movie.

You must save your file in My Tutorial; other tutorial files will point to your file in this location.

Note: As you complete the tutorial, remember to save your work frequently.

Create media in Director

You can create media in Director or import it from other programs. Simple media, such as text and backgrounds, are ideally suited for creation in Director.

Create a vector shape

Director lets you create multiple-curve vector shapes: mathematical descriptions of shapes filled with color or gradient colors. A vector shape uses much less memory than a comparable bitmap and downloads faster from the Internet.

You will create a vector shape filled with gradient colors to serve as your movie's background.

1 Choose Window > Vector Shape.

2 Click the Filled Rectangle tool and drag the cross hair from the upper left corner of the Vector Shape window to the lower right corner, creating a rectangle close to the size of your Stage.

Exact size is not important; you can resize the image later.

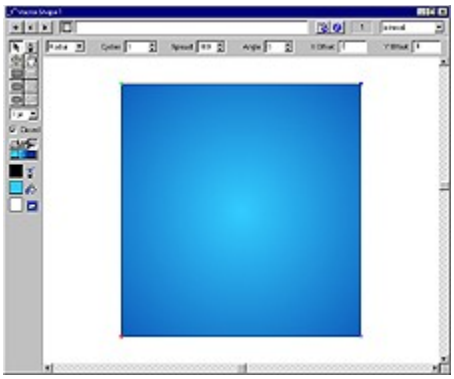
3 Click the rightmost Gradient Colors box and select a dark to medium shade of blue from the Color menu. 

4 Click the leftmost color box and select a light sky blue.

5 Click the Gradient button to create a smooth transition from light blue to dark blue. 

6 From the Gradient Type pop-up menu at the top of the window, select Radial.

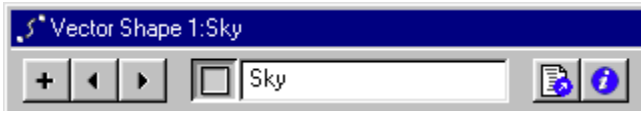
Radial creates a circular, rather than linear, gradient effect.



7 In the Y Offset box, type **-70**. (If you do not see the Y Offset box, make the window larger.)

This offsets the center of the gradient by moving it up 70 pixels, creating a sunlit sky effect. You would use a positive number in the Y Offset box to move the gradient down.

8 In the Cast Member Name field at the top of the window, type **Sky** as the name.



Naming cast members makes it easier to identify them in the Score. If you don't enter a name, Director assigns a number to the cast member based on its position in the Cast window.

9 Close the Vector Shapes window.

Create a text cast member

The Text window offers standard text formatting controls in a window that resembles a word processing program.

1 Choose Window > Text.

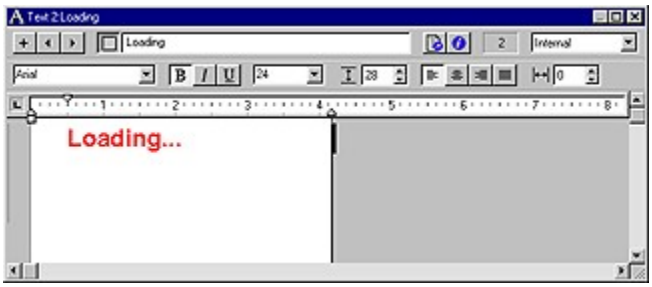
2 If necessary, resize the window to see all of the controls along the top.

3 Use the various fields to set font attributes. To match the font attributes of the Completed_Tutorial movie, use Arial, 24-point bold.

4 Choose Modify > Font and click the Color box to select a shade of red.

5 In the Text window, type **Loading...**

6 Name the text cast member **Loading**.

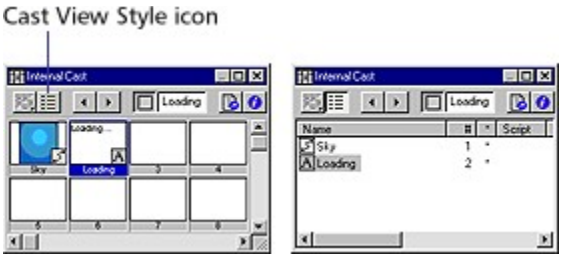


7 Close the Text window.

View cast members in the Cast window

Notice how the cast members you've created appear in the Internal Cast window with the names you've entered.

Use the Cast View Style icon to toggle between Cast List view and Cast Thumbnail view. Note that each view offers different features that assist you in managing your cast members.



This movie only requires a single Cast window; it does not use many cast members or media types. For future projects, keep in mind that you can create as many Cast windows as necessary to organize your work.

Import cast members

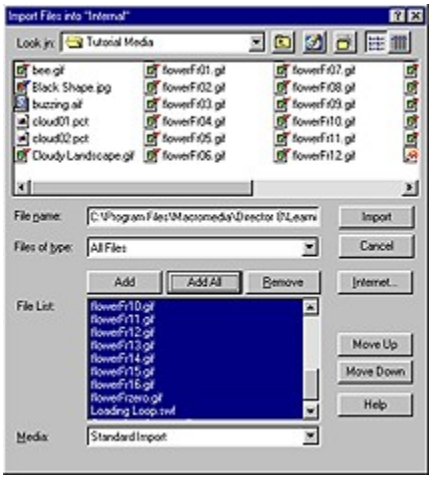
The cast members you've worked with so far are typical of media that you create within Director. To use more complex media, you usually import from other applications.

Director can import many popular types of media, including bitmaps, text, digital video, Flash movies, and sounds. For this movie, you'll import bitmap cast members created in an image editing program, an audio file, and a Flash movie.

1 Choose File > Import.

2 Browse to the Learning folder within the Director application folder, and then open the Tutorial_Media folder.

3 Click Add All.



Files in the current folder appear on the list of files to import.

4 Verify that Standard Import appears in the bottom pop-up menu, and click Import.

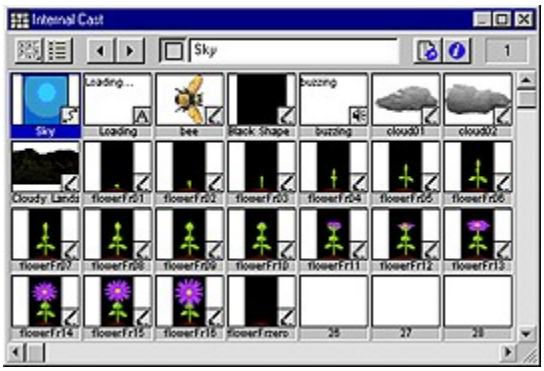
Director begins importing the files. Depending on the type of computer you have and how many colors your system is set to display, Director may prompt you to confirm the type of media you are importing or to change the color depth (number of colors) in the current image.

5 In the Select Format dialog box, select Bitmap Image and Same Format for Remaining Files, and then click OK.

6 In the Image Options dialog box, select Same Settings for Remaining Images. Accept the other default settings, and then click OK.

7 If a Format dialog box prompts you for information about importing the buzzing.aif sound file, specify Sound rather than QuickTime as the format.

The files that you import appear in alphabetical order by file name as cast members in the Internal Cast window. The Cast window assigns the member a number, based on its position in Cast Thumbnail view.



Note: If you change the cast member's position in the Cast window, the number assigned to the cast member also changes. In contrast, Cast List view offers a variety of list sorting options that do not affect the number assigned to the cast member.

Rename cast members

Although the cast member names are set to the file names of the imported files, you can change the names of cast members.

Notice that while most of the flower graphics in the tutorial follow the naming convention of flowerFr01, flowerFr02, flowerFr03, and so on, one flower is named flowerFrZero. You will rename flowerFrZero to make its name consistent with that of the other flowers.

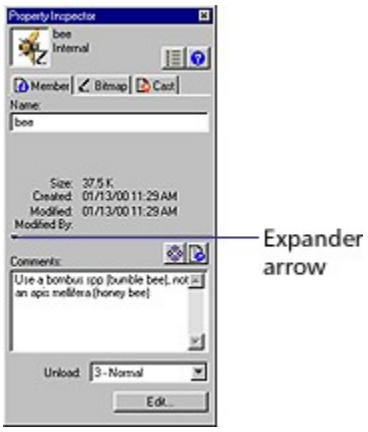
1. In the Cast window, select flowerFrZero.
2. In the Cast Member Name field at the top of the Cast window, select the text and change the cast member name to **flowerFr00**.

Add cast member comments

Often you'll have comments that you'd like to include with a cast member. Director lets you add cast member comments on the Member tab of the Property Inspector. You can then view the comments in the Cast window (in List view).

For the tutorial, you'll make a note to yourself about the bee.

1. If the Cast window is not in List view, click the Cast View Style icon.
2. Click the bee cast member to select it.
3. On the Member tab of the Property Inspector, click the Comments field. (If you do not see the Comments field, click the expander arrow.) Type the following text: **Use a bombus spp (bumble bee), not an apis mellifera (honey bee).**



4 Select the Cast window to see your comment in the Comments field.

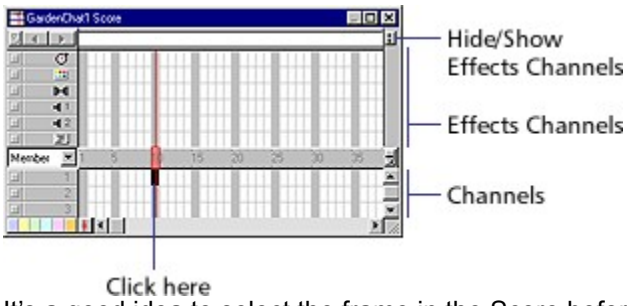
To see the comment, you might have to scroll to the right or enlarge your Cast window.

Create sprites from cast members

You're now ready to start creating sprites—objects that control when, where, and how your cast members appear in your movie. For example, when you move a cast member to the Stage, you're creating a sprite to indicate where the cast member appears in your movie. When you move a sprite to the Score, you're creating a sprite to indicate when the cast member appears.

1 Make sure the Cast window, Score, Stage, and Property Inspector are visible. If they're not, choose them from the Window menu.

2 In the Score, click frame 10 of channel 1 to select it.



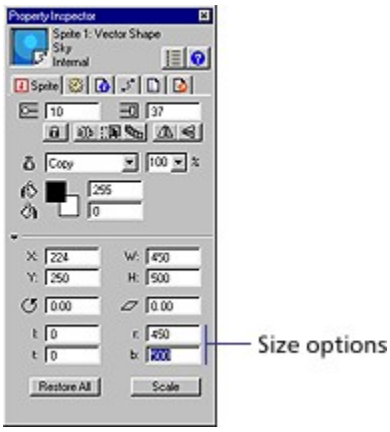
It's a good idea to select the frame in the Score before creating a sprite to ensure that the cast member ends up in the desired frame.

3 In the Cast window, drag the Sky cast member to the center of the Stage.

You've created a sprite. Notice that the sprite starts on frame 10 in the Score, which is the frame you selected in the previous step.

Now you need to resize the Sky sprite to fit on the Stage. The most accurate method is to use the Property Inspector.

4 Click the Sky sprite to select it. On the Sprite tab of the Property Inspector, set the Left, Top, Right, and Bottom options to **0**, **0**, **450**, and **500**, respectively.



Most changes that you make to a sprite do not affect the cast member assigned to the sprite. When you resize a sprite, therefore, the cast member used to create the sprite does not resize.

Note: Sprites, by default, span 28 frames. You can change this default setting in the Sprite Preferences dialog box (Choose File > Preferences > Sprite.)

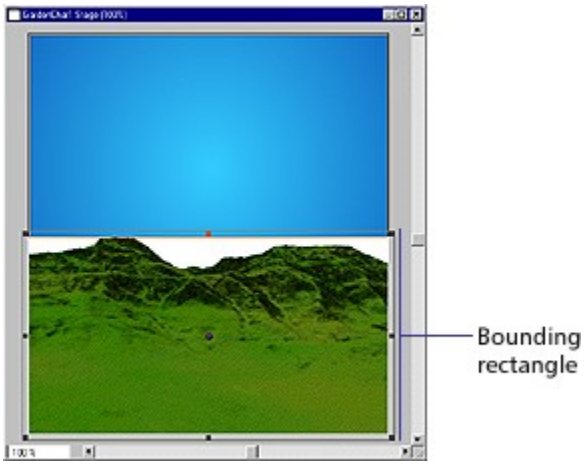
Change a sprite's ink

You can control the way a sprite's colors appear in Director by applying inks.

1. Drag the Sunny Landscape cast member to frame 10 of channel 2 in the Score.

   The new sprite appears inside a white box—the sprite's bounding rectangle—in the center of the Stage.

2. Drag the Sunny Landscape sprite to the bottom of the Stage.



You can make the bounding rectangle transparent by applying Background Transparent ink, which takes the pixels of a specified color (the default is white) and makes them transparent.

3. Make sure the Sunny Landscape sprite is selected. In the Sprite tab of the Property Inspector, select Background Transparent from the Ink pop-up menu.

The landscape's bounding rectangle becomes transparent.

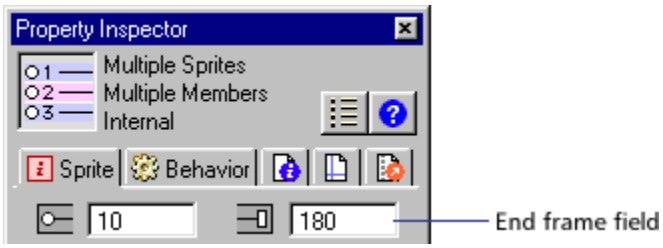
Change the duration of sprites

The Sky and Sunny Landscape sprites should be on the Stage while most of the movie plays, until frame 180.

1 Hold down Shift and click both sprites in the Score.

When you select multiple sprites, you can change settings for all selected sprites in the Property Inspector.

2 To extend the sprites to the 180th frame, enter 180 in the End Frame field on the Sprite tab of the Property Inspector. When you click anywhere in the window, the sprite spans extend to the 180th frame.



Lock sprites

You can lock a sprite to avoid inadvertent changes to it, either by you or by others working on the same project. Since you will be aligning one landscape over another, lock the Sunny Landscape in place.

After you lock a sprite, you cannot move it or change its settings until you unlock it.

1 Select the Sunny Landscape sprite either on the Stage or in the Score.

2 On the Sprite tab of the Property Inspector, click the Lock button. 

Note: To unlock a sprite that is locked, you can select it in the Score and then click the Lock button.

Create additional sprites

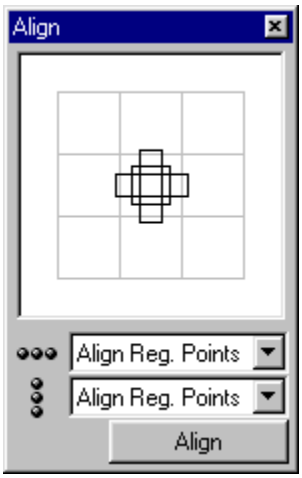
Since the tutorial movie begins with a cloudy day, you'll create additional sprites on top of the sunny landscape background to produce the overcast effect.

1. Drag the Cloudy Landscape cast member to frame 10 of channel 3 in the Score. If necessary, click the sprite to select it and make the Score window active.

2. In the Sprite tab of the Property Inspector, select Background Transparent from the Ink pop-up menu.

3. To align the two landscapes accurately, select both landscape sprites in the Score and choose Modify > Align.

4. In the Vertical Alignment and Horizontal Alignment pop-up menus, select Align Reg. Point and then click Align.



5. Click OK when Director warns that the change will only apply to the unlocked sprite.

   The two sprites align by their registration points. By default, the registration point of a bitmap cast member is its center.

6. On the Sprite tab of the Property Inspector, enter **130** in the End Frame field.

   Again, click OK when Director warns that the change will affect only the unlocked sprite.

   Remember to save your work frequently.

Zoom the Stage

Before you create an animated sequence that moves clouds across the sky, you will reduce the size of the Stage to make it easier to arrange the clouds.

In Director's authoring environment, you can use zooming to make the Stage either larger or smaller than your original movie. Zooming only affects your view of the Stage; it does not affect the Stage Size settings specified in the Property Inspector.

Director offers several different ways to zoom the Stage out, including the following method:

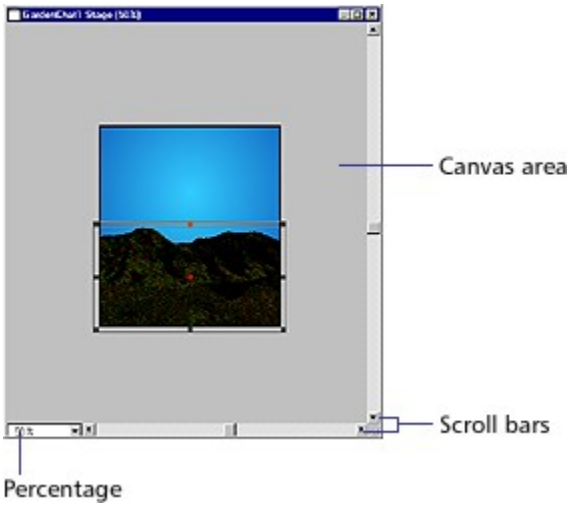
1. Click the Stage to make sure it's active.

2. Press Control-minus (Windows) or Command-minus (Macintosh) once to decrease the Stage size to 50%.

   The percentage of the Stage size appears in the Stage title bar.

   Notice that as you decrease the size of the Stage, you're increasing the size of the canvas area—the offstage area where you can drag cast members either before or after they appear on the

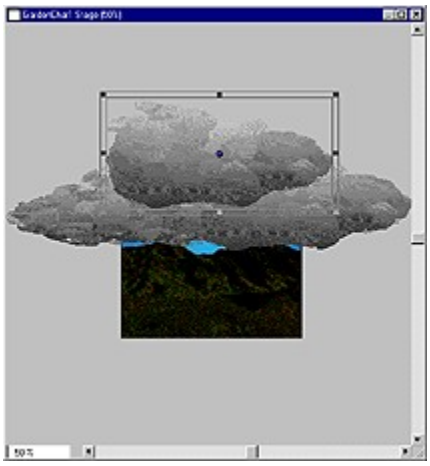
Stage.



Add the cloud sprites

1. In the Score, select frame 10 of channel 4.
2. Drag the Cloud02 cast member to the Stage, placing it just above the mountain closest to the right edge of the Stage. It does not matter if the Cloud extends off the Stage into the canvas area.

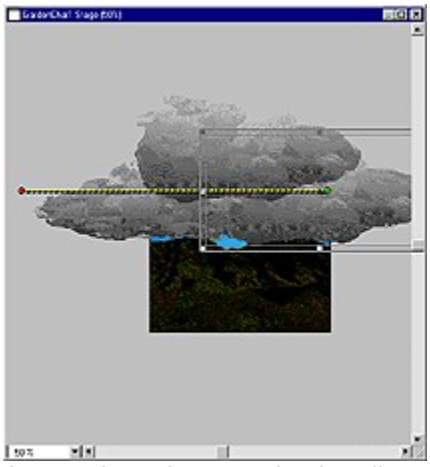
3. On the Sprite tab of the Property Inspector, type **120** in the End Frame field to extend the sprite's duration.
4. Set the sprite's ink to Background Transparent.
5. In the Score, create a sprite of the Cloud01 cast member in frame 10 of channel 5. Select the sprite and set its end frame to 95 and its ink to Background Transparent.
6. On the Stage, position the Cloud01 sprite to the left of the Cloud02 sprite.
7. Create another sprite of Cloud02 in frame 10 of channel 6. Select the sprite and set its end frame to 75 and its ink to Background Transparent.
8. Position the sprite on the Stage to cover as much of the visible blue sky as possible.



Create simple tweening animation

To make the clouds move across the sky, you'll use a simple animation technique called tweening. To tween, you define settings for the starting and ending frames, and Director fills in the frames in between.

1. Select the Cloud02 sprite in channel 4 of the Score.
2. On the Stage, locate the blue and red circle in the middle of Cloud02. This is a handle for tweening the path of a sprite.
3. Hold the Shift key and drag the handle to the left, all the way off the Stage and into the canvas area. Scroll to the left, if necessary.



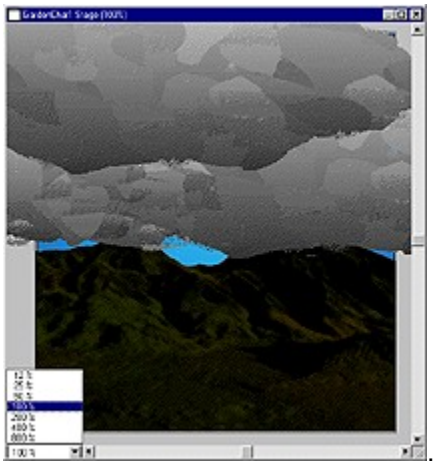
As you drag, the tweening handle separates into different circles. A green circle indicates the starting location of the sprite, a blue circle shows the sprite in relation to the current frame, and a red circle represents the ending location. Holding the Shift key constrains the movement to a straight vertical or horizontal line.

4. Select the Cloud01 sprite and use the tweening handles to drag it all the way off the Stage, to the left, and into the canvas area.
5. Select the Cloud02 sprite in channel 6 and tween it off the Stage, to the left.
6. Return the Stage to 100% using one of these methods:

- With the Stage active, Press Control-plus (Windows) or Command-plus (Macintosh) once to

increase the Stage size to 100%.

- Choose View > Zoom > 100%.
- Choose View > Zoom Stage In until the title bar indicates the Stage size is 100%.
- Click the Zoom menu and select 100%

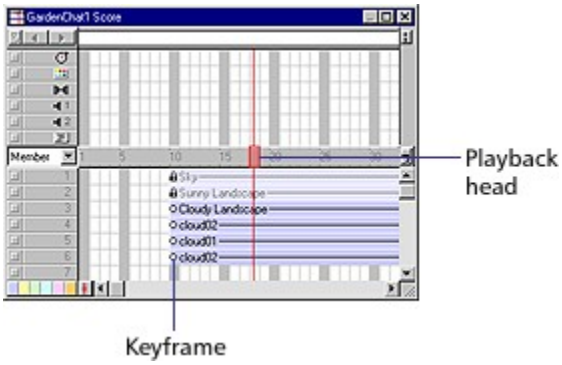


7 Organize your desktop to see both the Stage and the Score.

8 Click Rewind and Play on the toolbar along the top of the screen.

The clouds move between the starting and ending points you've defined.

Notice in the Score that the playback head (the red vertical bar) moves across each frame in the Score as the movie plays. The playback head indicates the current frame. You can drag the playback head across the Score to view the desired frame.



9 Click Stop.

Note: In the Score, small circles now appear at the beginning and end of the three cloud sprites. These circles represent keyframes, and they indicate where the property of a sprite changes.

Stop the playback head from looping

When you play your movie, the playback head goes to the last frame in the movie and loops back to the first frame. In the tutorial movie, you want the playback head to stop on the last frame. Later in this tutorial, you will add a behavior to make the last frame play continuously.

To stop the playback head from looping, choose Control > Loop Playback to deselect it.

Now when you play your movie, the playback head stops on the last frame.

Blend sprites

In addition to tweening the path of a sprite, you can tween other sprite properties such as size, rotation, and blend. You tween blend settings to make a sprite fade in or out. In your tutorial movie, for example, after the clouds move off the Stage, you want the landscape to appear sunnier. You'll accomplish this with blend settings that fade the dark landscape out as the sunny one fades in.

You first want to indicate the frame in which the blend effect should start to take place.

1. In the Score, click frame 80 of the Cloudy Landscape sprite.
2. Choose Modify > Split Sprite.

   The sprite splits into two at the selected frame.
3. Select the end keyframe (the small rectangle in frame 130) of the second Cloudy Landscape sprite. On the Sprite tab of the Property Inspector, select Blend from the Ink pop-up menu and 10% from the Blend pop-up menu.

   Notice that the end keyframe changes to a small circle, indicating a change in the sprite's property.
4. Rewind and play the movie to see the blend effect.

Create frame-by-frame animation

A sprite is often an instance of a single cast member on the Stage or in the Score. However, one sprite can also include several cast members. In addition to using multiple sprites to create animation, you can use multiple cast members within a single sprite. Animation that uses multiple cast members is called frame-by-frame animation. This technique offers a way to create animation that is more complex than simple tweened animation.

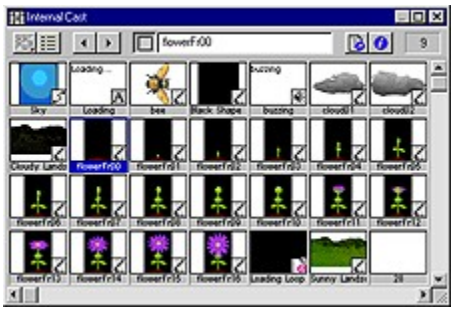
Change the order of cast members

You can create frame-by-frame animation several different ways in Director. For this tutorial, you'll use the Cast to Time method, which lets you move a series of cast members to the Score as a single sprite.

To prepare for the Cast to Time method of animation, your cast members must appear in the Cast window in the same order that they'll appear in the animation. In the Cast Thumbnail view, notice that the first flower graphic, flowerFr00, is out of sequence.

1. If you are not already in the Cast Thumbnail window, click the Cast View Style icon.
2. Drag the thumbnail of flowerFr00 on top of flowerFr01.

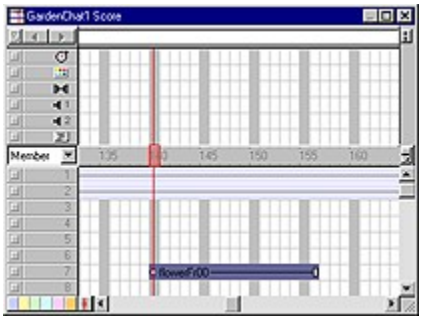
   The flowerFr00 thumbnail takes the position flowerFr01 occupied, and flowerFr00 becomes the first flower in the group of flowers. The numbers the Cast window assigns to the cast members reflects the new order.



Create Cast to Time animation

1. Select frame 140 of channel 7 in the Score.
2. With the Cast window active, verify that the flower bitmaps are in order according to their title (flowerFr00, flowerFr01, flowerFr02...).
3. Select all of the flowers by Shift-clicking the first flower in the series, flowerFr00, and Shift-clicking the last flower (flowerFr16).
4. Choose Modify > Cast to Time, or press Alt (Windows) or Option (Macintosh) while dragging the cast members to the Stage.

   The flowers appear as a single sprite in the selected Score frame.



By default, Background Transparent ink is set to turn white bounding rectangles transparent. With the flower sprite, the background is black. You can set Director to make the black background transparent.

5. With the flower sprite selected, go to the Sprite tab of the Property Inspector. Click the Background Color box and select black.

6. Set the sprite's ink to Background Transparent.
7. In the Score, move the playback head to frame 165.
8. On the Stage, select and drag the flower sprite to the bottom center edge of the window.

8 Extend the flower sprite in the Score to frame 180.

9 Rewind and play the movie.

Attach behaviors to sprites

Director drag-and-drop behaviors offer functionality beyond what you can accomplish by simply dragging sprites to the Stage and Score. Behaviors also add intelligence and flexibility to a movie.

Instead of playing a series of frames exactly as the Score dictates, a behavior can control the movie in response to specific conditions and events.

In this tutorial, you will use behaviors to make a bee move randomly around a flower and also follow the movements of the user's mouse pointer.

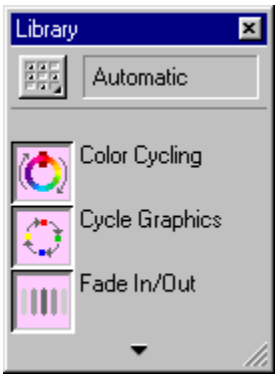
When attaching behaviors to an entire sprite, rather than a frame, you can drag the behavior to the sprite in the Score or on the Stage.

1 Click frame 175 of channel 8 in the Score.

2 Drag the bee cast member to the Stage, close to the flower, and shorten the sprite to frame 180.

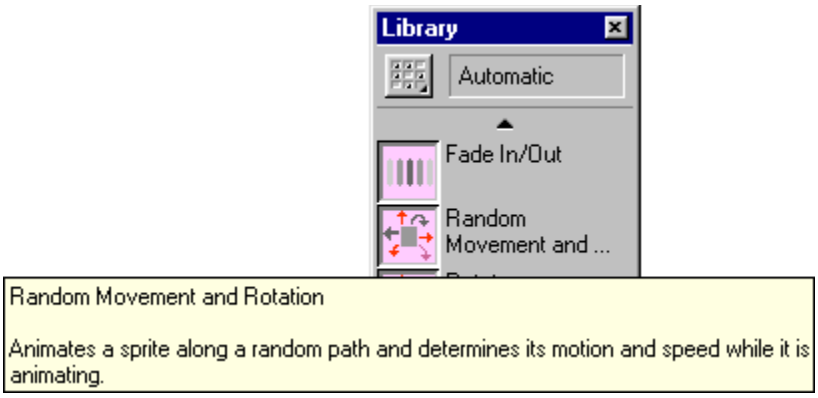
In addition to using the Property Inspector, you can drag a sprite's end frame to shorten or extend the sprite.

3 If the Library palette is not visible, choose Window > Library palette.



The Library palette displays categories containing all behaviors included with Director. The name of the active category appears in the Library List field at the top of the palette.
The name of each behavior appears next to an icon indicating its type.

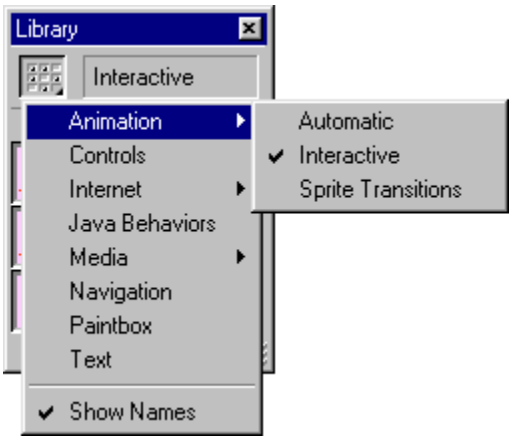
4 Verify that you're in the Automatic category in the Library List field. Scroll down to the `Random Movement and Rotation` behavior. Hold the mouse pointer over the behavior icon to read the description of the behavior.



5 Drag the `Random Movement and Rotation` icon to the bee sprite in the Score. A Parameters dialog box appears, letting you specify how you want the behavior to perform. Accept the defaults by clicking OK.

Now you'll add another behavior to the bee, but you'll drag the behavior to the sprite on the Stage. You'll add the `Turn Towards Mouse` behavior, which will make the bee turn in response to the movement of your user's mouse pointer.

6 Select Animation > Interactive from the Library List pop-up menu.



7 Scroll down to the `Turn Towards Mouse` behavior. Hold the mouse pointer over the behavior and read the description that appears. Drag the behavior to the bee on the Stage.

Notice that a shaded rectangle appears around the selected sprite as you drag the behavior on top of it.

8 In the Parameters dialog box, from the middle pop-up menu, select Always and click OK.

You attach behaviors to frames for actions that affect how the movie, rather than a particular sprite, behaves. Now you'll attach a behavior that makes the last frame of the movie play repeatedly.

9 From the Library List pop-up menu, choose Navigation.

10 Scroll down to the `Hold on Current Frame` behavior and drag it to the script channel and the last frame of your movie. `Hold on Current Frame` is one of the few behaviors that does not

require you to set additional parameters.

11 Rewind and play the movie.

Notice that when the movie reaches the last frame, it continues to play.

Remember to save your work frequently.

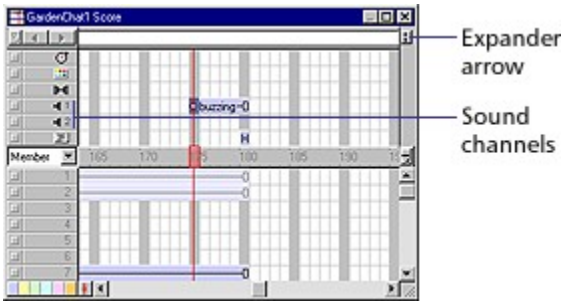
Add sound

Since a bee without its buzz seems less than adequate, you'll now add sound to the movie. Director offers many ways to add and synchronize sound, including use of cue points and Lingo. In this tutorial, you'll use a very simple procedure to control sound in the Score.

Controlling sounds in the Score is similar to controlling other sprites. Rather than using Score channels, however, you place the buzzing sound in a sound channel at the top of the Score. Next, you extend the sound so it's synchronized with the appearance of the bee.

1 If the sound channels are not visible along the top of the Score, click the expander arrow on the right side of the Score.

2 Drag the buzzing cast member to frame 175 of sound channel 1, and shorten the end keyframe to frame 180.



3 On the Sound tab of the Property Inspector, select Loop. This sets the sound so that it plays continuously while the bee is on the Stage.

4 Rewind and play the movie.

Control streaming

Streaming from the Internet causes a movie to play as soon as the content required for the first frame is downloaded to your user. The remaining content downloads in the background. Streaming dramatically shortens the perceived download time of a movie.

Create a looping introduction for a streaming movie

To create a movie that streams well, it's often a good idea to start with an introductory scene that downloads quickly and loops until the next scene has downloaded. In this tutorial, you'll add a loading Flash movie that lets your user know the rest of the movie is downloading.

1 Drag the Black Shape cast member to frame 1 of channel 1 in the Score. Rather than extending 28 frames, the sprite fits in the 10 available frames.

2 On the Sprite tab of the Property Inspector, resize the sprite by setting the Left, Top, Right, and Bottom options to **0**, **0**, **450**, and **500**, respectively.

3 In the Score, select frame 1 of channel 2, and drag the Loading text cast member to an area just above the middle of the Stage.

4 Select the Loading text sprite and set its ink to Background Transparent.

5 Drag the Loadloop cast member to frame 1 of channel 3 in the Score. This cast member, a Flash movie, will add interest to the movie while your user waits for more of the frames to download.

6 On the Stage, drag the Loadloop Flash movie so that it's centered underneath the text.

7 From the Library List pop-up menu, choose Internet > Streaming.

8 Drag the `Loop Until Media in Frame is Available` behavior to frame 1 of the script channel.

9 In the Parameters dialog box, type **10** in the Wait for Media in Frame field, then click OK.

You are telling Director to play the Flash movie until all of the media in frame 10 downloads. Once the sprites in frame 10 download, the looping behavior ends and the movie proceeds to play.

Note: You can view looping behaviors when you play the movie from a server.

Publish your movie for the Web in one step

Amazingly, Director can create a Web-friendly Shockwave version of your movie in one step.

To publish your move, save it and choose File > Publish.

Using the default Publish settings, Director creates a Shockwave version of your movie in the same directory as your original movie. Your browser window opens and your Shockwave movie plays.

Change Publish settings

When you use the Publish command, you take advantage of Director's default Publish settings, or you can modify them with the Publish Settings dialog box.

For the tutorial movie, rather than use the default HTML template specified in the Publish Settings dialog box, you'll use a special HTML page designed for GardenChat.

1 In Director, choose File > Publish Settings.

The Publish Settings dialog box appears.

2. On the Formats tab, choose No HTML Template from the HTML Template pop-up menu.

3. Save your movie and then choose File > Publish. Click OK at the prompt to overwrite your previous DCR file.

   When you save your movie, Director also saves any changes you've made to the Publish Settings dialog box. The next time you want to publish your movie in the same way, you can simply choose File > Publish without having to modify Publish settings.

Add multiuser chat functionality to GardenChat

To add another layer of functionality to your Web site, you're going to use Director's multiuser behaviors to create chat capabilities. A group of your GardenChat users will then be able to discuss soil conditions, the best fertilizer, and the weather simultaneously and in real time.

If you've never used Director's multiuser behaviors, you'll see how simple it is to create a chat in about 5 minutes.

To build a chat, you launch a local server application that supports the chat, and you create a Director movie designed to communicate with the server. Then you create a Shockwave version of the movie.

Launch the Shockwave Multiuser Server and determine the server IP address

The Multiuser Server is included in the Director 8 default installation.

1. In your Director 8 application folder, open the Shockwave Multiuser Server 2.1 subfolder, and double-click the MultiuserServer icon.

   The server launches.

2. To determine the server's IP address and see additional information about the server, choose Status > Server.

3. Write down the server IP address or copy it to the Clipboard. You will need this information later.

Open the chat movie

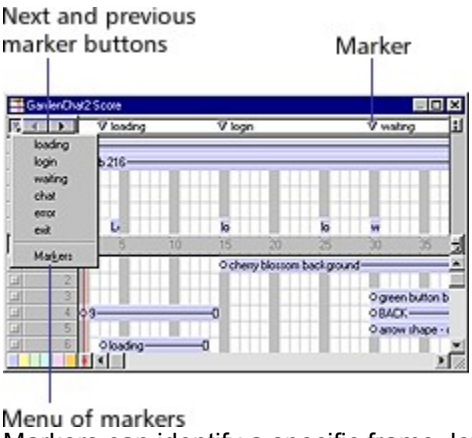
1. In Director, choose File > Open.

2. Browse to your Director 8 application folder. Open the Learning folder and the My Tutorial folder, and then open Chat.dir.

If you were to create this chat movie from scratch, you would place the text, fields, and artwork on the Stage. You could also use behaviors to add special effects. Most of this work is already completed for you. You will add the behaviors to your template to give the chat its multiuser functionality.

Use markers

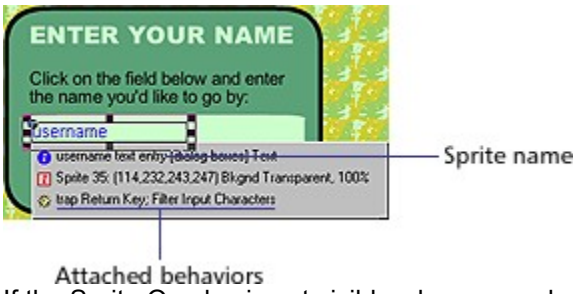
In the Score, notice that six markers identify key scene changes.



Markers can identify a specific frame, let you specify a frame to which to take your user, and so on. Now you will use markers to go to the beginning of a scene.
Click Next Marker twice to move the playback head to the login marker in frame 15.

Use the Sprite Overlay

The Sprite Overlay displays important information on the Stage about a selected sprite, including the sprite's name and the name of behaviors attached to the sprite. You can click the icons on the overlay to view different properties in the Property Inspector.



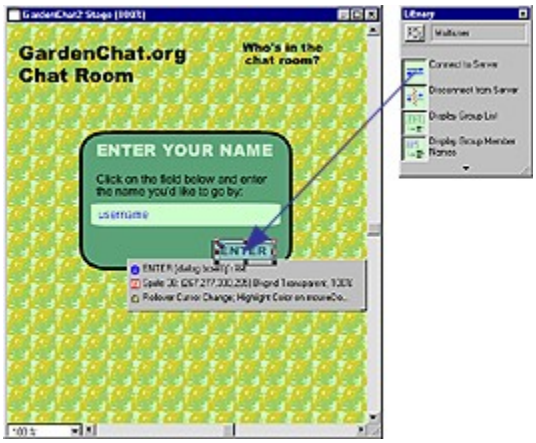
If the Sprite Overlay is not visible when you select a sprite, choose View > Sprite Overlay > Show Info.

Note: As you complete the following steps to attach behaviors, make sure you drag the behavior to the specified sprite and not to a background sprite, which would have a different name. Use the sprite name in the Sprite Overlay to assist you.

Add the Connect to Server behavior

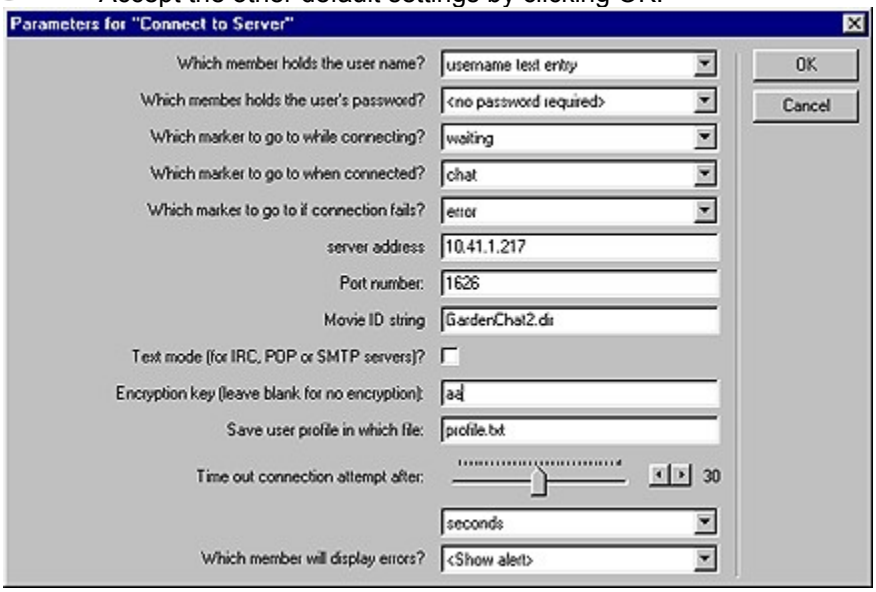
The `Connect to Server` behavior connects your user to the Multiuser Server. You attach the `Connect to Server` behavior to the sprite your user will click to establish a server connection. In this tutorial, you will attach the `Connect to Server` behavior to the Enter sprite.

1. If the Library window is not open, choose Window > Library Palette.
2. From the Library List pop-up menu, choose Internet > Multiuser.
3. Drag the `Connect to Server` behavior to the Enter sprite on the Stage.



4 Set the following parameters:

- In the Which Member Holds the User Name pop-up menu, choose Username Text Entry.
- Accept the default setting of the Which Member Holds the User's Password pop-up menu.
- In the Which Marker to Go to When Connecting pop-up menu, choose Waiting.
- In the Which Marker to Go to When Connected pop-up menu, choose Chat.
- In the Which Marker to Go to If Connection Fails pop-up menu, use the default setting.
- In the Server Address field, enter the server IP address that you recorded from the server application window.
- Verify that 1626 is in the Server Port Number field.
- Verify that the name of the movie is in the Movie ID String field.
- Accept the other default settings by clicking OK.

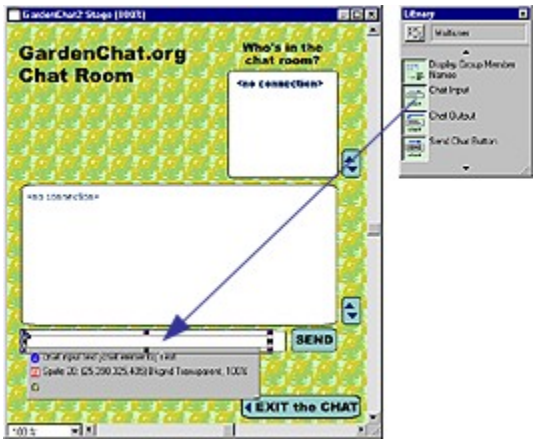


Add the Chat Input behavior

Now you can add the `Chat Input` behavior to the Chat Input text sprite. `Chat Input` is the behavior you attach to the text or field sprite in which your user enters information.

1 Click Next Marker twice to go to frame 45.

2 From the Library palette, drag the `Chat Input` behavior to the Chat Input text sprite on the Stage.



3 Accept the default parameters, and then click OK.

Add the Chat Output behavior

You attach the `Chat Output` behavior to the text or field sprite that will display the current chat text.

1 From the Library palette, drag the `Chat Output` behavior to the Chat Output sprite on the Stage.

2 Accept the default parameters by clicking OK.

Add the Send Chat Button behavior

You attach the `Send Chat Button` behavior to the sprite your user clicks to send the chat input text to the Multiuser Server, which then sends the text to chat participants. The `Send Chat Button` behavior includes a parameter that lets you select the sprite containing the information to send.

1 Drag the `Send Chat Button` behavior to the Send sprite on the Stage.

2 In the pop-up menu, select 20-Member ‘Chat Input Text,’ and then click OK.

Add the Disconnect from Server behavior

The `Disconnect from Server` behavior ends the server connection when your user has finished chatting.

To add the `Disconnect from Server` behavior, drag it to the Exit sprite on the Stage.

The `Disconnect from Server` behavior does not require additional parameters.

You’ve finished adding chat functionality to the movie.

Create a Shockwave chat movie

To allow your users to take advantage of the chat feature on the Web, you’ll create a Shockwave version of your movie.

1 Choose File > Publish Settings.

2 On the Formats tab, select No HTML Template from the HTML Template pop-up menu and click OK.

3 Save Chat and choose File > Publish.

Chat appears in your Web browser.

4 Type your name in the Enter Your Name field and click Enter.

Remember, the Enter sprite is where you attached the `Connect to Server` behavior.

5 In the Chat Room, type a gardening question in the Chat Input area and click Send.

6 The text appears in the Chat Output field where other users, connected to the same server, can view and respond to the question.

Talk to yourself

Although the server and browser are on the same computer for this project, the server and chat room participants can be anywhere on the Internet. If no one else is connected to the server running your chat movie, you can chat with yourself using two different browser windows.

1 In your browser window, select the URL and copy it.

If you are using Netscape Navigator, you can find the URL in the Location field; if you are using Microsoft Internet Explorer, the URL is in the Address field.

2 Open a new browser window and paste the URL in the same field of the new browser. Press Enter (Windows) or Return (Macintosh).

A new version of GardenChat appears.

3 Resize the two browser windows to view both of them, side by side.

4 In the second version of GardenChat, log in by typing a new user name in the Enter Your Name field and click Enter.

You can now answer the gardening question that you posted, or you can post a new question.

Continue to switch from one browser to another, and type messages back and forth to yourself. View the conversation in the Chat Output field.

5 When you finish chatting, click Exit the Chat, which is where you attached the `Disconnect from Server` behavior.

6 Close one of the two open browser windows.

Set up other chat participants

If you have access to a Web server, you can place the HTML file and Shockwave movie on the server, then run your chat from any Internet location. For the chat to work, the Shockwave Multiuser Server application must be running at the IP address you specified for the movie.

View your GardenChat Web site

You’re now ready to view both of your DCR files in the GardenChat Web site.

1 In your Director 8 application folder, open the Learning folder and the My_Tutorial folder, and drag the MyProject.html page to your browser window.

2 View the opening animation, and then click the Chat button to view your chat movie.

Continue learning about Director

By completing this tutorial, you've become familiar with Director features and procedures necessary to produce your own movie. You now know how to do the following:

- Create a new movie, cast members, and sprites
- Use inks
- Create animation using tweening, frame-by-frame animation, and blends
- Import media
- Synchronize sound
- Attach behaviors
- Control streaming over the Web
- Publish your movie for Web playback

Your final steps, outside the scope of this tutorial, would be to continue adding functionality to the HTML pages and to upload the HTML files to a server where they would be accessible to your users.

Continue learning about Director's many useful features by reading topics in *Using Director* and Director Help.

Director basics: Overview

Macromedia Director 8 Shockwave Studio is the tool of choice for legions of Web and multimedia developers. With Director you can create movies for Web sites, kiosks, and presentations, as well as movies for education and entertainment. Movies can be as small and simple as an animated logo or as complex as an online chat room or game. Director movies can include a variety of media, such as sound, text, graphics, animation, and digital video. A Director movie can link to external media or be one of a series of movies that refer to one another.

Your users view completed Director movies in one of two ways:

- In the Shockwave movie format, which plays in Shockwave-enabled Web browsers. Millions of Web users already have the Shockwave player on their computers, browsers, or system software. Others have downloaded Shockwave, which is free, from Macromedia's Web site.
- In a projector, which plays on your user's computer as a stand-alone application.

Creating a new movie

Director is organized around a movie metaphor.

To create a new movie:

Choose File > New > Movie.

Introducing the Director workspace

When creating and editing a movie, you typically work in five key windows: the Stage, the Score, the Cast window, the Property Inspector, and the Control Panel.

The Stage

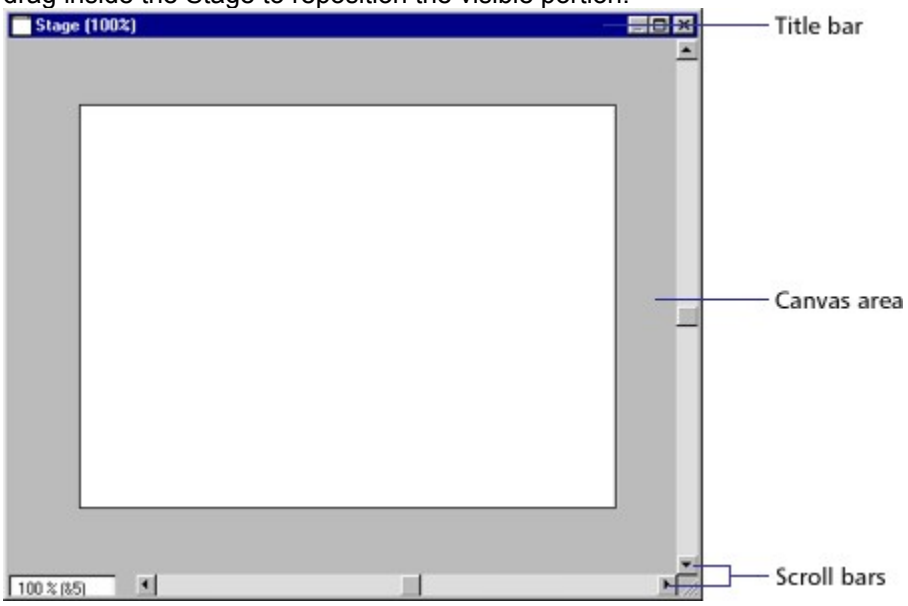
If the Stage is not open, choose Window > Stage.

The Stage is the visible portion of a movie on which you determine where your media elements appear.

During authoring you have the ability to define the properties of your Stage, such as its size and color. As you work on your movie, you can use zooming to make the Stage either larger or smaller than the original movie, while also scaling the coordinates for the Stage objects. To align objects on the Stage, you can choose to display guides and grids or use the Align window.

To scroll around the Stage, do one of the following:

- Use the scroll bars. (To show or hide Stage scroll bars, choose File > Preferences > General and select or deselect Show Stage Scrollbars.)
- Select the Hand tool from the Tool palette, then drag inside the Stage to reposition the visible portion.
- Bring the Stage to the front, hold down the Spacebar to temporarily switch to the Hand tool, then drag inside the Stage to reposition the visible portion.



The Score

If the Score is not visible, choose Window > Score.

The Score organizes and controls a movie's content over time in rows that contain the media, called channels. The Score includes special channels that control the movie's tempo, sound, and color palettes. The Score also includes frames and the playback head. You use the Score to assign scripts —Lingo instructions that specify what the movie does when certain events occur in the movie.

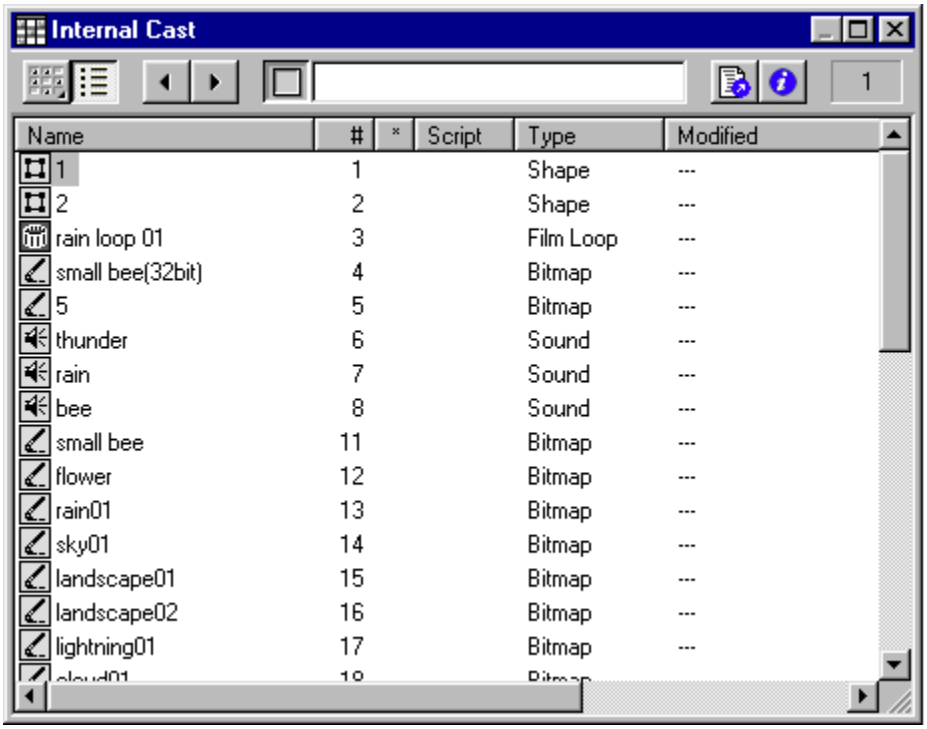
You can control the Score by zooming to reduce or magnify your view and by displaying multiple Score windows. You can also control the Score's appearance by using File > Preferences > Score.

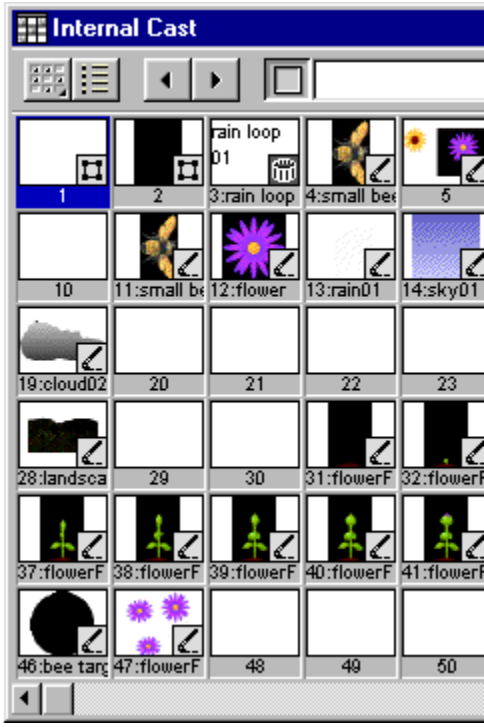
-

The Cast window

If the Cast window is not visible, choose Window > Cast.

In the Cast window you can view your cast members, which are the media in your movie, such as sounds, text, graphics, and other movies. Cast members can also include assets that you use in your Score but not on the Stage, such as scripts, palettes, fonts, and transitions. You can create cast members in Director, and you can import existing media to include in your cast. The Cast window lets you view your cast members in either of two ways, depending on your preference: as a list or as thumbnails.





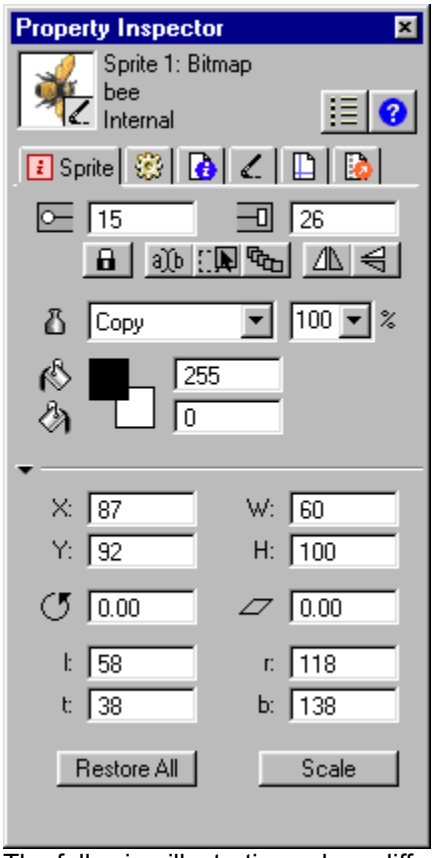
The Property Inspector

If the Property Inspector is not visible, choose Window > Inspectors > Property.

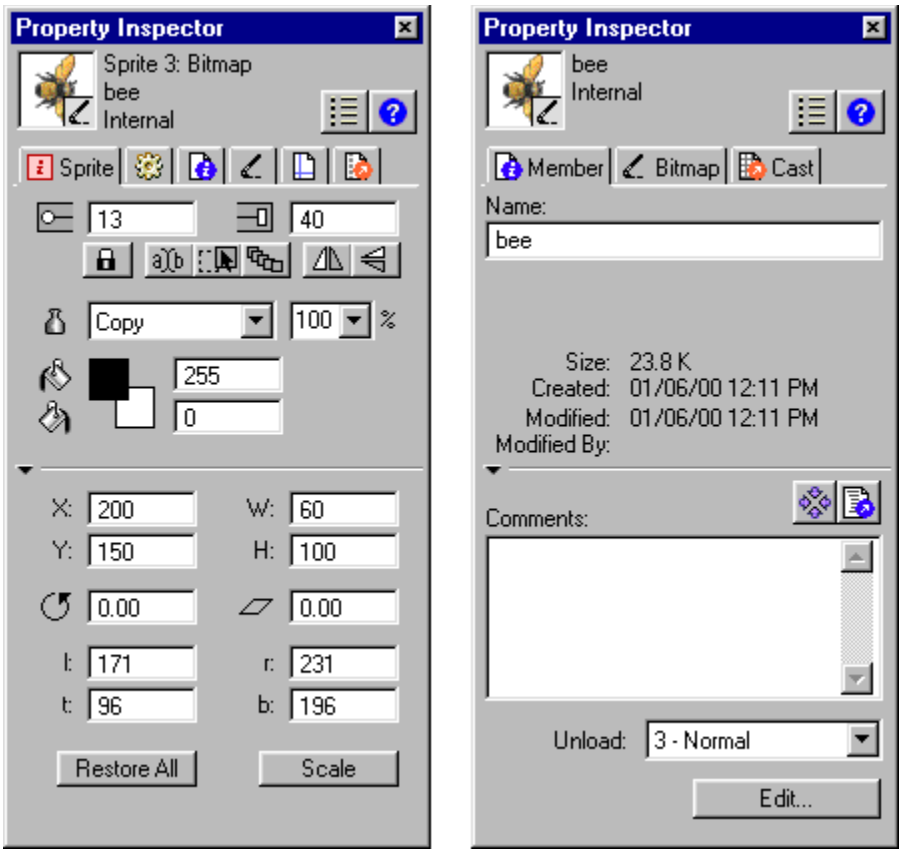
Instead of dialog boxes that let you view and change information related to different Director elements, Director now uses a single, tabbed Property Inspector. The tabs visible in the Property Inspector change to reflect the properties of the selected elements.

The Property Inspector provides a convenient way to view and change attributes of any selected object, or multiple objects, in your movie. Once you select an object, relevant category tabs and associated fields for it appear on the Property Inspector. If you select multiple objects, only the information common to all of the selected objects appears.

The List View Mode icon on the Property Inspector lets you toggle between a List and a Graphical view.



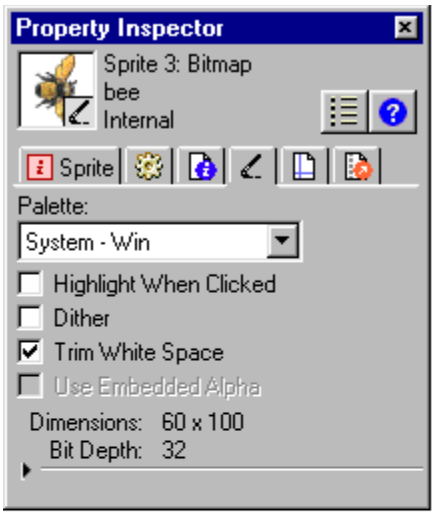
The following illustrations show different information appearing in the Property Inspector depending on what is selected. In the first illustration, a sprite is selected. In the second illustration, a cast member is selected.

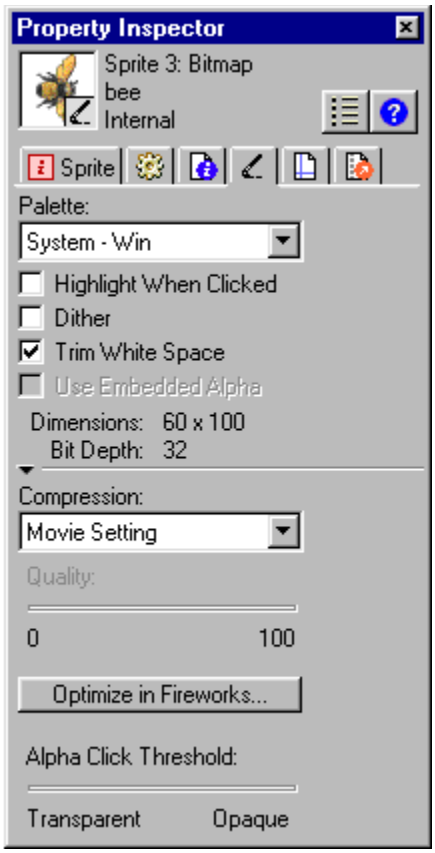


To show more or less information in the Property Inspector:

Click the expander arrow in the Property Inspector.

The following illustrations show different information appearing in the Property Inspector depending on whether the expanded information is hidden or shown.


Property Inspector
Sprite 3: Bitmap
bee
Internal
Sprite
Palette:
System - Win
Highlight When Clicked
Dither
Trim White Space
Use Embedded Alpha
Dimensions: 60 x 100
Bit Depth: 32


Property Inspector
Sprite 3: Bitmap
bee
Internal
Sprite
Palette:
System - Win
Highlight When Clicked
Dither
Trim White Space
Use Embedded Alpha
Dimensions: 60 x 100
Bit Depth: 32
Compression:
Movie Setting
Quality:
0
100
Optimize in Fireworks...
Alpha Click Threshold:
Transparent
Opaque

The Control Panel

If the Control Panel is not visible, choose Window > Control Panel.

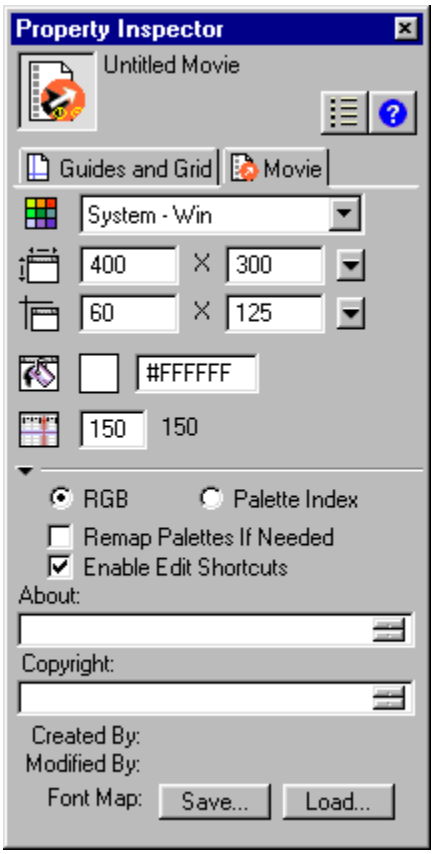
The Control Panel governs how movies play back in the authoring environment only.

You can also use the toolbar buttons or keyboard shortcuts to play a movie. To go to a specific frame number, enter the number in the frame counter and press Enter (Windows) or Return (Macintosh).

-

Setting Stage and movie properties

You use the Property Inspector's Movie tab to specify settings that affect the entire movie, such as how colors are defined, the size and location of the Stage, the number of channels in the Score, copyright information, and font mapping. These settings apply only to the current movie, whereas the settings you choose from File > Preferences apply to every movie.



To set Stage and movie properties:

1. Click the Movie tab of the Property Inspector in Graphical view. (Note that the Movie tab appears only if you do not have an object selected on the Stage or in the Score.)

2. To choose a color palette for the movie, select a palette from the Movie Palette pop-up menu. This palette remains selected until Director encounters a different palette setting in the palette channel.

   For a complete discussion of color palettes and using color in Director, see **Controlling color**.

3. To define the size of the Stage, choose a preset value from the Stage Size pop-up menu or manually enter values in the Width and Height fields.

4. To specify the location of the Stage during playback if the movie does not take up the full screen, choose an option from the Stage Location pop-up menu or enter values for Left and Top; these values specify the number of pixels the Stage is placed from the top left corner of the screen, and they apply only if the Stage is smaller than the current monitor's screen size.

- Centered places the Stage window in the center of your monitor. This option is useful if you play a movie that was created for a 13-inch screen on a larger screen, or if you're creating a movie on a larger

screen that will be seen on smaller screens.

- Upper Left places the Stage in the upper left corner of the screen.

5 To set the color of the Stage for the movie, double-click the color box next to Stage Fill Color and select a color, or enter an RGB value in the box on the right.

6 To specify the number of channels in the Score, enter a value for Score Channels.

7 To determine how the movie assigns colors, choose either RGB or Palette Index.

- RGB makes the movie assign all color values as absolute RGB values.
- Palette Index makes the movie assign color according to its position in the current palette.

8 To remap colors in bitmaps with different color palettes to colors in the current palette, select Remap Palettes If Needed.

This option dynamically remaps bitmaps on the Stage without changing the cast members. For example, if a cast member uses a grayscale palette, it is drawn on the Stage using the grays available in the common palette.

9 To let users cut, copy, and paste Editable fields while a movie is playing, select Enable Edit Shortcuts.

10 To enter copyright and other information about the movie, enter text in the About and Copyright boxes.

This information is important if your movie is going to be downloaded from the Internet and saved on a user's system.

11 To save the current font map settings in a text file named Fontmap.txt, click Save. To load the font mapping assignments specified in the selected font map file, click Load. See **Mapping fonts between platforms for field cast members**.

Increasing or decreasing your view of the Stage

You can author in Director on a zoomed Stage—one that is either larger or smaller than the normal size of the movie. Additionally, the Stage includes an offstage canvas area within the Stage window but outside of the active movie area. This canvas area is useful for assembling your media either before or after they appear on the Stage.

The offstage canvas is also useful as a way to preload media in projectors. Sprites in a frame, but offstage, are loaded into memory so they are ready to play in the subsequent frame.

When you change the size of the Stage, any guides or grids you use to assist you with alignment will also scale to the zoomed size, and you can manipulate Stage objects just as you would on a Stage that is not zoomed.

The Stage window does not need to be in front when you zoom in or out.

To zoom the Stage, do one of the following:

- Press Control+the plus (+) key (Windows) or Command+the plus (+) key (Macintosh) to zoom in and increase the Stage size. Press Control+the minus (-) key (Windows) or Command+the minus (-) key (Macintosh) to zoom out and decrease the Stage size. (In Windows, if you want to use the keys on the numeric keypad, NumLock must be off.)

  You can press the keys repeatedly until the Stage is the desired size.

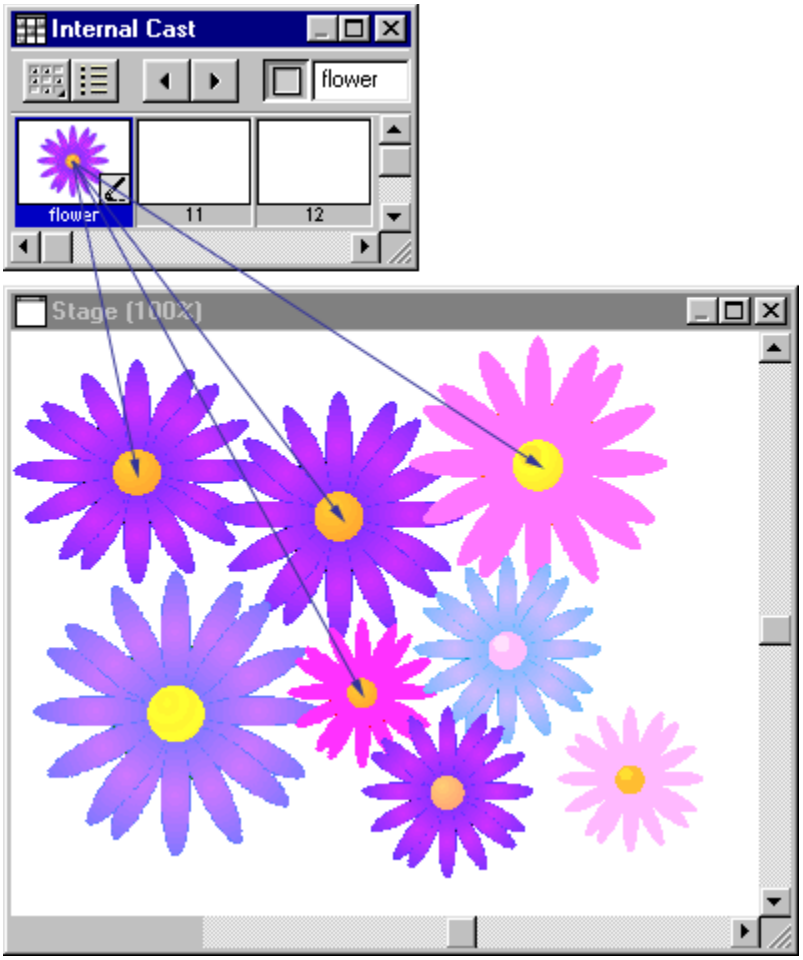
- Choose View > Zoom and select Zoom Stage In to increase the size of the Stage in increments, Zoom Stage Out to decrease Stage size, or a percentage to select a specific Stage size.
- If scroll bars are visible, click in the lower left corner of the scroll bar and choose a percentage. (To turn on scroll bars, choose File > Preferences > General and check Show Stage Scrollbars.)
- To zoom in while selecting an area of the Stage to center within the zoomed window, select the Magnifying Glass tool from the Tool palette. Click a point on the Stage to zoom and center.
- To zoom out while selecting an area of the Stage to center within the zoomed window, select the Magnifying Glass tool from the Tool palette. Press Alt (Windows) or Option (Macintosh) while clicking a point on the Stage to zoom and center.

The Stage's title bar indicates the zoom Stage size expressed as a percentage of the normal Stage size.

About Sprites

A sprite is an object that controls when, where, and how cast members appear in a movie. You create sprites by placing cast members on the Stage or in the Score. Creating a Director movie consists largely of defining where sprites appear, when they appear in the movie, how they behave, and what their properties are. Different sprites can be created from a single cast member. Each sprite can have its own values for different properties, and most changes to these properties do not affect the cast member. Most changes to a cast member, however, will change sprites created from that cast member.

For information on creating and changing sprites, see **Creating sprites**.



All these sprites display one bitmap image with different attributes.

About Channels in the Score

Channels are the rows in the Score that control your media. The Score contains sprite channels and special effects channels.

Sprite channels are numbered and contain sprites that control all visible media in the movie. Effects channels at the top of the Score contain behaviors as well as controls for the tempo, palettes, transitions, and sounds. The Score displays channels in the order shown here.

The first channel in the Score contains markers that identify places in the Score, such as the beginning of a new scene. Markers are useful for making quick jumps to specific locations in a movie. See **Using markers**.

While the Score can include up to 1000 channels, most movies use as few channels as possible to improve performance in the authoring environment and during playback. Sprites in higher channels appear on the Stage in front of sprites in lower channels. Use the Property Inspector's Movie tab to control the number of channels in the Score for the current movie. See **Setting Stage and movie properties**.

Turning on and off channels

To hide the contents of any channel on the Stage, or to disable the contents if they are not visible sprites, you use the button to the left of the channel. When you turn off a special effects channel, the channel's data has no effect on the movie. You should turn off Score channels when testing performance or working on complex overlapping animations. Turning off a channel has no effect on projectors or Shockwave.

To turn off a Score channel:

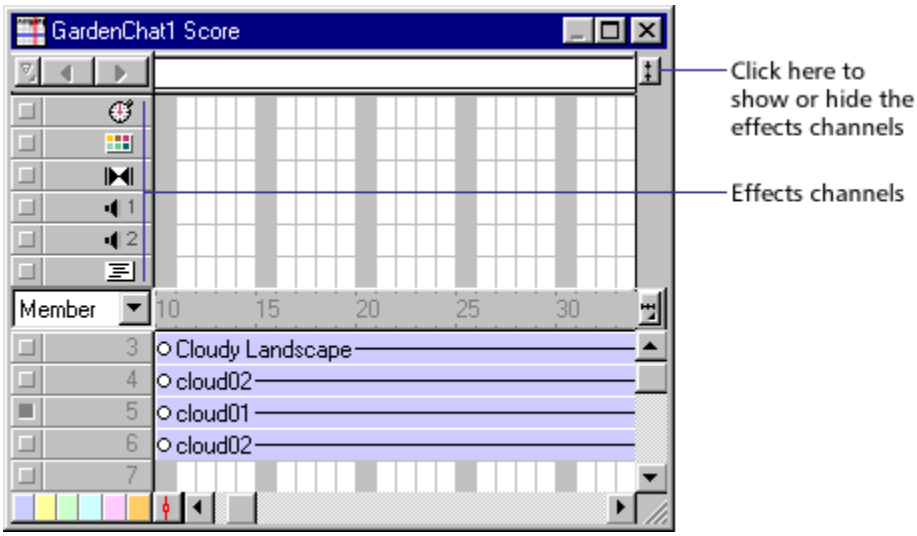
Click the gray button to the left of the channel. A darkened button indicates that the channel is off.

To turn multiple Score channels off and on:

Press Alt (Windows) or Option (Macintosh) and click a channel that is on to turn all the other channels off, or click a channel that is off to turn the other channels on.

To show or hide the special effects channels:

Click the Hide/Show Effects Channels button in the upper right corner of the Score to change the display.

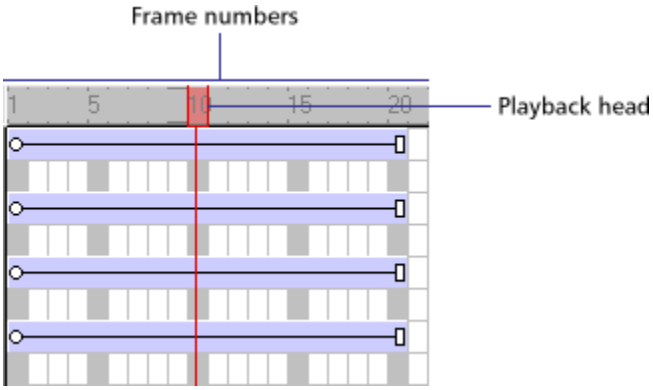


About Frames

A frame in a movie represents a single point in time, similar in theory to a frame in a celluloid film. Numbers listed horizontally in the sprite and special effects channels represent frames. Setting the number of frames displayed per second sets the movie's playback speed.

About the playback head

The playback head moves through the Score to show the frame currently displayed on the Stage. As you play your movie, the playback head automatically moves through your Score. You can also click any frame in the Score to move the playback head to that frame, and you can drag the playback head backward or forward through frames.

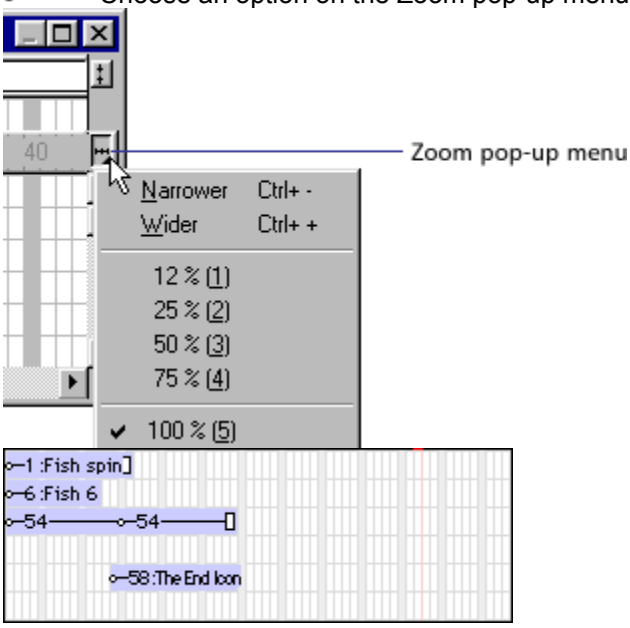


Changing your view of the Score

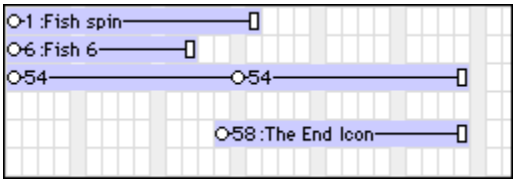
To narrow or widen the Score, you change the zoom percentage. Zooming in widens each frame, which lets you see more data in a frame. Zooming out shows more frames in less space, and is useful when moving large blocks of Score data.

To change the zoom setting, do one of the following:

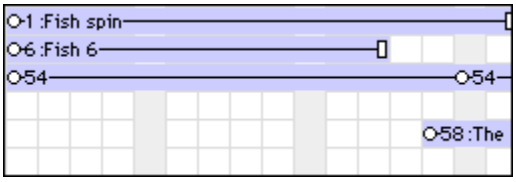
- Choose View > Zoom and then choose an option.
- Choose an option on the Zoom pop-up menu to the right of the Score.



Score zoomed out to 50%



Score at 100%



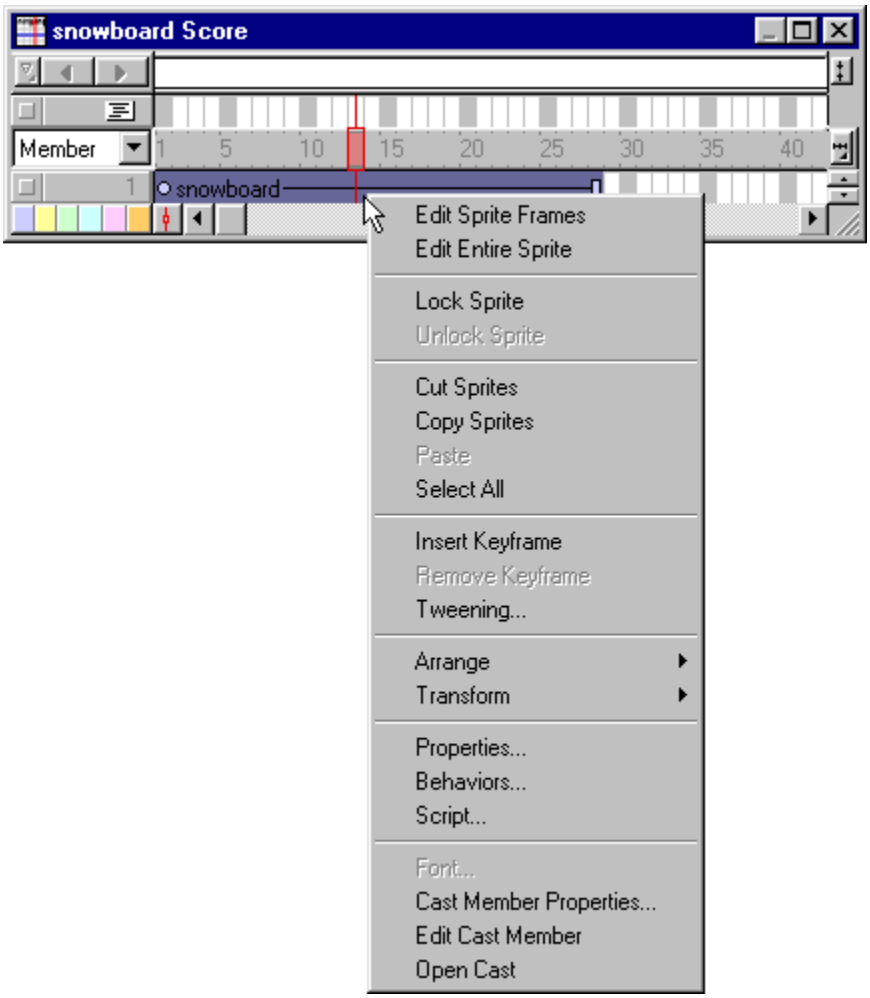
Score zoomed in to 200%

You can also display more frames in a Score without changing the zoom setting. To do so, place a sprite in the rightmost frame of the score. Director will automatically display additional frames in the current view of the Score.

Using context menus

To let you quickly access certain commands, Director provides menus that display commands relevant to a particular element. These are called context menus because the commands on the menu vary depending on the context in which the menu is displayed.

In the following illustration, Director is displaying the context menu for a sprite.



To display a context menu:

Position your mouse over an element and then right-click (Windows) or Control-click (Macintosh).

Using many Score windows

You can view and work in different parts of a movie at the same time by opening additional Score windows. If your sprite bars occupy many frames in the Score, for example, you can open a second Score window to work on another place in the movie without scrolling. You can also drag sprites from one Score window to another.

To open a new Score window:

1. Activate the current Score window.
2. Choose Window > New Window to create a second Score window.

You can scroll in this window to a different location in the Score. Only the first Score window automatically scrolls to show the playback head location.

Changing Score settings

To control the appearance of the Score and the information displayed in numbered sprite channels, you set preferences for the Score. By doing so, you can display a script preview and cast member information. In addition, if you are accustomed to older versions of Director, you can make the Score work as it did in Director 5.

To change Score settings:

1. Chose File > Preferences > Score.

2. The Extended display option lets you display information within sprites in the Score. See **Displaying sprite labels in the Score**. To specify what cast member information appears in numbered sprite channels when Extended display is on, choose from the following options:

- Cast Member displays the cast member number, name, or both.
- Behaviors displays the behaviors attached to the sprite.
- Ink Mode displays the type of ink applied to the sprite.
- Blend displays the blend percentage applied to the sprite.
- Location shows the sprite's *x* and *y* screen coordinates.
- Change in Location shows the change in *x* and *y* coordinates relative to the previous cast member in that channel.

3. To display the first few lines of the selected script in a box at the top of the Score, select Script Preview.

4. To display the cast member's name and number when the pointer is over a sprite for a few seconds, check Show Data Tips.

5. To make Score features work like those in Director 5 and earlier versions, choose from the following options:

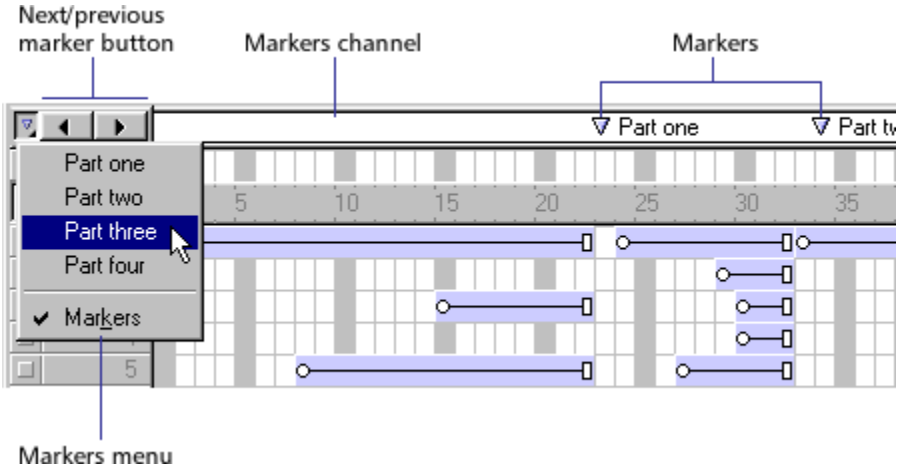
- Director 5 Style Score Display modifies the Director 8 Score window so it looks and behaves like the Director 5 Score window.
- Allow Drag and Drop makes sections of the Score moveable by dragging in the Director 5 style Score. To override this setting temporarily, press the Spacebar while the Score window is open.
- Allow Colored Cells displays a cell color selector at the left of the Score window so that you can choose a color for selected cells. Otherwise, the cell color selector is hidden, which improves performance when scrolling the Score window. If you've already applied color to cells, turning off this option hides cell colors but doesn't remove them.

Using markers

Markers identify fixed locations at a particular frame in a movie; you use markers when you're defining navigation. Using Lingo or draggable behaviors, you can instantly move the playback head to any marker frame. This is useful when jumping to new scenes from a menu or looping while cast members download from the Web. You can also use markers while authoring to advance quickly to the next scene.

Once you've marked a frame in the Score, you can use the marker name in your behaviors or scripts to refer to exact frames. Marker names remain constant no matter how you edit the Score. They are more reliable to use as navigation references than are frame numbers, which can change if you insert or delete frames in the Score.

You can use the Markers window to write comments associated with markers you set in the Score and to move the playback head to a particular marker.



To create a marker:

1. Click the markers channel.

   A text insertion point appears to the right of the marker.

2. Type a short name for the marker.

To delete a marker:

Drag the marker up or down and out of the markers channel.

To jump to markers while authoring, do any of the following:

- Click the Next Marker and Previous Marker buttons on the left side of the markers channel.
- Press the 4 and 6 keys on the numeric keypad to cycle backward and forward through markers.
- Choose the name of a marker from the Markers menu.

To move the playback head to a marker and enter marker comments:

1. Select a marker in the Score window and choose Window > Markers. The Markers window opens and displays comments associated with that frame.

2. Click a marker name in the list. Comments associated with markers appear in the right column.

Note: Use Control+Left Arrow or Control+Right Arrow (Windows) or Command+Left Arrow or Command+Right Arrow (Macintosh) to move to the previous or next marker.

3 To enter or edit comments, begin typing at the insertion point that appears in the right column.

By default, the marker name appears as the first line of text in the right column.

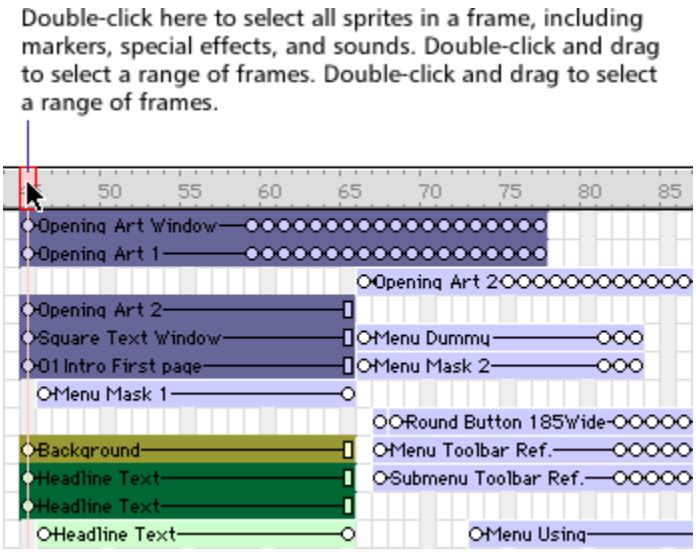
4 If you don’t want to edit the marker name, press Enter (Windows) or Return (Macintosh) to start a new line.

Selecting and editing frames in the Score

You can select a range of frames in the Score and then copy, delete, or paste all the contents of the selected frames.

To move, copy, or delete all the contents of a range of frames:

1 Double-click in the frame channel to select frames.



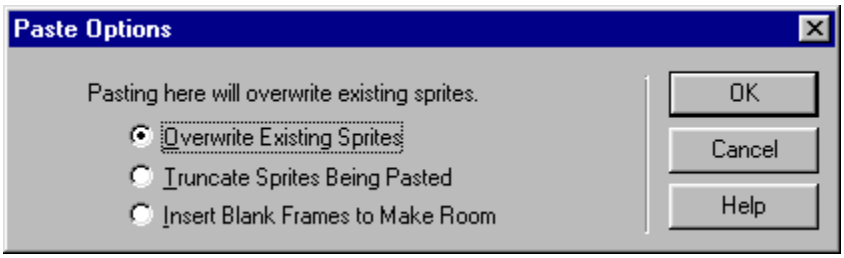
2 If you want to move or copy frames, choose Edit > Cut Frames or Edit > Copy Frames. If you want to delete frames, choose Edit > Clear Frames or press Delete.

If you cut, clear, or delete the selected frames, Director removes the frames and closes up the empty space.

Note: To delete a single frame, you can also choose Insert > Remove Frame.

3 To paste frames that you have cut or copied, select any frame and choose Edit > Paste Sprites.

If no frame is selected when you choose Edit > Paste Sprites, the Paste Options dialog box lets you decide how you want the frames to be pasted.



To add new frames:

1 Select a frame in the Score.

2 Choose Insert > Frames.

3 Enter the number of frames to insert.

The new frames appear to the right of the selected frame. Sprites in the frames you select are extended or tweened. (For more information about tweening, see **Animation: Overview**.)

About adding interactivity with Lingo

Lingo, Director's scripting language, adds interactivity to a movie. Lingo can accomplish many of the same tasks—such as moving sprites on the Stage or playing sounds—that you can accomplish using the Director interface.

Much of Lingo's usefulness, however, is in the flexibility it brings to a movie. Instead of playing a series of frames exactly as the Score dictates, Lingo can control the movie in response to specific conditions and events. For example, whether a sprite moves can depend on whether the user clicks a specific button; when a sound plays can depend on how much of the sound has already streamed from the Internet.

Director includes a set of prepackaged Lingo instructions, called behaviors, that you can simply drag to sprites and frames. Behaviors let you add Lingo's interactivity without writing Lingo scripts yourself. You can modify behaviors or create your own. For more information about the behaviors included with Director 8, see **Using Director 8 Behaviors** in the Director Support Center.

If you prefer writing scripts to using the Director interface and behaviors, Lingo provides an alternative way to implement common Director features; for example, you can use Lingo to create animation, stream movies from the Web, perform navigation, format text, and respond to user actions with the keyboard and mouse.

Writing Lingo also lets you do some things that the Score alone can't do. For example, Lingo's lists let you create and manage data arrays, and Lingo operators let you perform mathematical operations and combine strings of text.

For more general information about Lingo, see **Writing scripts with Lingo: Overview**.

Converting movies created in previous versions of Director

Director 8 can convert movies from Director 6 and 7. You can also update movies to Director 8 by simply opening and saving them, but the Update Movies command is faster for converting large projects. It's also more effective for preserving links to external media. See **Processing movies with Update Movies**.

Note: The Director 8 Shockwave player can play Shockwave movies created with Director 5, 6, 7, and 8.

When you open a Director 6 or 7 movie in Director 8, or convert it to the new format with Update Movies, note the following:

- The data structure is changed to the latest file format.
- Shapes are not converted to the new Bézier shapes.
- Ink functionality is not updated unless you turn off Maintain Outdated Ink Mode Limitations in the Movie Properties dialog box.
- Old Score data, from versions of Director prior to Director 5, is converted to the new Score, combining adjacent frames in the old Score containing the same cast members into single sprites in the new Score. You may want to split or join sprites to make working in the Score more convenient.

Setting general preferences

To control some of Director's default settings relating to the Stage and the user interface, you can use the General Preferences dialog box. These settings control the appearance of movies only in the authoring environment, not during playback.

To set Director default values:

1. Choose File > Preferences > General.
2. To specify the default size and location of the Stage and the way it animates when deactivated, choose Stage options.

   The Stage location settings affect the location of the Stage only in the authoring environment. To set the location of the Stage during playback, see **Setting Stage and movie properties**.

- Use Movie Settings sets the Stage size to the movie's Stage size and location.
- Match Current Movie opens a new movie in the Stage size of the movie that's currently open.
- Center positions the Stage in the center of the screen by default, which is useful if the Stage size is smaller than the screen size. Otherwise, the movie plays using its original Stage position.
- Reset Monitor to Movie's Color Depth (Macintosh only) automatically changes the color depth of your monitor to the color depth of a movie when it is opened in the authoring environment. See **Changing the color depth of a movie**.
- Animate in Background runs animation in the background while you are working with other applications. When you are running animation in the background, the Stage remains on the screen, and the active application window appears in front of the Stage.

3. To set defaults for the Director user interface, choose User Interface options:

- Dialogs Appear at Mouse Position displays dialog boxes at the mouse position. If this option is not selected, dialog boxes are centered on the monitor that contains the menu bar.
- Save Window Positions on Exit saves the positions of all open windows every time you quit so they reappear in the same location when you start again.
- Message Window Recompiles Scripts makes Director recompile all scripts when you press Enter (Windows) or Return (Macintosh) in the Message window. With this option off, Director recompiles scripts only when you choose Control > Recompile All Scripts.
- Show Tooltips controls the definitions that appear when the pointer is over tools and buttons. Turn off this option to stop definitions from appearing.
- Show Stage Scrollbars makes scroll bars appear in the Stage window.

4. To specify the unit of measure to use in the text ruler, choose inches, centimeters, or pixels from the Text Units pop-up menu. See **Formatting paragraphs**.
5. (Macintosh only) To allow Director to use available memory beyond the amount allocated to it, turn on Use System Temporary Memory.

Choosing Internet connection settings

Director can connect to the Internet to import cast members and retrieve data. It also launches a browser to preview movies and open those Director Help topics that are on http://www.macromedia.com. Use settings in the Network Preferences dialog box to control how the connection works and to define a preferred browser.

To choose Internet connection settings:

1. Choose File > Preferences > Network.
2. To specify the browser to launch when a movie running in the authoring environment encounters the `gotoNetPage` Lingo command, enter the path to the browser in the Preferred Browser field.

   To locate the browser, click the Browse button.
3. To enable or disable browser launching, select Launch When Needed.
4. To specify the amount of space that Director can use to cache data from the Internet on your hard disk, enter a value in the Disk Cache Size field.
5. To immediately empty the cache, click Clear.
6. To specify how often cached data is compared to the same data on the server, choose a Check Documents option:

- Once Per Session checks for data revisions only once from the time you start to the time you quit the application. This option improves performance but may not always display the most current version of a page.
- Every Time checks for changes whenever you request a page. This option slows performance but assures you are always viewing the most current version of a page.

7. To specify the configuration of your system's proxy server, choose a Proxies option.

   Browsers usually don't require proxy servers to interact with the network services of external sources, but in some network configurations, where a firewall blocks the connection between the browser software and a remote server, interaction with a proxy may be required.

   A firewall protects information in internal computer networks from external access, and in doing so, it may limit the ability to exchange information. To overcome this limitation, browser software can interact with proxy software. A proxy server interacts with the firewall and acts as a conduit, providing a specific connection for each network service protocol. If you are running browser software on an internal network from behind a firewall, you will need the name and associated port number for the server running proxy software for each network service.

- No Proxies specifies that you have a direct connection to the Internet.
- Manual Configuration controls proxy settings for your system. Enter the HTTP or FTP URL and port number.

Printing movies

You can print movie content to review it and mark changes, to distribute edits to a team, to make handouts from a presentation, or just to see your work on paper. You can print a movie while in authoring mode in a variety of ways. You can print an image of the Stage in standard or storyboard format, the Score, the cast member number and contents of text cast members in the Cast window, all scripts or a range of scripts (movie, cast, Score, and sprite scripts), the comments in the Markers window, the Cast window artwork, or the entire Cast window.

Note: Using Cast Text on the Print pop-up menu, you can print a table of text cast members at the resolution of your printer.

You can also use Lingo to control printing. See **printFrom**.

To print part of a movie:

1. Choose File > Print.
2. To specify what part of the movie to print, choose an option from the Print pop-up menu.

   You can print an image of the Stage, the Score, all scripts or a range of scripts (movie, cast, Score, and sprite scripts), cast text, cast art, cast thumbnails, and the comments in the Markers window.

   The Scripts, Cast Text, Cast Art, and Cast Thumbnails print options specify a range of casts and cast members—internal or external. Information displayed in the Print dialog box depends on the selection to be printed.
3. To specify which frames of your movie are printed, choose a Frames option:

- Current Frame prints the frame that is currently on the Stage.
- Selected Frames prints the frames that are selected in the Score.
- All prints all the frames in your movie.
- Range prints the range of frames specified in the Begin and End boxes.

4. To specify which frames in the defined range to print, choose an Include option:

- Every Frame is the default setting and prints every frame specified in Range.
- One in Every _ Frames prints frames at the interval you specify in the box. For example, if you type 10, Director prints every tenth frame.
- Frames with Markers prints only the frames that have markers in the Score window.
- Frames with Artwork Changes in Channel _ prints the frames in which cast members move or in which new cast members are introduced in the Score. Specify the channel in the box.

5. To determine the layout of the items to print, click Options and choose from the following:

- Scale provides options to print at 100%, 50%, or 25% of the original size.
- Frame Borders creates a border around each frame.
- Frame Numbers prints the frame number with each frame.
- Registration Marks places marks on every page to align the page for reproduction.
- Storyboard Format is available only when you select 50% or 25% images to print. This option places marker comments next to the frame image.
- Date and Filename in Header prints a header on each page. The header consists of the name of the Director movie and the current date.
- Custom Footer prints a footer on each page. Type the footer in the field.

  The image at the left of the dialog box previews the layout options.

Monitoring memory use

The Memory Inspector displays information about how much memory is available to Director for your movie and indicates how much memory different parts of the current movie use and the total disk space the movie occupies. It also can purge all removable items from RAM if you are about to perform a memory-intensive operation.

To use the Memory Inspector:

1. Choose Window > Inspectors > Memory.
2. Observe the following memory use indicators:

- Total Memory displays the total system memory available, including the amount of RAM installed on your computer and any virtual memory available.
- Physical memory (Windows only) shows the amount of actual RAM installed in the system.
- Partition Size (Macintosh only) shows the amount of memory allocated to Director in the Get Info box, and is available only if Temporary Memory is enabled.
- Total Used indicates how much RAM is being used for a movie.
- Free Memory indicates how much more memory is currently available in your system.
- Other Memory indicates the amount of memory taken up by other applications.
- Used by Program indicates the amount of memory used by Director (excluding the amount of memory taken up by the Director application file itself).
- Cast and Score indicates the amount of memory used by the cast members in the Cast window and the notation in the Score window. Cast members include all the artwork in the Paint window, all the text in the Text windows, cast members that use the Matte ink in the Score, thumbnail images in the Cast window, and any sounds, palettes, buttons, digital video movies, or linked files imported into the cast and currently loaded into memory.
- Screen Buffer shows how much memory Director reserves for a working area while executing animation on the Stage.

3. To remove all purgeable items from RAM, including all thumbnail images in the Cast window, click Purge.

   All cast members that have Unload (purge priority) set to a priority other than 0–Never (as specified in the Member tab of the Property Inspector) are removed from memory. This procedure is useful for gaining as much memory as possible before importing a large file. Edited cast members are not purged.

About using Xtras to extend Director functionality

Xtras are software components that extend Director functionality; some Xtras are installed with Director and others are available through third-party developers. Xtras provide capabilities such as importing filters and connecting to the Internet. You can use preexisting Xtras and, if you know the C programming language, you can create your own Xtras.

For information on creating Xtras, download the Xtras Developer's Kit from the Director Support Center.

You must distribute any Xtra that a movie requires along with the movie itself. Xtras can be packaged with projectors, or your user can download your required Xtras from the Internet. See **Managing Xtras for distributed movies**.

If your user is missing an Xtra that Director requires, an alert appears when the movie opens. For missing Xtra transition cast members, the movie performs a simple cut transition instead. For other missing Xtra cast members, Director displays a red X as a placeholder.

Types of Xtras

Five types of Xtras are supplied with Director: cast member Xtras, importing Xtras, scripting Xtras, transition Xtras, and tool Xtras.

- Cast member Xtras provide new media types to Director. They create or control a wide range of objects for use as cast members.

  Some of the cast member types built into Director, such as Shockwave Flash, Vector Shape, and Animated GIF, are provided as Xtras. Xtras provided by third-party developers can include databases, 3D graphics processors, special types of graphics, and so on. Cast member Xtras built into Director appear on the Insert > Media Element menu. Other cast member Xtras may not appear on this menu and may require Lingo implementation.

  When setting properties for an Xtra cast member, use the Property Inspector, which provides settings standard to all types of Xtra cast members. If there are settings that are unique to the current Xtra, you must click Option to open a second Properties dialog box that lets you change those settings.

  Some cast member Xtras have separate authoring and playback components. You should include only the playback components when distributing movies.

- Importing Xtras provide the code required to import various types of media into Director. When you link a movie to an external file, Director uses the importing Xtra to import the media every time the movie runs. To distribute a movie with external linked media, you must also include the Xtra required to import that type of media.
- Scripting Xtras add Lingo elements to predefined Lingo scripts. The NetLingo Xtra, for example, provides special Lingo elements for controlling Internet functions.
- Transition Xtras supply transitions in addition to the predefined transitions available in the Frame Properties: Transition dialog box.
- Tool Xtras provide useful functions in the authoring environment, but they don't do anything while a movie runs. They do not have to be distributed with movies.

Installing Xtras

To make custom Xtras available to Director, place them in the Xtras folder located in the same folder as the Director application. You must do this before you launch Director.

An Xtra can be in a folder within the Xtras folder up to five layers deep.

When you launch Director, you can use the `openXlib` command to open Scripting Xtras located in any folder. If you open an Xtra this way, you must use the `closeXlib` command to close it when Director has finished with it.

Copies of the same Xtra can have different file names or have the same file name but reside in different folders. If duplicate Xtras are available when Director launches, Director displays an alert. Delete any duplicate Xtras.

Director automatically closes Xtras when the application quits.

To make any Director movie appear on the Xtras menu and open as a movie in a window during authoring, place it in the Xtras folder.

About distributing movies

When you finish creating a movie, you have several choices about how to distribute it to users. You can distribute the movie as a Shockwave movie that plays within a Web page or as a projector that downloads to the user's computer or that you distribute on a disk.

- A Shockwave movie is a compressed version of the movie data only.
- A projector is a stand-alone version of a movie. You can include several movies in a single projector. Projectors appear on the system desktop as applications.

For more information about distributing movies, see **Packaging Movies for Distribution: Overview**.

Movies distributed from the Internet can begin playing as soon as the content for the first frame is downloaded. This is called streaming. You can control streaming with behaviors that make the movie wait for media at certain frames, or you can specify that a movie download completely before it begins playing. See **Setting movie playback options**.

To create a Shockwave movie that can play in a Web page, you use the Publish command. Director leaves your original movie in its DIR format. Director also creates a Shockwave movie in the DCR format.

If you use the default Publish settings, Director creates an HTML page completely configured with `EMBED` tags and everything else you need to run your movie in a browser. Director saves all of these new files, by default, in the same folder as your original Director movie. For more information about putting your Director movie on the Web, see **Creating Shockwave movies**.

For information on how to distribute Xtras with projectors, refer to TechNote 13965 in the Director Support Center. Although the note may refer to Director 7, the information is the same for Director 8.

Cast members and Cast windows: Overview

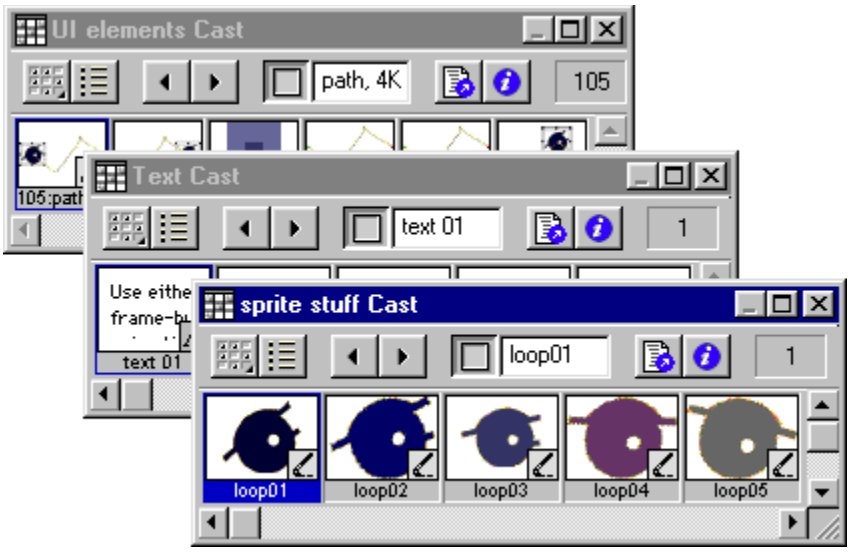
Cast members are the media and other assets in your movie. They can be bitmaps, vector shapes, text, scripts, sounds, Flash movies, QuickTime movies, AVI videos, and more. When you place a cast member on the Stage or in the Score, you create a sprite. For more information on sprites, see **Sprites**.

You use windows called casts to group and organize your cast members. To populate casts, you import and create cast members. You can create and use multiple casts in a movie.

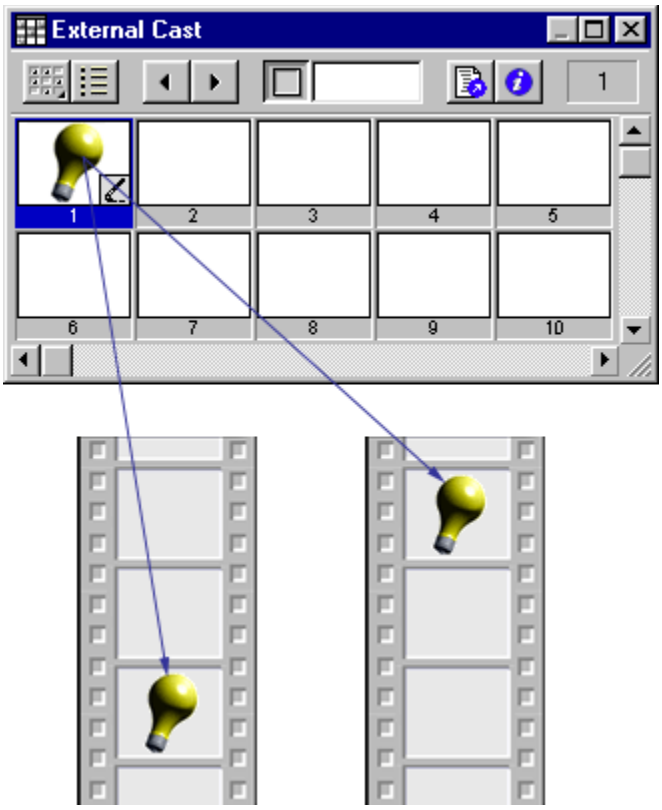
You can create and edit cast members in Director using basic tools and media editors such as the Paint and Text windows, and you can also edit cast members using external editors. In addition, you can import cast members from nearly every popular media format into a movie file and link cast members to external files, for some media types, on a disk or the Internet for dynamic updating.

The Property Inspector contains asset management fields for cast members on the Member tab. These fields let you name your cast members, add comments about them, and view information such as creation and modification dates, and file size.

Casts can be internal—stored inside the movie file and exclusive to that movie—or external—stored outside the movie file and available for sharing with other movies. When you create a new movie, an empty internal cast is automatically created, and when you open the Cast window it is in the default List view.



Internal casts



External casts

External casts are also useful for creating groups of commonly used cast members that you can switch while the movie plays, such as when you want to switch the language used in a movie. Using external casts can keep the movie size small for downloading; an external cast can download separately from the movie file if or when it is needed.

Creating new casts

Before assembling a large number of cast members, it's good practice to create the casts necessary to keep them organized. You can sort casts by type, edit cast properties, and use external casts for storing and sharing common media elements.

You can create as many casts as necessary; the number of casts does not affect the size of a movie for downloading.

You can include up to 32,000 cast members in a single cast, but it's usually best to group media such as text, buttons, and images logically in a few different casts for each movie.

To create a new cast:

1. Choose File > New > Cast.
2. Type a name for the new cast.
3. Specify how the cast is stored:

- Internal stores the cast within the movie file. This option makes the cast available only to the current movie.
- External stores the cast in a separate file outside the movie file. This option makes the cast available for sharing with other movies. For information about internal and external casts, see **Managing external casts**.

4. If you chose External and you don't want to use the cast in the current movie, deselect the Use in Current Movie option.
5. Click Create.

   The cast is created, and a Cast window for the cast is opened in List view. See **Using the Cast window**.

6. If you created an external cast, choose File > Save while its Cast window is active, then save the cast in the desired directory.

Creating cast members

You can create several types of cast members in Director, which includes editors to create common media such as text, shapes, and bitmaps. You can also use Director for basic editing of media imported from other applications. You define external editors to launch from within Director when you double- click a cast member, and edit almost any type of supported media. See **Launching external editors**.

You can also import cast members. See **Importing cast members**.

To create a new cast member from the Insert menu:

1 Open the Cast window for the cast member you are creating.

To place a cast member in a specific position, select the position in Thumbnail view. See **Using Cast Thumbnail view**. Otherwise, Director places the new cast member in the first empty position or after the current selection in the Cast window.

2 Choose Insert > Media Element and then choose the type of cast member to create.

For more information on each choice, see the following sections:

- **Using the Paint window**
- **Using the Color Palettes window**
- **Streaming linked Shockwave Audio and MP3 audio files**
- **Creating text cast members**
- **Embedding fonts in movies**
- **Creating an animated color cursor cast member**
- **Using animated GIFs**
- **Drawing vector shapes**
- **Using Flash Movies**

3 To create a control or button, choose from the following options:

- Choose Insert > Control > Field to create a field cast member. Creating a field cast member also creates a sprite on the Stage. See **Working with fields**.
- Choose Insert > Control > Push Button, Radio Button, or Check Box to create a button cast member and a sprite on the Stage. See **Using shapes**.
- (Windows only) Choose Insert > Control > ActiveX to create an ActiveX cast member. See **Using ActiveX controls**.

To create a cast member in a media editing window:

1 Open a media editing window by choosing Window and then choosing the type of cast member you want to create (Paint, Text, Script, and so on).

2 Click the New Cast Member button to create a cast member of the corresponding type. The cast member is added to the most recently active Cast window.

Using the Cast window

In the Cast window, you can view the cast in either the default List view or the Thumbnail view. (You can change the default so that the Cast window opens in Thumbnail view. See **Setting Cast window preferences**.)

The Cast window lets you do the following:

- Organize and display all media in a movie.
- Move groups of cast members.
- Launch editors for cast members.
- Launch the Property Inspector to view, add, and change comments about your cast members, and to view and modify cast member properties.

To view the Cast window:

Choose Window > Cast or press Control+3 (Windows) or Command+3 (Macintosh). If there is more than one cast in the movie, you can choose which Cast window to open by choosing Window > Cast and then selecting a cast name from the Cast submenu.

Switching from one Cast window view to another

You can easily toggle between List and Thumbnail views of the Cast window.

To switch from one Cast window view to another, do one of the following:

- Click the Cast View Style icon on the Cast window to toggle between the two views.
- With the Cast window active, choose View > Cast and select either List or Thumbnail, as desired.
- Right-click (Windows) or Control-click (Macintosh) the Cast window and select either List or Thumbnail, as desired, from the context menu.

Using Cast window controls

The controls along the top of the Cast window are the same for both the Cast List and the Cast Thumbnail views. You use the controls to change the cast displayed in the Cast window, the cast member selection, or the name of a cast member. You can also use them to move cast members and to open a cast member's Script window or the Property Inspector.

To change the cast displayed in the current Cast window:

Click the Cast button and choose a cast from the pop-up menu.

To open a cast in a new Cast window:

Right-click (Windows) or Control-click (Macintosh) the Cast button and choose a cast from the context menu.

To select the previous or next cast member:

Click the Previous Cast Member or Next Cast Member button.

To move a selected cast member to a new position in the Cast window (Thumbnail view) or to the Stage:

Drag from the Drag Cast Member button to the desired position in the Cast window or on the Stage.

This procedure is useful when the selected cast member has scrolled out of view.

To enter a cast member name:

Select a cast member and enter the name in the Cast Member Name box.

To edit a cast member script:

Select a cast member and click the Cast Member Script button.

To view cast member properties:

1 Select a cast member.

2 Do one of the following:

- Click the Cast Member Properties button.
- Right-click (Windows) or Control-click (Macintosh) and choose Cast Member Properties from the context menu.
- Choose Window > Inspectors > Property.

See **Viewing and setting cast member properties**.

To view the cast member number:

Refer to the Cast Member Number field.

Selecting cast members in the Cast window

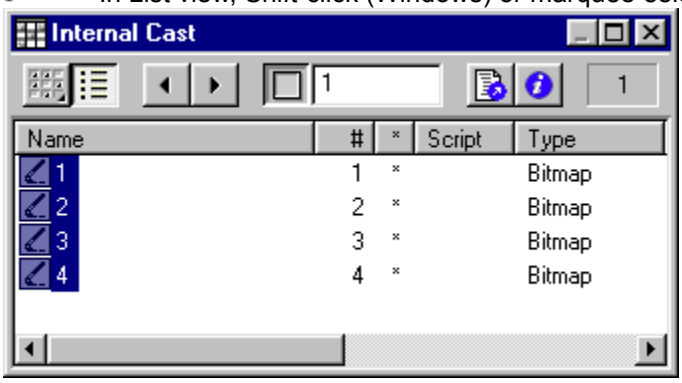
Before changing, sorting, or moving cast members, you must select them in the Cast window.

To select a single cast member, do one of the following:

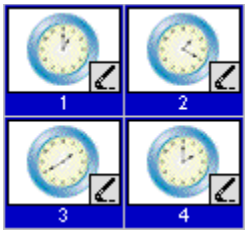
- In List view, click the name or icon (Windows) or click any part of the text or icon (Macintosh).
- In Thumbnail view, simply click the thumbnail image.

To select multiple adjacent cast members, do one of the following:

- In List view, Shift-click (Windows) or marquee-select the cast members (Macintosh)

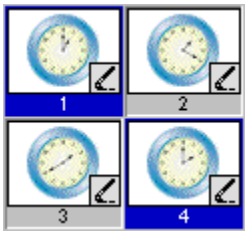


In Thumbnail view, click the first cast member in the range and then Shift- click the last cast member in the range.



To select multiple nonadjacent cast members:

In either List or Thumbnail view, Control-click (Windows) or Command-click (Macintosh) each cast member that you want to select.



Copying cast members

You can easily create multiple versions of a cast member in a single cast. For example, you may want several cast members to be identical except for color or size. You can also copy cast members from one Cast window to another.

To copy a cast member:

1. In either List or Thumbnail view, select the cast member (or multiple cast members) that you want to copy.
2. Alt-click (Windows) or Option-click (Macintosh) and drag the cast member to a new location in Thumbnail view or to the bottom of the list in List view.

   You can drag the cast member to a location in the same Cast window or to a different Cast window. Director creates a cast member with a new number but with all other information identical to the original.
3. If you copied the cast member into the same Cast window, change the name of the copied cast member so that you (and Lingo scripts) can distinguish it from the original. See **Naming cast members**.

Naming cast members

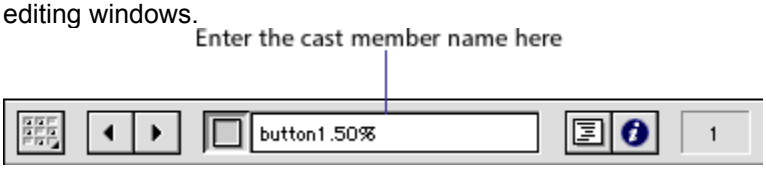
To avoid problems in Lingo when referring to cast members, you should name them and refer to them by name. Naming cast members doesn't affect Director performance. The name stays the same even if the cast member number changes.

Avoid duplicating cast member names. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast.

To name a cast member:

Select the cast member in either the List or the Thumbnail view of the Cast window and do one of the following:

- Enter a name in the Cast Member Name field at the top of the Cast window or in any of the editing windows.



- Enter a name in the Name field on the Cast or Member tab of the Property Inspector.

To name a cast member using Lingo:

Set the name cast member property. See **name (cast member property)**.

Using Cast List view

Cast List view, the default view in which the Cast window opens, provides seven columns of information by default. They are as follows:

Column Title	Column Information
Name	The name of the cast member and an icon describing the cast member type. For more information on what the icons represent, see **Using Cast Thumbnail view**.
#	The number assigned to the cast member. Note that this number represents the order in which this cast member appears in Thumbnail view.
*	An asterisk in this column indicates the cast member has changed but you have not yet saved those changes.
Script	The word *Member* in this column means the cast member contains a script. The word *Movie* in this column means the cast member is a movie script. The word *Behavior* in this column means the cast member is a Behavior. You can use the Script icon to view the script or behavior.
Type	The cast member type
Modified	The date and time the cast member was changed
Comments	Displays text entered in the Property Inspector's Member tab, in the Comments field

Four additional columns are available via the Cast Window Preferences dialog box. See **Setting Cast window preferences**. The additional columns that you can display are as follows:

Column Title	Column Information
Size	The size in bytes, kilobytes, or megabytes
Created	The date and time the cast member was created
Modified By	Who modified the cast member. This value comes from the user login name (Windows) or the Sharing setup name (Macintosh).
Filename	The full path to the cast member if it is a linked asset

Resizing columns in Cast List view

You can resize the columns holding the pointer over the column boundary to activate the Resizing tool. Click and drag the column to the desired size.

Sorting Cast List view columns

You can sort the Cast List view columns in ascending and descending order by clicking the column title. When you sort the Cast List window by clicking the column title, you're only changing the way in which the information is displayed. You are not changing any cast member attributes.

About cast member order in Cast List view

Unlike the way in which cast members appear in Thumbnail view, in List view the cast member order does not always correspond to the member's physical location in the cast.

When working in List view, also keep in mind the following:

- In List view, Director places new cast members at the end of the list, and the cast member number becomes the first available number after the current selection.
- You can use Thumbnail view to reorder (and renumber) cast members by dragging them to different locations in the window; you cannot reorder cast members by dragging in List view.

Using Cast Thumbnail view

As the name suggests, the Cast Thumbnail view shows a very small (thumbnail) version of the cast member along with an icon that represents the cast member media type:

Icon	Cast member type	Icon	Cast member type
	Animated GIF		Behavior
	Bitmap		Button
	Check box		Custom Cursor
	Digital video		Field
	Film loop		Font
	Flash movie		Linked bitmap (all linked cast member icons are changed in the same way)
	OLE		Palette
	PICT		QuickTime video
	Radio button		Script
	Shape		Shockwave Audio
	Sound		Text
	Transition		Vector shape
	Xtra		

To turn off or on the display of cast member icons in Thumbnail view and change the Cast window display:

Choose File > Preferences > Cast; see **Setting Cast window preferences**.

Creating a custom cast member thumbnail

For most cast members, Director displays a scaled version as the thumbnail unless you define a custom thumbnail. Creating a custom thumbnail is most useful for behaviors that you want to identify in the Library palette, because behaviors have no identifying image.

To create a custom cast member thumbnail:

1. Select the bitmap image to use as the new thumbnail and copy it to your system's clipboard.

   You can copy the image from any bitmap editor, including the Paint window. The image can be of any size, but smaller images look better because they require less scaling.

2. Using Thumbnail view, place the pointer over the cast member for which you are creating a custom thumbnail.

3. Right-click (Windows) or Control-click (Macintosh) and choose Paste Bitmap from the context menu.

   The image from the clipboard replaces the current cast member thumbnail.

   You can also use text as a thumbnail. Select text instead of a bitmap image in step 1, and then choose Paste Text from the context menu.

Moving cast members within the Cast window

To move a cast member to a new position within the Cast window, you can use Thumbnail view to see the representation of the cast member's position.

Note: When you move a cast member to a new position, Director assigns it a new number and updates all references to the cast member in the Score, but it doesn't automatically update references to cast member numbers in Lingo scripts. The best practice, therefore, is to always name cast members and refer to them by name in Lingo scripts.

To move a cast member to a new position or a different cast:

- Using Thumbnail view, drag the cast member to a new position in any open Cast window.

In Thumbnail view, a highlight bar indicates where the cast member will be placed. If you drag the cast member over a position that already contains a cast member, Director places your selected cast member in that position and moves the existing cast member one position to the right.

In List view, the cast member is added to the bottom of the list.

To cut, copy, and paste cast members to a new position or a different cast:

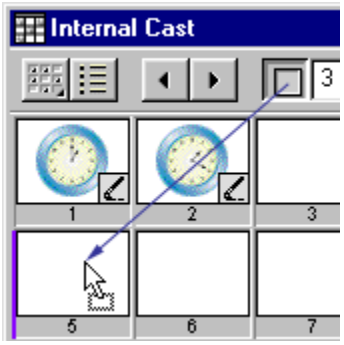
1. Select one or more cast members, then choose Cut or Copy from the Edit menu.
2. Do one of the following:

- In Thumbnail view, select an empty position in any open Cast window, and then choose Edit > Paste.
- In List view, deselect all cast members by clicking anywhere in the window except on a cast member name. Then choose Edit > Paste.

Note: In either Thumbnail or List view, if you paste cast members while other cast members are selected, you will overwrite the selected cast members.

To move a cast member to a position not currently visible in Thumbnail view:

1. Select the cast member you want to move.
2. Scroll the Cast window to display the destination position.
3. Drag from the Drag Cast Member button to the destination position.



Organizing cast members within the Cast window

The Sort command in the Modify menu helps clean up and organize the Cast window. Use Sort to order cast members by their media type, name, size, or usage in the Score. You can also use Sort to remove empty positions in a Cast window.

When you use the Sort command to sort a Cast window, Director can move cast members to new positions, with new cast member numbers.

Note: If you've written scripts that refer to cast members by number, Lingo won't be able to find cast members that have been moved. To avoid this problem, always name your cast members and refer to them by name in your scripts.

If you want to view the cast members in a different sort order without changing cast member numbers, click a column title in Cast List view. See **Sorting Cast List view columns**.

To sort the cast using the Modify menu:

1. With the Cast window active, select the cast members to sort or choose Edit > Select All.
2. Choose Modify > Sort.
3. In the Sort Cast Members dialog box, choose a sorting method.

- Usage in Score places selected cast members used in the Score at the beginning of the selection.
- Media Type groups all cast members according to their media type.
- Name groups the selection alphabetically by cast member name.
- Size arranges the selection with the largest files appearing first.
- Empty at End places all empty cast positions in the selection at the end.

4. Click Sort.

   Director reorders the cast members according to the sorting method you selected. The Score automatically adjusts to the new cast member numbers.

Setting Cast window preferences

You use the Cast window preferences settings to control the appearance of the current Cast window or, if desired, all Cast windows. You can set different preferences for each Cast window. The title bar of the dialog box displays the name of the Cast window preferences you are changing.

To set Cast window preferences:

1. Select a Cast window to change.
2. Choose File > Preferences > Cast.
3. To set the Cast window to display in either List or Thumbnail view, select the appropriate Cast View option.
4. If you want your preferences to apply to all Cast windows, select Apply to All Casts.
5. To select the columns that appear in Cast List view, select the desired Columns in List options. See **Using Cast List view**.
6. To specify the maximum number of cast members displayed in the Cast window, choose a value from the Thumbnails Visible pop-up menu.

   Note that this option does not limit the actual number of cast members that can exist in the cast. If you have a small number of cast members, you can hide the remaining unused cast positions to make better use of the vertical scroll bar. The default is 1000.
7. To specify the number of thumbnails in each row of the Cast window, choose an option from the Row Width pop-up menu.

   The options for 8 Thumbnails, 10 Thumbnails, and 20 Thumbnails specify fixed-row widths that are independent of the window size; if the Cast window is smaller horizontally than the width of the cast row, you must use the horizontal scroll bar to reveal the rest of the cast. The Fit to Window option automatically adjusts the number of cast members per row to fit the current width of the Cast window. In this mode, the horizontal scroll bar is disabled since the entire width of the cast is always in view. The default is Fit to Window.
8. To set the size of each cast thumbnail image displayed in the Cast window, choose an option from the Thumbnail Size pop-up menu:

- Small 44 x 33 pixels
- Medium 56 x 42 pixels (default)
- Large 80 x 60 pixels

Thumbnails always maintain the standard 4:3 aspect ratio.

If the thumbnails appear fuzzy, they are probably displaying larger than their original size. To correct this, change the Cast window preferences thumbnail setting to a smaller size. Click OK when the alert message asks if thumbnails should be regenerated.

9. To select the display format of the cast member ID displayed below each cast thumbnail image in the Cast window, choose an option from the Label pop-up menu:

- Number displays the cast number.
- Name displays the cast name, if one exists; otherwise, this option displays the cast number in decimal format.
- Number:Name displays the cast number (in decimal format) and cast name, separated by a

colon: for example, 340:Dancing Potato. If no name exists, this setting displays just the cast number in decimal format.

The chosen format is also used in other windows, including the Score, whenever a cast ID is displayed.

10 To specify whether Director displays an icon in the lower right corner of each cast member indicating the cast member's type, choose one of the following from the Media Type Icons pop-up menu: All Types, All but Text and Bitmap, or None.

11 To display a script indicator icon in the lower left corner of each cast member that has a script attached, select Script.

12 To make your preference settings the default settings, click Save as Default.

13 When you finish selecting your preferences, click OK.

Changing Cast properties

You use the Property Inspector to change the name of a Cast and to define how its cast members are loaded into memory.

To change Cast window properties:

1. With the Cast window as the active window, do one of the following:

- If the Property Inspector is not open, choose Window > Inspectors > Property, then click the Cast tab and display the Graphical view.
- If the Property Inspector is open, click the Cast tab and display the Graphical view.
- Choose Modify > Cast Properties.
- Right-click (Windows) or Control-click (Macintosh) in the Cast window and choose Cast Properties from the context menu.

2. To change the name of the current cast, enter the new name in the Name box.

3. Choose a Preload option to define how cast members are loaded into memory when the movie runs:

- When Needed loads each cast member into memory when it is required by the movie. This setting can slow down the movie while it plays, but it makes the movie begin playing sooner. This setting is the best choice when controlling cast members loading with Lingo.
- After Frame One loads all cast members (except those required for frame 1) when the movie exits frame 1. This setting can ensure that the first frame is displayed as quickly as possible, and it may be the best choice if the first frame of the movie is designed to remain onscreen for a number of seconds.
- Before Frame One loads all cast members before the movie plays frame 1. This setting makes the movie take longer to start playing, but it provides the best playback performance if there is enough memory to hold all cast members.

Viewing and setting cast member properties

You can display and set properties for individual cast members, or for multiple cast members at once, even if the cast members are different types. In both cases, you use the Property Inspector.

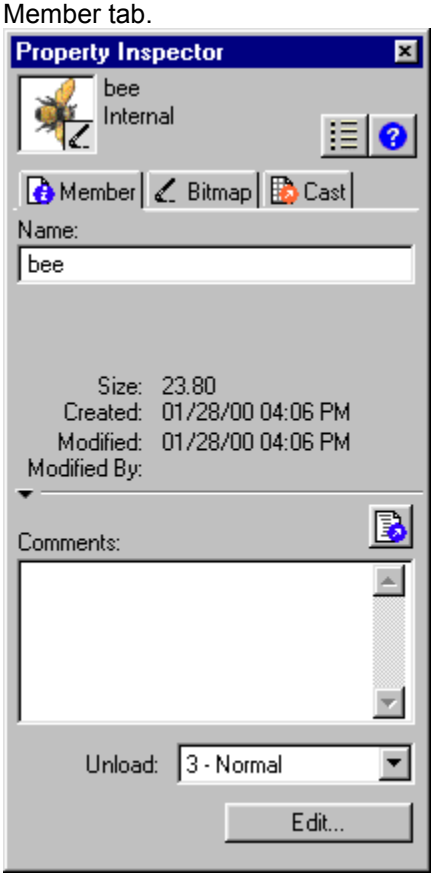
You can also set cast member properties by using Lingo (see **Setting cast member properties using Lingo**).

To view and set cast member properties:

1 Select one or more cast members.

2 Do one of the following:

- If the Property Inspector is open, click the Member tab.
- If the Property Inspector is not open, choose Window > Inspectors > Property, then click the Member tab.



As with all fields in the Property Inspector, if you've selected multiple cast members, the information common to all of the selected cast members appears. Any changes you make apply to all of the selected cast members.

3 Display the Graphical view on the Member tab.

The Member tab displays the following:

- Editable fields to view or change the cast member's name, a Comments field to enter text that appears in the Comments column of the Cast List window, and an Unload pop-up menu that lets you choose how to remove a cast member from memory. For more information on using the Unload pop-up menu, see **Controlling cast member unloading**.

- View-only fields that indicate the cast member's size, when the cast member was created and modified, and the name of the person who modified the cast member.

For an Xtra cast member, the information displayed in the Property Inspector is determined by the developer of the Xtra. Some Xtras have options in addition to those listed here. For non-Macromedia Xtras, refer to documentation supplied by the developer.

For information on specific cast member properties, see these topics:

- **Using animated GIFs**"
- **Embedding fonts in movies**
- **Using Flash Movies**
- **Setting bitmap cast member properties**
- **Setting vector shape properties**
- **Setting button cast member properties**
- **Synchronizing media**
- **Setting film loop properties**
- **Setting palette cast member properties**
- **Setting PICT cast member properties**
- **Setting shape cast member properties**
- **Setting sound cast member properties**
- **Setting text or field cast member properties**
- **Setting transition cast member properties**
- **Setting Xtra cast member properties**
- **Creating an animated color cursor cast member**
- **Streaming linked Shockwave Audio and MP3 audio files**

Launching cast member editors

You can open any cast member in the appropriate editor directly from the Cast window. You can use Director's internal media editors, such as the Text, Paint, or Vector Shape window, or you can specify external editors for certain types of cast members. See **Launching external editors**.

To launch an editor for a cast member:

Do one of the following:

- Double-click a cast member in the Cast window.
- Double-click a sprite containing the cast member in the Score or on the Stage. See **Sprites: Overview**.

Finding cast members

You can search for cast members by name, type, and color palette. You can search for selected cast members used in the Score, such as when you are preparing a movie for distribution. You can also search for cast members not used in the Score—for example, to clean up a movie and reduce the space and memory required to save and run the movie.

Before releasing a movie, it's a good idea to remove unused cast members to make the movie as small as possible for downloading.

To find cast members:

1. Choose Edit > Find > Cast Member.
2. In the Find Cast Member dialog box, choose a Cast window to search from the Cast pop-up menu.

   To search every cast in the movie, select All Casts.
3. Choose a search option:

- Select Name and enter search text in the text box. For example, to search for a group of related cast members that share a common element in their names, you might enter the word **Bird** to search for cast members named Bird 1, Bird 2, and Bird 3.
- Select Type and choose an option from the pop-up menu to search for cast members by media type.
- Select Palette and choose an option from the pop-up menu. You can use this option to search for and resolve palette conflicts.
- Select Usage to locate all cast members that aren't used in the Score. Note that cast members found with this option may be used in the movie by a Lingo script.

   Director displays the found cast member.

4. Do one of the following:

- Choose a cast member on the list and click Select to close the dialog box and select the cast member in the Cast window.
- Click Select All to close the dialog box and select all listed cast members in the Cast window.

To find a cast member in the Score:

1. Select a cast member to search for in the cast or the Score. If you select a sprite that includes multiple cast members, Director searches for the first cast member in the sprite; to select a cast member other than the first, open the sprite to select the cast member. (For information on selecting sprites, see **Selecting sprites**.)
2. Choose Edit > Find > Selection or press Control-H (Windows) or Command-H (Macintosh).

   Director searches the Score and highlights the first Score cell it finds.
3. Choose Edit > Find Again to find the next occurrence of the cast member in the Score.

Importing cast members

Importing lets you create cast members from external media. You can either import data into a Director movie file or create a link to the external file and re- import the file each time the movie opens. Linked files let you display dynamic media from the Internet, such as sports scores, sounds, and weather pictures, which makes downloading movies faster. See **About linking to files**.

Director can import cast members from almost every popular media file format. See **About import file formats**.

You can import files by using the Import dialog box, dragging files from the desktop to a Cast window, or using Lingo.

To import cast members and specify import options:

1. In Thumbnail view, select an empty position in a cast.

   If no cast position is selected, Director places the new cast member in the first available position in the current cast in Thumbnail view. In List view, Director places the new cast member at the end of the list.

2. Choose File > Import.

3. To import a file from the Internet, click Internet and enter a URL.

4. To import local files, choose the type of media to import from the Files of Type (Windows) or Show (Macintosh) pop-up menu.

   All the files in the current directory appear unless you make a selection.

5. To select a file or files to import, do one of the following:

- Double-click a file.
- Select one or more files and click Add.
- Click Add All.

   You can switch folders and import files from different folders at the same time.

6. From the Media pop-up menu at the bottom of the dialog box, choose an option to specify how to treat imported media:

- Standard Import imports all selected files, storing them inside the movie file but not updating them when changes are made to the source material. If you selected the option to import from the Internet in step 3, Director retrieves the file immediately from the Internet if a connection is available.

   Note: AVI and QuickTime files are always linked to the original external file (see the next option, Link to External File), even if you select Standard Import.

- Link to External File creates a link to the selected files and imports the data each time the movie runs. If you choose to import from a URL via the Internet, the media is dynamically updated. See **About linking to files**.

   Note: Text and RTF files are always imported and stored inside the movie file (see the previous option, Standard Import), even if you select Link to External File.

- Include Original Data for Editing preserves the original data in the movie file for use with an external editor.

   When this option is selected, Director keeps a copy of the original cast member data and sends

the original to the external editor when you edit the cast member. This option preserves all of the editor's capabilities. For example, if you specify Photoshop to edit PICT images, Director maintains all of the Photoshop object data. See **Launching external editors**.

- Import PICT File as PICT prevents PICT files from being converted to bitmaps.

7 If you selected a PICS or Scrapbook file to import, click Options to specify options for these files. See **Setting import options for PICS and Scrapbook files**.

8 When you've finished selecting the files, click Import.

If you've imported a bitmap with a color depth or color palette that differs from the current movie, the Image Options dialog box appears, in which you must enter additional information. See **Choosing import image options**.

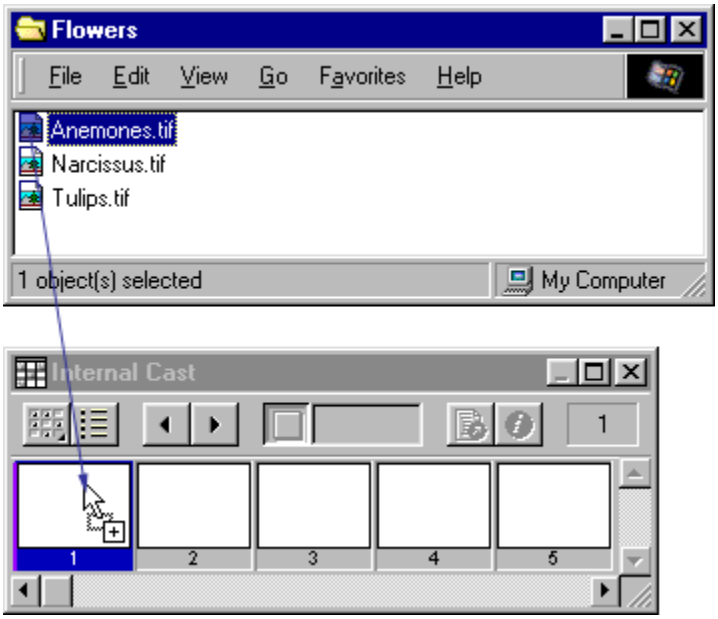
For information on importing specific media, see these sections:

- **About importing bitmaps**
- **Importing internal and linked sounds**
- **Using Director movies within Director movies**
- **Importing internal and linked sounds**
- **Text: Overview**
- **Using animated GIFs**
- **Using Flash Movies**
- **Using linked scripts**

To import files by dragging:

1 In the Explorer (Windows) or on the system desktop (Macintosh), select a file or files to import.

2 Drag the files from the desktop to the desired position in the Cast window Thumbnail view, or to the Cast window List view.



If you drag to List view, the imported files are added at the bottom of the list.

To import files with Lingo:

Use the `importFileInto` command to import a file. Set the `fileName` cast member property to assign a new file to a linked cast member. See **importFileInto** and **fileName (cast member property)**.

About import file formats

Director can import files in all the formats listed in the following table. For information on additional file formats Director may support, see the **Director Support Center Web** site.

Type of file	Supported formats
Animation and multimedia	Flash movies, animated GIF, PowerPoint presentations, Director movies, Director external cast files
Image	BMP, GIF, JPEG, LRG (xRes), Photoshop 3.0 (or later), MacPaint, PNG, TIFF, PICT, Targa
Multiple-image file	Windows only: FLC, FLI
	Macintosh only: PICS, Scrapbook
Sound	AIFF, WAV, MP3 audio, Shockwave Audio, Sun AU uncompressed and IMA compressed
	Macintosh only: System 7 sounds
Video	QuickTime 2, 3 and 4; AVI
Text	RTF, HTML, ASCII (often called Text Only), Lingo scripts
Palette	PAL, Photoshop CLUT

About linking to files

Director re-imports media every time a movie runs when Link to External File is selected in the Import dialog box (Choose File > Import). Linking makes it easier to use bulky media such as long sounds and is especially useful for showing media from the Internet that change frequently. Linking also makes downloading movies faster; users can choose whether to view linked files, so the files do not download unless they're needed.

When you link to an external file, Director creates a cast member that stores the name and location of the file. Saving a movie saves only the link to the linked cast member. Keep linked files in a folder that's close to the original movie file. Pathnames are restricted to 4096 characters by the system. URLs can be up to 260 characters. If you store a file too many folders away from the movie or using a very long URL, it may fail to link correctly.

When distributing movies with linked media, follow these guidelines:

- If you distribute a movie, you also must include all linked cast member files, and they must be in their expected locations. In addition, the Xtras used to import the media must be present when the movie runs (either on the user's computer or included in your movie). For more information, see **Setting Xtra cast member properties**.
- When you link to media on the Internet, the media must be present at the specified URL when the movie runs. Provide for link failure, because you can't guarantee that an Internet transaction will be successful.
- To retrieve media from the Internet during playback, Director requires that the projector include certain Xtras. To include these Xtras automatically, click Add Network in the Movie Xtras dialog box. (Movies playing in Web browsers do not require these Xtras.)

Note: Use File > Preferences > Network to define standard network settings for the Director authoring environment; see the Network Preferences topic in Director Help.

Choosing import image options

If you import a bitmap cast member with a color depth or color palette different from that of the Stage (the current movie), Director lets you choose the image's color depth and color palette. You can choose to import the bitmap at its original color depth or at the Stage color depth. (The Stage color depth is the same as the system color depth.) You can also choose to import the image's color palette or remap the image's colors to a palette in the movie.

In many cases, it's easiest to change the image's color depth to the depth of the movie and remap the image to the color palette used in the rest of the movie. For more on controlling color in Director, see **Color, tempo, and transitions: Overview**.

If you change 16-, 24-, or 32-bit cast members to 8 or fewer bits, you must remap the cast members to an existing color palette.

To choose bitmap image options for importing:

1. Import a bitmap image by choosing File > Import. (For more information on this procedure, see **Importing cast members**.)
2. If the Image Options dialog box appears while you are importing a bitmap image using File > Import, choose a Color Depth option:

- Image specifies the color depth and palette of the image.
- Stage specifies the color depth of the current Stage.

3. Choose a Palette option to change palette settings for 2-, 4- or 8-bit images:

- Import imports the image with its color palette. The palette appears as a new cast member immediately following the bitmap cast member.
- Remap To replaces the image's colors with the most similar solid colors in the palette you select from the pop-up menu.

4. Choose Image options:

- Trim White Space removes any white pixels from the edges of the image. Deselect this option to preserve the white canvas around an image.
- Dither blends the colors in the new palette in the Palette section to approximate the original colors in the graphic.

5. To apply the current settings to all the remaining files selected for importing, select Same Settings for Remaining Images.

Setting import options for PICS and Scrapbook files

You can import PICS and Scrapbook files several different ways. These file formats are available only on the Macintosh.

To set import options for PICS and Scrapbook files:

1. Import the PICS or Scrapbook cast member by choosing File > Import. (For more information on this procedure, see **Importing cast members**.)

2. If the PICS/Scrapbook Options dialog box appears while you are importing an image using File > Import, specify the range of images to import:

- All Frames imports up to 512 frames in a PICS file or from a Scrapbook file. Each frame will be imported as a separate cast member.
- From/To selects a range of cells. The imported PICS or Scrapbook frames are added to the Score beginning at the selected cell; any existing Score data will be replaced by the imported frames.

   If the PICS file was created using a previous version of Director, you must enter 1 as the starting frame to import.

3. To import only the image, not the surrounding white space, select Contract White Space.

4. To place the imported artwork in the Paint window in its original position relative to the other artwork in the series, select Original Position.

5. To center each piece of imported artwork relative to the rest of the artwork in the series, click Centered.

Launching external editors

You can specify external applications to edit many types of media. All the types of media for which you can define an external editor are listed in the Editors Preferences dialog box. Once you set up an external editor for a particular media type, Director launches the application when you edit a cast member of that type. When you finish editing a cast member in an external editor and then save and close the file, Director re-imports the cast member media.

If you want to use an external editor for a cast member you import, select Include Original Data for External Editing when you import the cast member. (For more information, see **Importing cast members**.)

It is not possible to define an external editor for any cast member created by an Xtra. These include text, vector shapes, Flash movies, and custom cursors.

To define an external editor:

1. Choose File > Preferences > Editors.
2. Choose a type of media for which you want to define an external editor.
3. Click Edit.
4. Click Browse or Scan to locate the application.

   You can specify any application capable of editing the selected type of media.
5. To determine which editor appears when you double-click a cast member, choose Use Internal Editor or Use External Editor.

- If you usually want to make changes inside of Director and only occasionally want to use the external editor, choose Use Internal Editor.
- If you usually want to use the external editor to make changes to the cast member, choose Use External Editor.

To launch an external editor:

1. Select a cast member of a media type for which you have defined an external editor, then do one of the following:

- If you specified Use External Editor when you defined the external editor for this media type, double-click the cast member.
- Choose Edit > Launch External Editor.
- While the cast member is selected and the Cast window is active, right- click (Windows) or Control-click (Macintosh) and choose Launch External Editor from the context menu.

   Director launches or switches to the application used to create the cast member, sending the original data to the external editor.

Note: If you've specified an external editor and you want to edit a cast member with Director's internal editors, select the cast member and choose Edit > Edit Cast Member.

2. Edit the cast member.

   Note that if you change an image in the Paint window and then edit the image with an external editor, changes made in the Paint window, with the exception of registration points, are lost. Director warns you if this is a possibility.

3 Save and close the file. Director re-imports the cast member.

Controlling cast member unloading

When Director runs low on memory, it removes cast members from memory. You use the Property Inspector to specify the priority with which a cast member is removed from memory. When a cast member is available in memory, it appears almost instantly. When it needs to be loaded from disk, the loading can cause a delay. Set your cast members so that frequently used cast members remain in memory as long as possible.

These settings are the same for all types of cast members.

To specify the Unload setting:

1. Select the cast members in the Cast window.
2. In the Member tab of the Property Inspector, display the Graphical view and then choose an option from the Unload pop-up menu:

- 3—Normal sets the selected cast members to be removed from memory after any priority 2 cast members have been removed.
- 2—Next sets the selected cast members to be among the first removed from memory.
- 1—Last sets the selected cast members to be the last removed from memory.
- 0—Never sets the selected cast members to be retained in memory; these cast members are never unloaded.

Managing external casts

An external cast is a separate file that must be explicitly linked to a movie for the movie to use its cast members.

If you link an external cast to a movie, Director opens the cast every time it opens the movie. If you don't link an external cast to a movie, you must open and save the file separately. You can use unlinked external casts as libraries to store commonly used elements for authoring, such as scripts, buttons, and so on; see **Creating libraries**.

When you distribute a movie that uses an external cast, you must include the external cast file. For disk-based movies, the cast must be in the same relative path in your files as it was when the movie was created. For Shockwave movies on the Web, the cast must be at the specified URL.

To create an external cast:

1. Choose File > New > Cast.
2. Type a name for the new cast.
3. Specify that the cast be stored as an external cast.

   If you don't want to use the cast in the current movie, deselect the Use in Current Movie option.
4. Click Create.

   The cast is created, and a Cast window for the cast is opened in List view. See **Using the Cast window**.
5. Choose File > Save while the Cast window is active, then save the cast in the desired directory.

To link an external cast to a movie:

1. Choose Modify > Movie > Casts.
2. In the Movie Casts dialog box, click Link.
3. Locate and select the external cast you want and then click Open.

   You can link to casts on your local disk or to casts stored at any URL. Click Internet to enter a URL for a linked external cast.
4. Click OK.

To unlink a cast from a movie:

1. Choose Modify > Movie > Casts.
2. In the Movie Casts dialog box, select the external cast.
3. Click Remove.

To save a movie and all open casts, linked or unlinked:

Choose File > Save All.

Note: To use a cast member from an external cast without creating a link to the external cast, first copy the cast member to an internal cast or to a (different) linked external cast.

Creating libraries

A library is a special type of unlinked external cast. When you drag a cast member from an external cast library to the Stage or Score, Director automatically copies the cast member to one of the movie's internal casts. Libraries are useful for storing any type of commonly used cast members, especially behaviors. A library cannot be linked to a movie. See **Attaching behaviors**.

When you create a library as explained in the following procedure, it appears on the Library pop-up menu in the Library palette.

To create a library:

1. Create an external cast file, following the procedure under **Creating new casts**. Do not select Use in Current Movie.
2. With the Cast window for the external cast active, choose File > Save and place the external cast in the Libs folder in the Director application folder.

Setting cast member properties using Lingo

Lingo lets you control and edit cast members by setting their properties. Some properties are available for every type of cast member. Other properties are available only for specific cast member types.

To specify the cast member's content:

Set the `media` cast member property. See **media**.

To specify the cast member's name:

Set the name cast member property. See **name (cast member property)**.

To set the contents of the cast member's comments field:

Set the `comments` cast member property. You can store any text information in this field that you find useful and access it at runtime by getting the `comments` property. See **comments**.

To specify the cast member's purge priority:

Set the `purgePriority` cast member property. See **purgePriority**.

To specify the content of the script, if any, attached to the cast member:

Set the `scriptText` cast member property. See **scriptText**.

To specify the file assigned to a linked cast member:

Set the `fileName` cast member property. See **fileName (cast member property)**.

For additional cast member properties that Lingo can test and set, refer to the *Lingo Dictionary* sections that discuss the specific cast member type.

Setting Xtra cast member properties

Xtra cast members have the same Name and Unload properties as other cast members, but they may also contain an extra panel of options accessible from the Property Inspector. To set cast member properties, use the Member tab and the custom tab for the type of cast member you are working with. The Member tab contains an Edit button and may contain a More Options button. Use the Edit button to edit the cast member with its default editor. Use the More Options button to display the Cast Member Properties dialog box for the current cast member.

The custom tab for the type of cast member you are working with may also contain a More Options button. This button will display the Cast Member Properties dialog box for the current cast member.

The content of the Properties dialog box is determined by the developer of the Xtra. For non-Macromedia Xtras, refer to any documentation supplied by the developer.

To view or change Xtra cast member properties:

1. Select an Xtra cast member.
2. Open the Property Inspector and click the Member tab.

   The Member tab displays information about the member:

- The cast member name
- The name of the cast that contains the cast member
- The size in kilobytes
- The creation date
- The date the cast member was last modified
- The name of the user who last modified the cast member

3. Use the Name field to view or edit the cast member name.
4. To specify how Director removes cast members from memory if memory is low, choose options from the Unload pop-up menu. See **Controlling cast member unloading**.
5. To set special options for the current Xtra cast member, click the custom tab for the cast member you are working with. Some types of Xtra cast members will also have a More Options button on this tab. Use it to set any properties of the cast member that are not displayed on the tab itself.

Sprites: Overview

A sprite is an object that controls when, where, and how cast members appear in a movie. Multiple sprites can use the same cast member. You can also switch the cast member assigned to a sprite as the movie plays. You use the Stage to control *where* a sprite appears and you use the Score to control *when* it appears in your movie.

Sprites display on the Stage layered according to the channel in which they're in the Score. Sprites in higher-numbered channels appear in front of sprites in lower-numbered channels. A movie can include up to 1000 sprite channels. Use the Movie tab of the Property Inspector to control the number of channels. See **Setting Stage and movie properties**.

Sprite properties include the sprite's size and location, the cast member assigned to the sprite, and the frames in which the sprite occurs. Different properties can alter the appearance of a sprite. You can rotate, skew, flip, and change the color of sprites without affecting cast members. You can change sprite properties with the Property Inspector or Lingo.

In Lingo, some properties are available only for certain types of sprites. Such properties typically are characteristics related to the specific sprite type. For example, Lingo has several digital video properties that determine the contents of tracks in digital video sprites.

To control the way a sprite's colors appear on the Stage, you use sprite inks. By selecting Background Transparent ink in the Property Inspector, for example, you turn all white pixels transparent and remove the white border (the bounding rectangle) around bitmap images (assuming the sprite is over a white background). Other inks provide more complex and interesting effects such as reversed colors or colors that change in different ways depending on the background color.

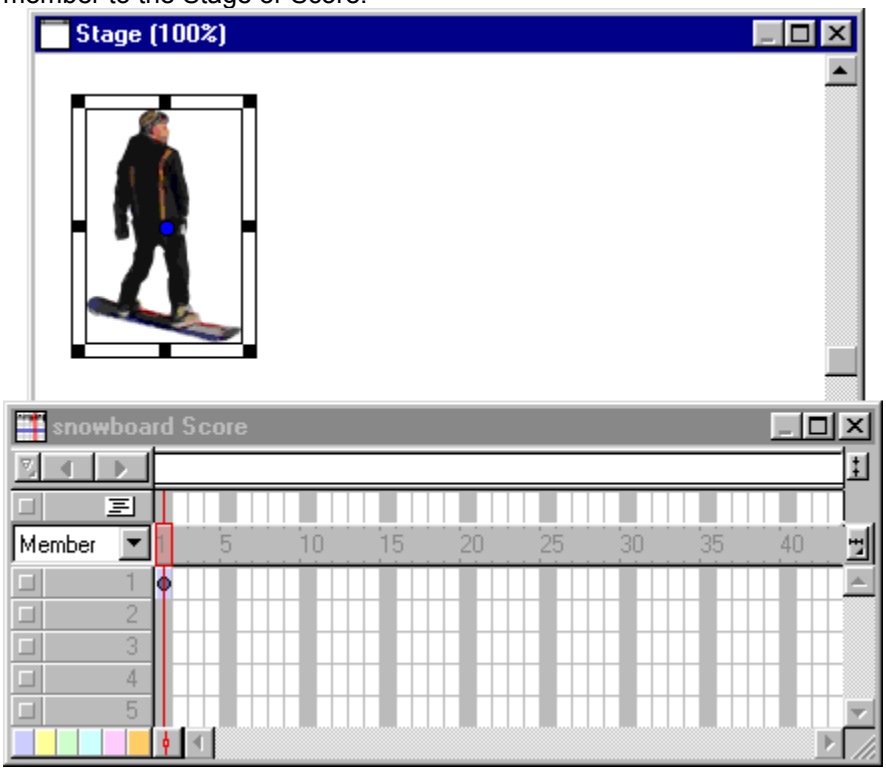
Creating sprites

You create a sprite by dragging a cast member to either the Stage or the Score; the sprite appears in both places. New sprites, by default, span 28 frames. To change the default sprite duration, choose File > Preferences > Sprite; see **Changing sprite preferences**.

To create a new sprite:

1. Click to select the frame in the Score where you want the sprite to begin.
2. From the Cast window, in either List or Thumbnail view, do one of the following:

- Drag a cast member to the position on the Stage where you wish to place the sprite.
- Drag a cast member to the Score. Director places the new sprite in the center of the Stage.
- To create a sprite one frame long, press Alt (Windows) or Option (Macintosh) and drag a cast member to the Stage or Score.



Changing sprite preferences

You use the Sprite Preferences dialog box to control the way sprites behave and appear in the Score window and on the Stage.

To change preferences for sprites:

1. Choose File > Preferences > Sprite.
2. To determine if selecting a sprite on the Stage selects the entire span of the sprite or only the current frame in the sprite, choose a Stage Selection option:

- Entire Sprite selects the sprite in all frames that it occupies.
- Current Frame Only selects only the current frame of the sprite.

3. To determine the appearance and behavior of sprites yet to be created, choose Span Defaults options. These options do not change settings for existing sprites.

- Display Sprite Frames turns on Edit Sprite Frames for all new sprites. See **Editing sprite frames**.
- Tweening turns on tweening for all tweenable properties. This option is on by default. With this option off, sprites must be manually tweened when new frames or keyframes are added to the sprite. For additional information on tweening, see **Animation: Overview**.

4. To determine the length of sprites measured in frames, choose Span Duration options.

- Frames defines the default number of frames for sprites.
- Width of Score Window sets the sprite span to the visible width of the Score window.
- Terminate at Markers makes new sprites end at the first marker. See **Using markers**.

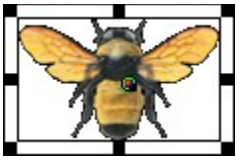
Selecting sprites

To edit or move a sprite, you must select it. You can select sprites, frames within sprites, and groups of sprites in several different ways.

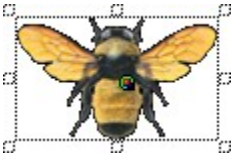
You use the Arrow tool on the Tool palette to select sprites prior to most operations. You can also select sprites with the Rotate and Skew tool to enable rotation and skewing. See **Rotating and skewing sprites**.

When selecting sprites, you often want to select a certain frame or range of frames within the sprite instead of the entire sprite. When you make certain changes to a frame within a sprite, it becomes a selectable object called a keyframe. See **Editing sprite frames**.

A selected sprite appears on the Stage with a double border. When you select a single frame within a sprite, the sprite appears on the Stage with a single border.



Entire sprite selected



Single frame within sprite selected

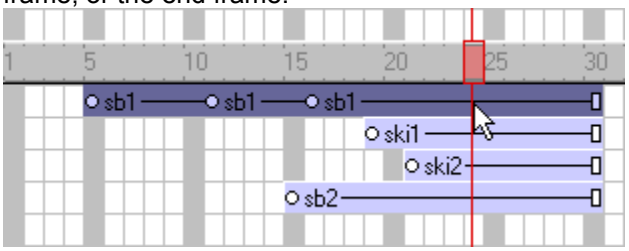
To select sprites, do one of the following:

Note: The following techniques select an entire sprite only if Edit Sprite Frames is not enabled for the sprite(s) you select.

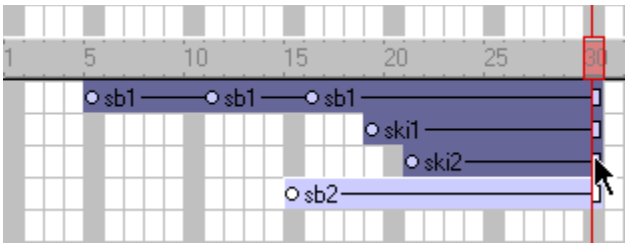
- On the Stage, click a sprite to select the entire sprite span.

  You can change sprite preferences so that selecting a sprite on the Stage selects only the current frame instead of the entire sprite. See **Changing sprite preferences**.

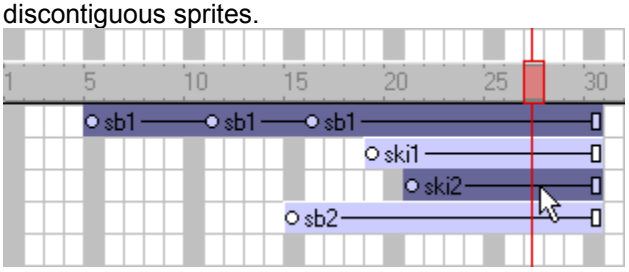
- In the Score, click the horizontal line within a sprite bar; do not click the keyframes, the start frame, or the end frame.



- To select a contiguous range of sprites either on the Stage or in the Score, select a sprite at one end of the range and then Shift-click a sprite at the other end of the range. You can also drag to select all the sprites in an area.

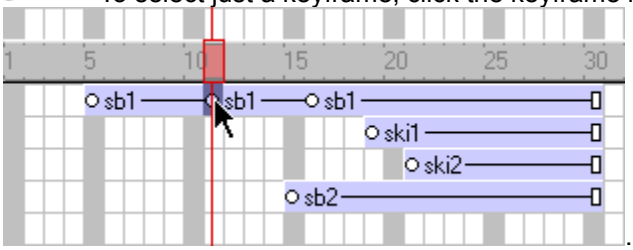


- To select discontiguous sprites, Control-click (Windows) or Command- click (Macintosh) the discontiguous sprites.



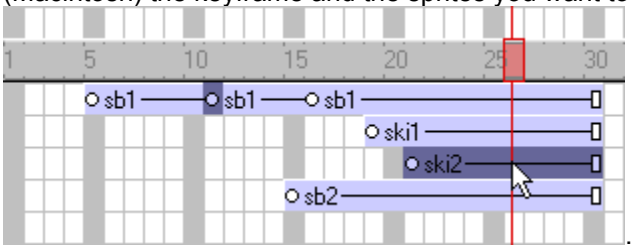
To select a keyframe, do one of the following:

- To select just a keyframe, click the keyframe indicator



.

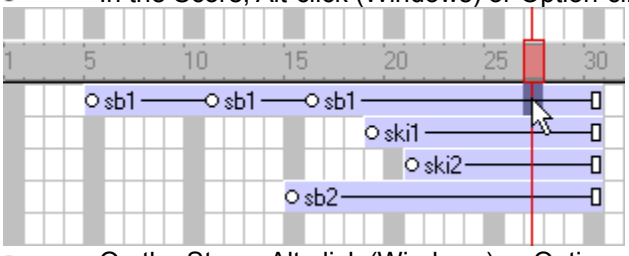
- To select a keyframe and sprites at the same time, Control-click (Windows) or Command-click (Macintosh) the keyframe and the sprites you want to select



.

To select a frame within a sprite that isn't a keyframe, do one of the following:

- In the Score, Alt-click (Windows) or Option-click (Macintosh) the frame within the sprite.



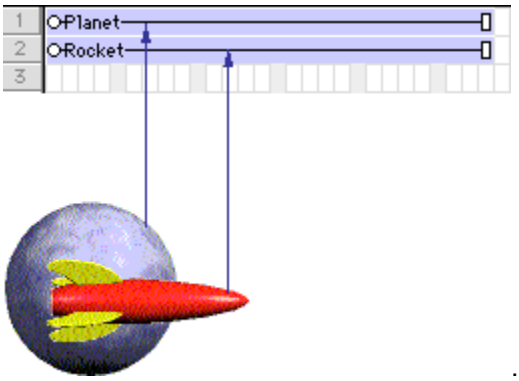
- On the Stage, Alt-click (Windows) or Option-click (Macintosh) to select only the current frame of the sprite. The sprite appears on the Stage with a single border.

To select all the sprites in a channel:

Click the channel number at the left side of the Score.

Layering sprites

A sprite appears in front of other sprites on the Stage according to its channel. Sprites in higher-numbered channels appear in front of sprites in lower-numbered channels



The rocket in channel 2 appears in front of the planet in channel 1.

To change a sprite's layer on the Stage:

1. Select the sprite. To select the contents of an entire channel, click the channel number at the left side of the Score.
2. Do one of the following:

- Choose Modify > Arrange and select a command from the submenu to change the order of sprites.
- Drag the sprite in the Score from one channel to another.
- If you selected a channel, drag its contents to another channel.

Displaying and editing sprite properties in the Property Inspector

Depending on your preference, you can use either the Sprite toolbar or the Property Inspector to perform many of the same procedures.

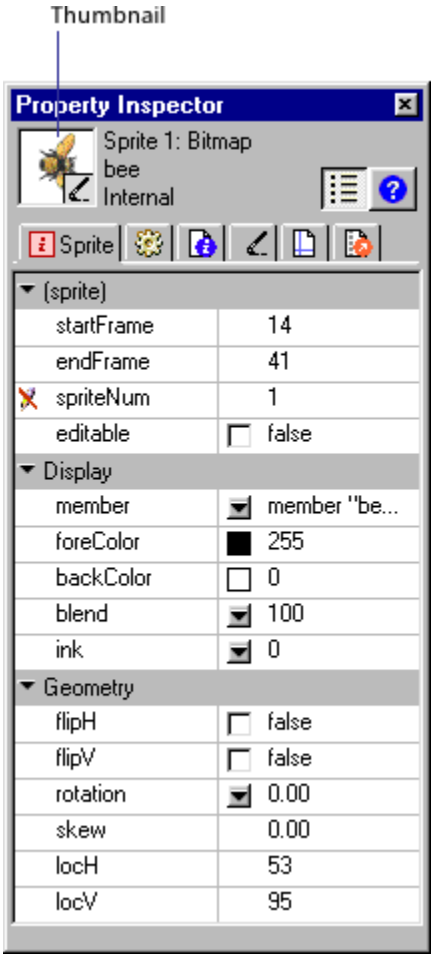
To display and edit sprite properties in the Property Inspector:

1 Select one or more sprites on either the Stage or the Score.

2 If the Property Inspector is not open, choose Window > Inspectors > Property.

The Property Inspector opens with focus on the Sprite tab. The Graphical view is the default view. You can toggle to the List view by clicking the List View Mode icon.

The Property Inspector displays settings for the current sprite. If you select more than one sprite, the Property Inspector displays only their common settings.



A thumbnail image of the sprite's cast member appears in the upper left corner of the Property Inspector.

Note: To open a window in which you can edit the sprite's cast member, you can double-click the thumbnail image.

3 Edit any of the following sprite settings in the Property Inspector:

- Start Frame and End Frame display the start and end frame numbers of the sprite. Enter new

values to adjust how long the sprite plays. See **Changing the duration of a sprite on the Stage**.

- Lock changes the sprite to a locked sprite so you or other users cannot change it. For additional information on locked sprites, see **Locking and unlocking sprites**.

- Editable applies only to text sprites, and lets you edit the selected text sprite on the Stage during playback. See **Selecting and editing text on the Stage**.

- Moveable lets you position the selected sprite on the Stage during playback. See **Visually positioning sprites on the Stage**.

- Trails makes the selected sprite remain on the Stage, leaving a trail of images along its path as the movie plays. If Trails is not selected, the selected sprite is erased from previous frames as the movie plays.

- Flip Horizontal and Flip Vertical reverse the sprite horizontally or vertically to form an inverted image. See **Flipping sprites**.

- The Ink pop-up menu displays the ink of the current sprite and lets you choose a new ink color. See **Using sprite inks**.
- Blend determines the blend percentage of the selected sprites. See **Setting blends**.
- Foreground and Background color boxes determine the colors of the selected sprite. See **Changing the color of a sprite**.
- Reg Point Horizontal (X) and Vertical (Y) display the location of the registration point in pixels from the top left corner of the Stage. See **Editing sprite properties with Lingo**.
- Width (W) and Height (H) show the size of the sprite's bounding rectangle in pixels.
- Rotation Angle rotates the sprite by the number of degrees you enter. See **Rotating and skewing sprites**.
- Skew Angle slants the sprite by the number of degrees you enter. See **Rotating and skewing sprites**.
- Left, Top, Right, and Bottom show the location of the edges of the sprite's bounding rectangle.
- Restore All reverts the height and width to that of the cast member.
- Scale opens the Scale Sprite dialog box, where you can resize the selected sprite. See **Resizing and scaling sprites**.

Displaying sprite properties in the Sprite toolbar

The Sprite toolbar displays a subset of the same information and fields found on the Sprite tab in the Property Inspector. You can use either the Sprite toolbar or the Property Inspector, depending on your preference, to perform many of the same procedures.

To show or hide the Sprite toolbar in the Score:

While the Score is active, choose View > Sprite Toolbar.

Using the Sprite Overlay

The Sprite Overlay displays important sprite properties directly on the Stage. You can open editors, inspectors, and dialog boxes to change sprite properties by clicking the corresponding icons in the Sprite Overlay.

To display the Sprite Overlay when a sprite is selected:

Choose View > Sprite Overlay > Show Info.

- [[Can we move the callout to Sprite Overlay panel up a wee bit so it doesn't look like it's pointing to the opacity control?]]

To use Sprite Overlay options to change how the overlay appears:

1. Click the Sprite on the Stage to select it.
2. In the Sprite Overlay panel, click the icon that represents the data you want to edit:

- To edit the Sprite's cast member, click this icon to open the tab in the Property Inspector that applies to this type of sprite. For example, clicking this icon displays the Vector tab for a vector sprite, the Text tab for a text sprite, and so on.

- To open the Sprite tab of the Property Inspector, click this icon.

- To open the Behavior tab of the Property Inspector, click this icon. See **Behaviors: Overview**.

To change the Sprite Overlay's appearance to suit your preferences:

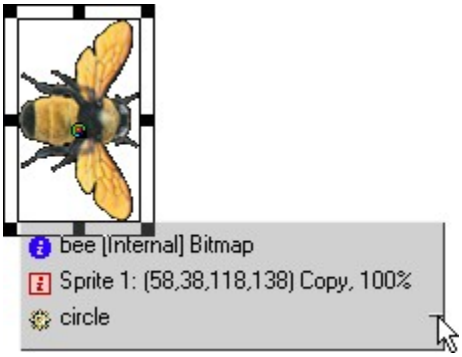
1. Choose View > Sprite Overlay > Settings.
2. Choose a Display option to determine when sprite properties are visible and active:

- Roll Over displays sprite properties only when the pointer is over a single sprite.
- Selection displays sprite properties when a sprite is selected.
- All Sprites displays sprite properties for all sprites on the Stage.

3. Use the Text Color box to select the color for text displayed in the Sprite Overlay.

To change the opacity of the Sprite Overlay panel:

Drag up or down the small thin line that appears on the right edge of the Sprite Overlay panel.



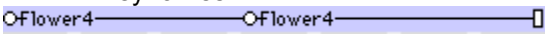
Displaying sprite labels in the Score

Sprite labels appear in the Score's sprite bars and display key information about the sprite in relation to the movie. For example, if you detect a strange blip caused by an ink effect, you can turn on Ink display and quickly locate the problem in a sprite label. You can change the information that appears in labels; for example, you can use the Extended display option to display the precise location of a sprite in every frame.

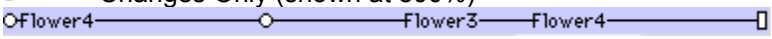
To display sprite labels:

1. With the Score as the active window, choose View > Sprite Labels.
2. Choose from the following options:

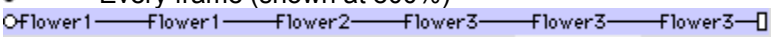
- Keyframes



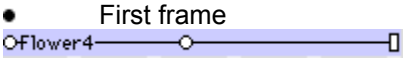
- Changes Only (shown at 800%)



- Every frame (shown at 800%)



- First frame

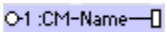


Note that many options are useful only when the Score is zoomed to 400% or 800%.

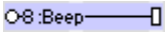
To change sprite label options:

Choose a display option from the Display pop-up menu in the Score or from the View > Display menu.

- Member displays the name and number of the sprite's cast member.

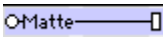


- Behavior displays the behavior assigned to the sprite.



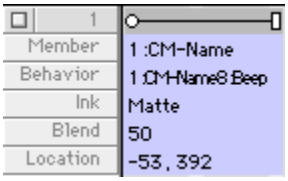
- Location displays the *x* and *y* coordinates of the sprite's registration point.

- Ink displays the ink effect applied to each sprite.



- Blend displays the blend percentage.

- Extended displays any combination of display options; choose options by choosing File > Preferences > Score.



Editing sprite properties with Lingo

You can use Lingo to check and edit sprite properties with scripts as the movie plays.

To check a property value:

Use the `put` command or check in the Watcher window. See **put**.

To edit a property:

Use the equals (=) operator or the `set` command to assign a new value to the property. See **= (equals)** and **set...to, set...=**.

Locking and unlocking sprites

During authoring, you can lock sprites to avoid inadvertent changes to the sprite, either by you or by someone else working on the same project. When you lock a sprite, you can no longer change its settings, although you still see it represented on the Stage and in the Score. While preserving the settings of your locked sprites, you can continue to create and edit unlocked sprites.

Locking sprites is not supported during playback.

Note: If you try to perform an operation on a group of locked and unlocked sprites, a message appears that indicates the operation will affect only the unlocked sprites.

To lock a sprite:

In the Stage or the Score, select one or more sprites to lock and do one of the following:

- Choose Modify > Lock Sprite.
- In the Sprite tab of the Property Inspector, click the padlock icon.
- Right-click (Windows) or Option-click (Macintosh) and choose Lock Sprite from the context menu.

In the Score, a locked sprite appears with a padlock in front of its name. On the Stage, a locked sprite appears with a padlock in its upper right corner.

To select a locked sprite on the Stage:

Hold down the L key while selecting the sprite.

To unlock a sprite:

In the Score or on the Stage, select one or more sprites to unlock and do one of the following:

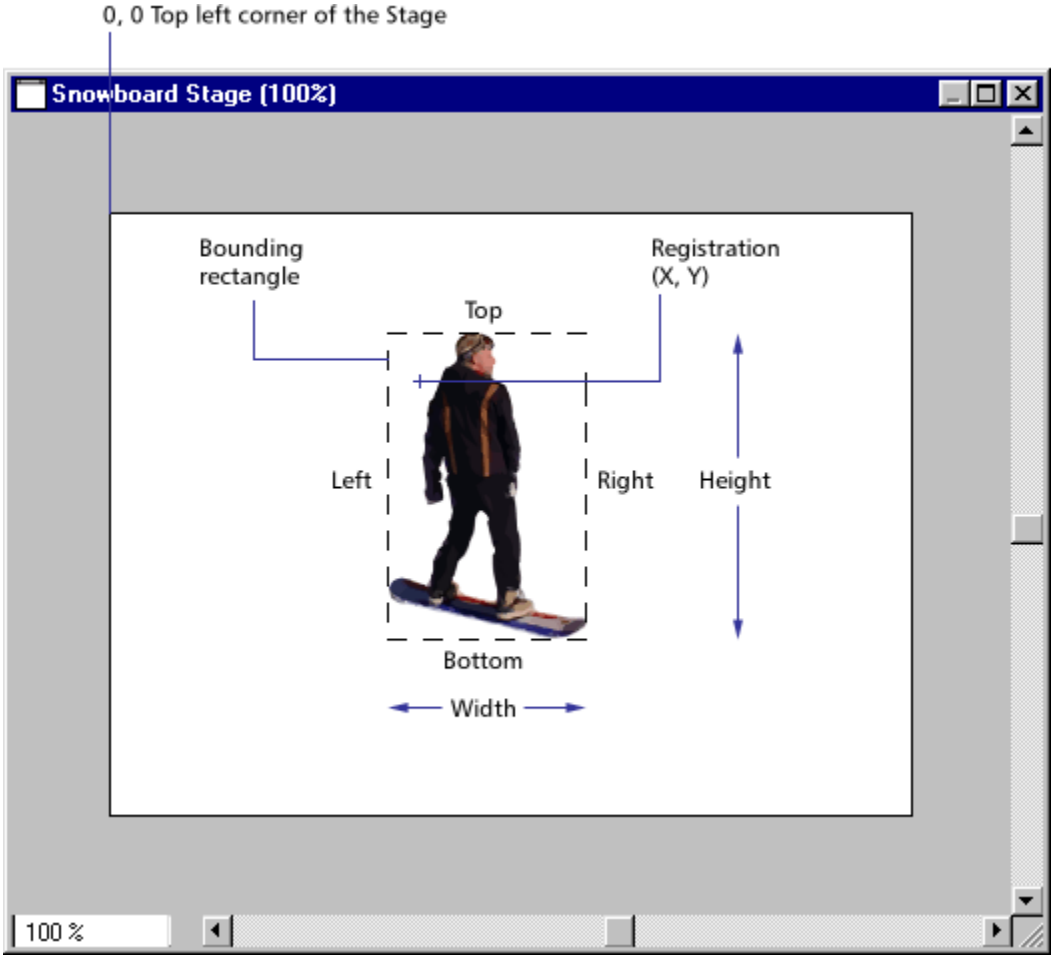
- Choose Modify > Unlock Sprite.
- In the Sprite tab of the Property Inspector, click the padlock icon.
- Right-click (Windows) or Option-click (Macintosh) and choose Unlock Sprite from the context menu.

Positioning sprites

The easiest way to position a sprite is to simply drag the sprite into place on the Stage. To position a sprite more precisely, you can do any of the following:

- Set a sprite's position on the Stage by entering coordinates in the Property Inspector.
- Use the Tweak window.
- Use guides or the grid.
- Use the Align window.
- Set the sprite's coordinates in Lingo.

The following diagram shows all of the sprite coordinates you can specify.



Director places the image of a cast member on the Stage by specifying the location of its registration point. For many cast members, such as bitmap or vector shapes, the registration point is in the center of the bounding rectangle by default. For other types of cast members, the registration point is at the upper left corner. (For instructions on changing the location of the registration point of bitmap cast members, see **Changing registration points**. For instructions on changing a vector shape cast member's registration point, see **Editing vector shapes**.)

Visually positioning sprites on the Stage

You can position sprites on the Stage by dragging them or by using the arrow keys.

To visually position a sprite on the Stage:

1. Choose Window > Stage to display the Stage.
2. Do one of the following on the Stage:

- Drag a sprite to a new position. Hold down Shift to constrain the movement to horizontal or vertical straight lines.
- Select a sprite and use the arrow keys to move the selected sprite 1 pixel at a time. Hold down Shift as you press an arrow key to move the selection 10 pixels at a time.

To visually position a sprite on the Stage during playback:

1. Select a sprite that you want to position during playback.
2. In the Sprite tab of the Property Inspector, click Moveable. See **Displaying and editing sprite properties in the Property Inspector**.
3. Begin playing back the movie.
4. On the Stage, drag the sprite to the new position.

Positioning sprites with the Property Inspector

You can use the Property Inspector to specify the exact coordinates of a sprite.

To set sprite coordinates in the Property Inspector:

1. With the Property Inspector open and in Graphical view, select a sprite to reposition.
2. On the Sprite tab of the Property Inspector, specify the sprite coordinates in pixels, with 0,0 at the upper left corner of the Stage, as follows:

- Specify attributes in the X and Y fields to change the horizontal and vertical coordinates of the registration point.
- Specify coordinates in the W and H fields to change the width and height of the sprite.
- Specify values for l, r, t, and b to change the left, right, top, and bottom edges of the sprite's bounding rectangle.

To move the sprite without resizing it, adjust only the *x* and *y* coordinates.

Positioning sprites with the Tweak window

You can use the Tweak window when you want to move sprites by a certain number of pixels.

To position sprites with the Tweak window:

1. Choose Modify > Tweak.
2. Select the sprite or sprites you want to move, as described in **Selecting sprites**.
3. In the Tweak window, drag the point on the left side of the window or enter the number of pixels in the fields for horizontal and vertical change; then click Tweak.
4. If you want to repeat the move, click Tweak again.

Positioning sprites using guides, the grid, or the Align window

On the Stage, you can align sprites using guides, the grid, or the Align window.

The grid consists of cell rows and columns of a specified height and width that you use to assist you in visually placing sprites on the Stage. The grid is always available.

Guides are horizontal or vertical lines you can either drag around the Stage or lock in place to assist you with sprite placement. You must create guides before they become available.

Moving a sprite with the Snap To Grid or Snap To Guides feature selected lets you snap the sprite's edges and registration point to the nearest grid or guide line. When you're not using the guides or the grid, you can hide them.

Guides and the grid are visible only during authoring.

You can create and modify the guides and the grid from the Property Inspector, or by using menu commands.

To add and configure guides:

1. With the Property Inspector open, click the Guides and Grid tab.

   The top half of the tab contains settings for Guides.

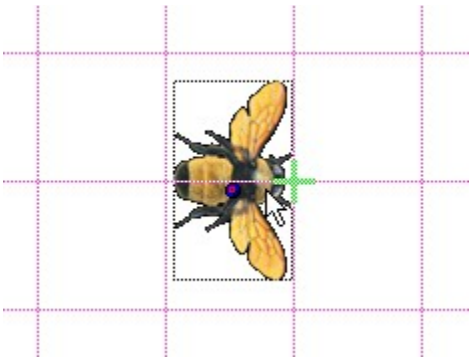
2. To change the guide color, click the Guide Color box and select a different color.
3. Select the desired options to make the guides visible, to lock them, and to make the sprites snap to the guides.
4. To add a guide, move the cursor over the new horizontal or vertical guide, then drag the guide to the Stage. Numbers in the guide tooltip indicate the distance, in pixels, the guide is from the top or left edge of the Stage.
5. To reposition a guide, move the pointer over the guide. When the sizing handle appears, drag the guide to its new position.
6. To remove a guide, drag it off the Stage.
7. To remove all guides, click Remove All on the Guides and Grid tab of the Property Inspector.

To display guides and align sprites:

1. If guides are not displayed on the Stage, choose View > Guides and Grid > Show Guides.
2. If Snap To Guides is not selected, choose View > Guides and Grid > Snap To Guides.
3. Move a sprite on the Stage near a guide line to make the sprite snap to that exact location.

To display a grid and align sprites:

1. If grid lines are not displayed on the Stage, choose View > Guides and Grid > Show Grid.



2 If Snap To Grid is not selected, choose View > Guides and Grid > Snap To Grid.

3 Move a sprite on the Stage near a grid line to make the sprite snap to that exact location.

Note: Press G while moving or resizing a sprite to temporarily turn Snap To Grid off or on.

To configure the grid:

1 With the Property Inspector open, click the Guides and Grid tab.

The bottom half of the tab contains Grid settings.

2 To change the grid color, click the Grid Color box and select a different color.

3 Select the desired options to make the grid visible and to make the sprites snap to the grid.

4 To change the size of the grid, enter numbers for Width and Height in the Spacing text box.

To align sprites using the Align window:

1 On the Stage or in the Score, select the sprites to align.

Select entire sprites, keyframes, or frames within sprites in as many different frames or channels as you need. All of the elements will align to the last sprite or frame selected.

2 Choose Modify > Align and do one of the following:

- Click an area in the preview window to view and select the way in which the sprites will align.
- Choose a vertical and/or horizontal alignment option from the pop-up menus.

3 Click Align.

4 Close the Align window when you finish aligning selections.

Positioning sprites with Lingo

Lingo lets you control a sprite's position by setting the sprite's coordinates on the Stage. You can also test a sprite's coordinates to determine a sprite's current position and whether two sprites overlap.

To check the location of a sprite's registration point or bounding rectangle on the Stage:

Test the `bottom`, `left`, `loc`, `locH`, `locV`, `right`, or `top` sprite property.

The `bottom`, `left`, `right`, and `top` sprite properties determine the location of the sprite's individual edges. See **bottom**, **left**, **right**, and **top**.

To place a sprite at a specific location:

Set one of the following properties:

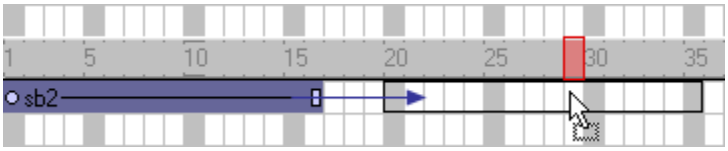
- The `loc` sprite property sets the horizontal and vertical distance from the upper left corner of the Stage to the sprite's registration point. The value is given as a point. See **loc**.
- The `locV` sprite property sets the number of pixels from the top of the Stage to a sprite's registration point. See **locV**.
- The `locH` sprite property sets the number of pixels from the left of the Stage to a sprite's registration point. See **locH**.
- The `rect` sprite property sets the location of the sprite's bounding rectangle on the Stage. See **rect (sprite)**.
- The `quad` sprite property sets the location of the sprite's bounding rectangle on the Stage. You can specify any four points; the points do not have to form a rectangle. The `quad` property can set the sprite's coordinates as precise floating-point numbers. See **quad**.

To determine whether two sprites overlap:

Use the `sprite...intersects` operator to determine whether a sprite's bounding rectangle touches the bounding rectangle of a second sprite. Use the `sprite...within` operator to determine whether a sprite is entirely within a second sprite. See **sprite...intersects** and **sprite...within**.

Changing when a sprite appears on the Stage

A sprite controls not only where media appears on the Stage, but also when. You change when a sprite appears on the Stage by moving the sprite to different frames in the Score and by changing the number of frames the sprite spans. You can either drag sprites to new frames or copy and paste them. Copying and pasting is easier when moving sprites more than one screen-width in the Score. You can also copy and paste to move sprites from one movie to another.



Moving a sprite in the Score

To change when a sprite appears on the Stage:

1. Choose Window > Score to display the Score.
2. Select a sprite or sprites, as described in **Selecting sprites**.
3. Drag the sprite to a different frame.

   To move a sprite without spreading it over additional frames, hold down the Spacebar and drag. This technique is also useful for moving any sprite that consists mostly (or entirely) of keyframes.

To copy or move sprites between frames:

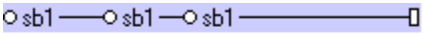
1. Select a sprite or sprites, as described in **Selecting sprites**.
2. Choose Edit > Cut Sprites or Edit > Copy Sprites.
3. Position the pointer where you want to paste the sprite and choose Edit > Paste Sprites.

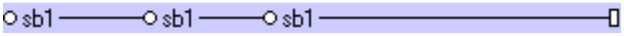
   If the pasting will overwrite existing sprites, choose a Paste option in the Paste Options dialog box:

- Overwrite Existing Sprites replaces the sprites with the content of the Clipboard.
- Truncate Sprites Being Pasted pastes the Clipboard contents in the space available without replacing existing sprites.
- Insert Blank Frames to Make Room adds new frames for the contents of the Clipboard.

Changing the duration of a sprite on the Stage

By default, Director assigns each new sprite a duration of 28 frames. You can change the duration of a sprite—that is, the amount of time the sprite appears in a movie—by adjusting its length, changing the number of frames in which it appears, or using the Extend command.





Director maintains the spacing proportions of keyframes when a sprite is lengthened. For a description of keyframes, see **Animation: Overview**.

To extend or shorten a sprite:

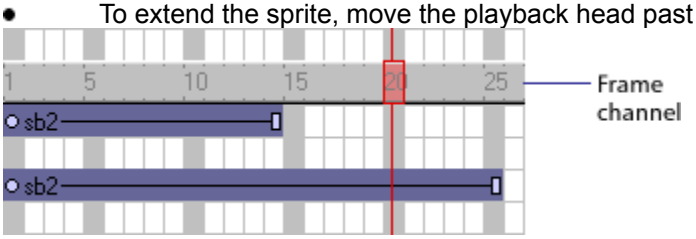
1. Choose Window > Score to display the Score.
2. Do one of the following:

- Drag the start or end frames. To extend a one-frame sprite, Alt-drag (Windows) or Option-drag (Macintosh).
- To extend a sprite and leave the last keyframe in place, Alt-drag (Windows) or Option-drag (Macintosh) a keyframe at the end of the sprite.
- To extend a sprite and leave all keyframes in place, Control-drag (Windows) or Command-drag (Macintosh) the end frame.
- Enter new values in the Start and End fields of the Sprite tab of the Property Inspector to change the start and end frames.

To extend a sprite to the current location of the playback head:

1. Select the sprite or sprites to extend.
2. Click the frame channel to move the playback head:

- To extend the sprite, move the playback head past the right edge of the sprite.



- To shorten the sprite, move the playback head to the left of the sprite's right edge, inside the sprite.
- To move the sprite's start frame, place the playback head to the left of the sprite.

3. Choose Modify > Extend Sprite.

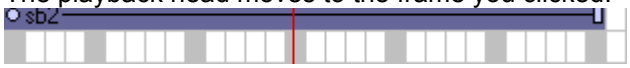
Splitting and joining sprites

You may need to split an existing sprite into two separate sprites or join separate sprites. If, for example, you created a complex animation as separate sprites and now want to move the entire sequence in the Score, you would join the sprites. Splitting and joining also lets you update movies created with older versions of Director that may have several fragmented sprites.

To split an existing sprite:

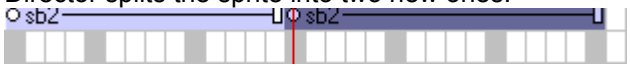
1. In the Score, click the frame within a sprite where the split will occur.

   The playback head moves to the frame you clicked.

2. 

3. Choose Modify > Split Sprite.

   Director splits the sprite into two new ones.

4. 

To join separate sprites into a single sprite:

1. Select the sprites you want to join, as described in **Selecting sprites**.

   Director fills the gaps between the selected sprites. You can also select sprites in several channels. Director joins selected sprites in each individual channel.

2. Choose Modify > Join Sprite.

Changing the appearance of sprites

You can change the appearance of sprites on the Stage without affecting the cast member assigned to the sprite. You can resize, rotate, skew, flip, and apply new foreground and background colors to sprites. Applying these changes allows you to reuse the same cast member to create several different versions of an image. For example, you can create a flipped and rotated sprite with a new color. Since each cast member adds to downloading time, reusing cast members in this way reduces the number of cast members in your movie and makes it download faster.

Resizing and scaling sprites

You can resize sprites directly on the Stage by dragging their handles. To resize the sprite precisely, you can enter coordinates or scale sprites by a specified percentage in the Sprite tab of the Property Inspector. You can also set the sprite's size with Lingo.

Changing a sprite's size on the Stage doesn't change the size of the cast member assigned to the sprite, nor is the size of the sprite affected if you resize its cast member.

In some cases, resizing bitmap sprites can cause noticeable delays. If a bitmap sprite must be a particular size, make the cast members displayed in the sprite the proper size. You can do this with Modify > Transform Bitmap or in any image editing program. Scaling and resizing sprites works best with vector shapes.

Note: The procedure for resizing a rotated or skewed sprite is different from the procedures that follow. See **Rotating and skewing sprites**.

To resize a sprite by dragging its handles:

1. Select the sprite.
2. On the Stage, drag any of the sprite's resize handles. Hold down Shift while dragging to maintain the sprite's proportions.

To scale a sprite by pixels or by an exact percentage:

1. Select the sprite you want to scale and click the Sprite tab of the Property Inspector (Graphical view).
2. Click the Scale button.

   The Scale Sprite dialog box appears.

3. Enter new values to scale the sprite by doing one of the following:

- Specify a pixel size in the Width or Height field. If Maintain Proportions is selected, all of the updatable fields adjust to reflect the new scaled size. If Maintain Proportions in not selected, you can specify new proportions in the Width and Height fields.
- Enter a percentage in the Scale field.

4. Click OK.

   The sprite is scaled relative to its current size, not to the size of its parent cast member.

To restore a sprite to its original dimensions, do one of the following:

- On the Sprite tab of the Property Inspector (Graphical view), click Restore All.
- Choose Modify > Transform > Reset Width and Height or Reset All.

To resize a sprite's bounding rectangle with Lingo:

Set the sprite's `quad` or `rect` sprite property. See **quad** or **rect (sprite)**.

The `rect` sprite property determines the coordinates of a sprite's bounding rectangle. The coordinates are given as a `rect` value, which is a list of the left, top, right, and bottom coordinates.

To change a sprite's height or width with Lingo:

Set the `height` or `width` sprite property. See **height** and **width**.

Rotating and skewing sprites

You can rotate and skew sprites to turn and distort images and to create dramatic animated effects. You rotate and skew sprites on the Stage by dragging. To rotate and skew sprites more precisely, use Lingo or the Property Inspector to enter degrees of rotation or skew. The Property Inspector is also useful for rotating and skewing several sprites at once by the same angle.

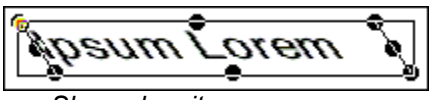
Director can rotate and skew bitmaps, text, vector shapes, Flash movies, QuickTime videos, and animated GIFs.

Director rotates a sprite around its registration point, which is a marker that appears on a sprite when you select it with your mouse. By default, Director assigns a registration point in the center of all bitmaps. You can change the location of the registration point using the Paint window. See **Changing registration points**.

To see animations of rotating and skewing, see the **Sprite Rotation** movie and the **Sprite Skewing** movie.

Rotation changes the angle of the sprite. Skewing changes the corner angles of the sprite's rectangle.

Rotated sprite



Skewed sprite

After a sprite is rotated or skewed, you can still resize it.

Director can automatically change rotation and skew from frame to frame to create animation. See **Tweening other sprite properties**.

To rotate or skew a sprite on the Stage:

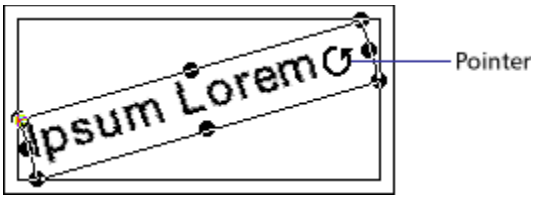
1. Select a sprite on the Stage.
2. Choose Window > Tool Palette to display the Tool palette.
3. Click the Rotate tool in the Tool palette.

   You can also press Tab while the Stage window is active to choose the Rotate tool.

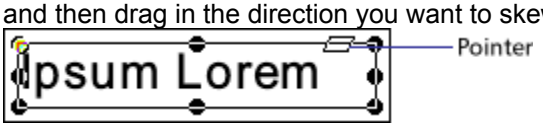
   The handles around the sprite change to indicate the new mode.

4. Do either of the following:

- To rotate the sprite, move the pointer inside the sprite and drag in the direction you want to rotate.

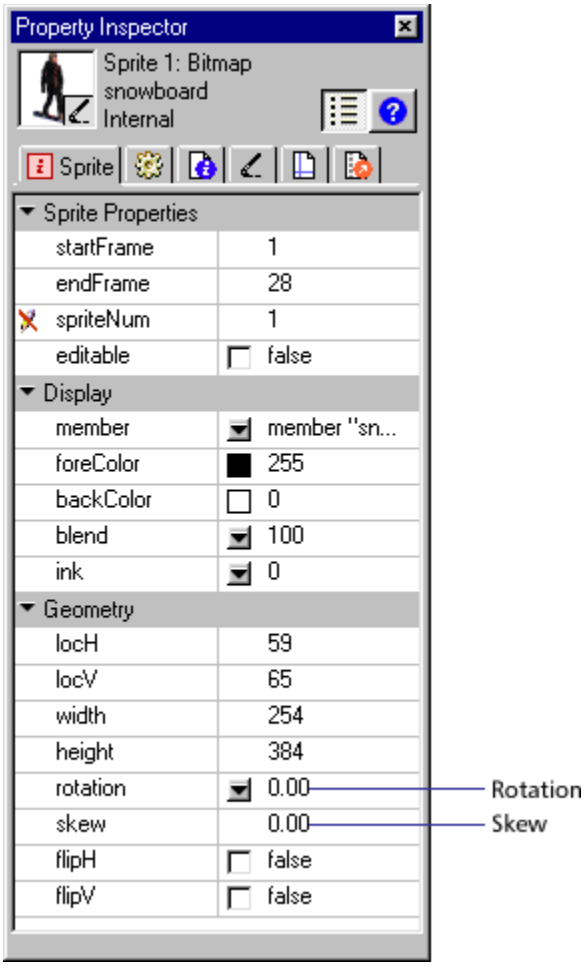


- To skew the sprite, move the pointer to the edge of the sprite until it changes to the skew pointer and then drag in the direction you want to skew.



To rotate or skew a sprite with the Property Inspector:

1. Select the sprite you want to rotate or skew and click the Sprite tab of the Property Inspector (List view).
2. To rotate the selected sprite, enter the number of degrees in the Rotation field.
3. To skew the selected sprite, enter the number of degrees in the Skew field.



To resize a rotated or skewed sprite, do one of the following:

- Click the Rotate or Skew tool and drag any of the sprite's handles. Alt-drag (Windows) or Option-drag (Macintosh) to maintain the sprite's proportions as you resize.

- Enter new values in the Sprite tab of the Property Inspector.
  Director resizes the sprite at the current skew or rotation angle.

To restore a skewed or rotated sprite to its original orientation:

Choose Modify > Transform > Reset Rotation and Skew or Reset All.

To skew a sprite with Lingo:

Set the `skew` sprite property. See **skew**.

Flipping sprites

Flipping a sprite creates a horizontally or vertically inverted image of the original sprite.

To flip a sprite:

1. Select a sprite.
2. Do any of the following:

- Click the Flip Vertical or Flip Horizontal button on the Sprite tab of the Property Inspector to flip the sprite without moving the registration point or changing the current skew or rotation angles.
- Choose Modify > Transform > Flip Horizontal in Place or Flip Vertical in Place to flip the sprite so that its bounding rectangle stays in place and the registration point is moved, if necessary.
- Choose Modify > Transform > Mirror Horizontal or Mirror Vertical to flip the sprite without moving the registration point, but inverting the skew and rotation angles.

Changing the color of a sprite

You can tint or color sprites by choosing new foreground and background colors from the Property Inspector or with Lingo. Choosing a new foreground color changes black pixels within the sprite to the selected color and blends dark colors with the new color. Choosing a new background color changes white pixels within the sprite to the selected color and blends light colors with the new color.

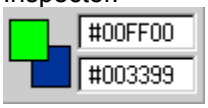
Director can animate foreground and background color changes in sprites, shifting gradually between the colors you specify in the start and end frames of a sprite. See **Tweening other sprite properties**.

To reverse the colors of an image, change the foreground color to white and the background color to black.

To change the color of a sprite:

1. Select a sprite.
2. Do one of the following:

- Choose colors from the Foreground and Background color boxes in the Sprite tab of the Property Inspector.



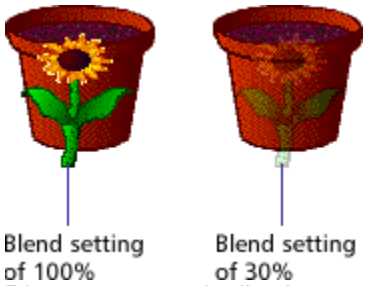
- Enter RGB values (hexadecimal) or palette index values (0-255) for the foreground and background colors in the Sprite tab of the Property Inspector.

To change the color of a sprite with Lingo, set the appropriate sprite property:

- The `color` sprite property sets the sprite's foreground color. The value is an RGB value. See **color (sprite property)**.
- The `bgColor` sprite property sets the sprite's background color. The value is an RGB value. See **bgColor**.

Setting blends

You can use blending to make sprites transparent. To change a sprite's blend setting, use the Sprite tab in the Property Inspector.

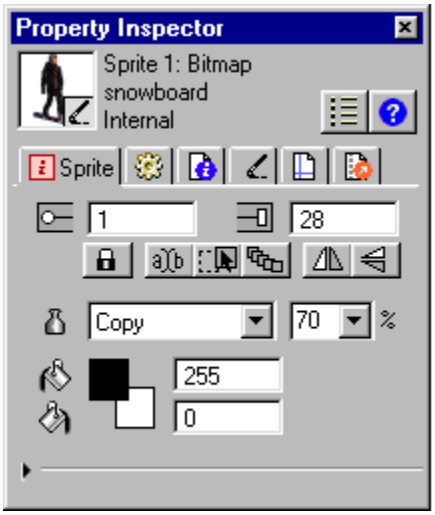


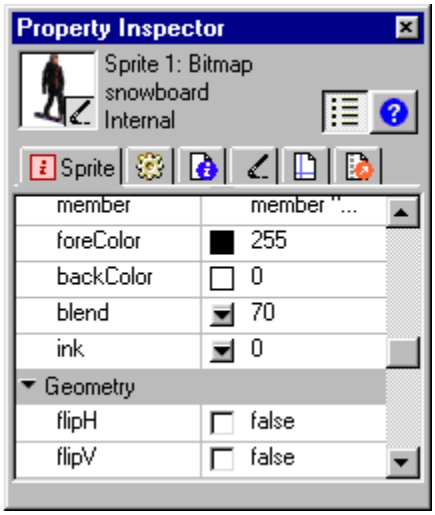
Director can gradually change blend settings to make sprites fade in or out. See **Tweening other sprite properties**.

The Blend percentage value affects only Copy, Background Transparent, Matte, Mask, and Blend inks.

To set blending for a sprite:

1. Select the sprite.
2. Choose a percentage from the Blend pop-up menu in the Property Inspector.





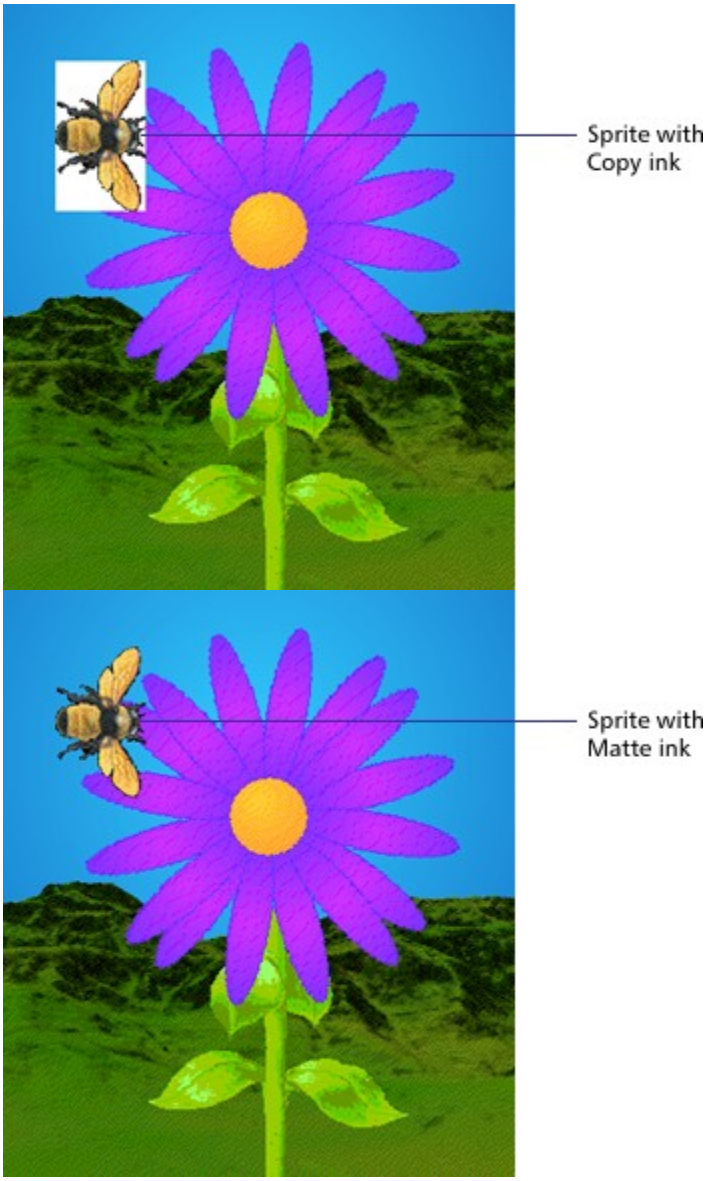
To set blending with Lingo:

Set the `blend` sprite property. See **blend**.

Using sprite inks

You can change a sprite's appearance on the Stage by applying inks. Sprite inks change the display of a sprite's colors. Inks are most useful to hide white bounding rectangles around images, but they can also create many compelling and useful color effects. Inks can reverse and alter colors, make sprites change colors depending on the background, and create masks that obscure or reveal portions of a background.

You change the ink for a sprite in the Property Inspector or with Lingo.



For a demonstration and description of all the inks, see the **Ink Effects** movie.
To achieve the fastest animation rendering on the screen, use Copy ink; other ink types may have a slight effect on performance.

To change a sprite's ink with the Property Inspector:

1 Select the sprite.

2 Choose the desired type of ink from the Ink pop-up menu in the Sprite tab of the Property Inspector.

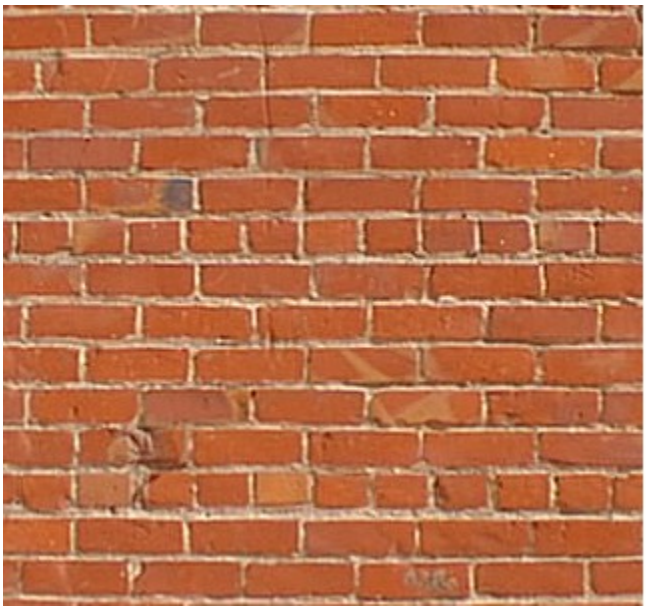
To change a sprite's ink with Lingo:

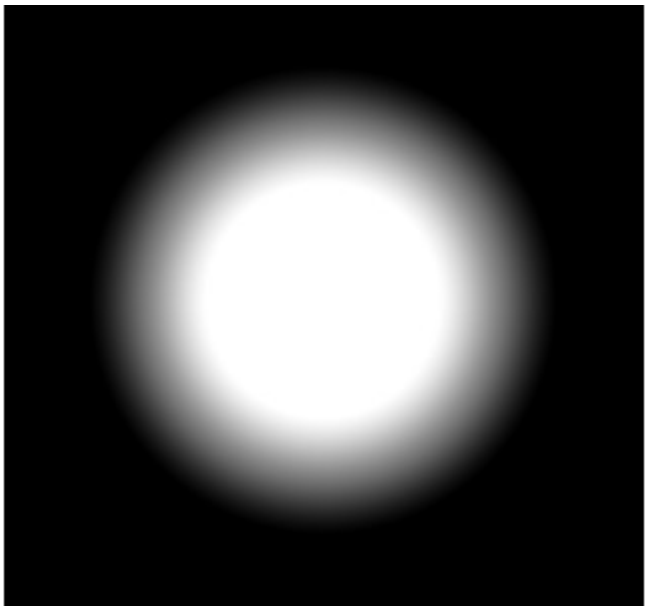
Set the sprite's `ink` sprite property. See **ink**.

Note: If Background Transparent and Matte inks don't seem to work, the background of the image may not be true white. Also, if the edges of the image have been blended or are fuzzy, applying these inks may create a halo effect. Use the Paint window or an image editing program to change the background to true white and harden the edges. You can also re-create the image with an alpha channel (transparency) and re-import the image.

Using Mask ink to create transparency effects

To reveal or tint certain parts of a sprite, you use Mask ink. Mask ink lets you define a mask cast member, which controls the degree of transparency for parts of a sprite.





Black areas of a mask cast member make the sprite completely opaque in those areas, and white areas make it completely transparent (invisible). Colors between black and white are more or less transparent; darker colors are more opaque.

When creating a bitmap mask for a sprite, use a grayscale palette if the mask cast member is an 8-bit (or less) image. An 8-bit mask affects only the transparency of the sprite and does not affect the color. Director ignores the palette of mask cast members that are less than 32-bit images; using a grayscale palette lets you view the mask in a meaningful way. If your mask cast member is a 32-bit image, the colors of the mask tint the sprite's colors.

If you do not need variable levels of opacity, use a 1-bit mask cast member to conserve memory and disk space.

There are many ways to use Mask ink, but the following procedure explains the most basic method.

To use Mask ink:

1. Decide which cast member you want to mask.

   The cast member can be a bitmap of any depth.

2. In the next position in the same cast, create a duplicate of the cast member to serve as the mask.

   The mask cast member can actually be any image, but a duplicate of the original is usually the most useful.

3. Edit the mask cast member in the Paint window or any image editor.

   Black areas of the mask make the sprite completely opaque in those areas, and white areas make it completely transparent (invisible).

4. Drag the original cast member to the Stage or Score to create a sprite.

5. Make sure the new sprite is selected and choose Mask ink from the Ink pop-up menu in the Sprite

tab of the Property Inspector.

Only the parts of the sprite revealed by the mask are visible on the Stage.

About Darken and Lighten inks

Darken and Lighten inks provide a great deal of control over the RGB properties of a sprite. You use them to create color effects in sprites varying from the subtle to the surreal.

Darken and Lighten both change how Director applies the foreground and background color properties of a sprite. Darken makes the background color equivalent to a color filter through which the sprite is viewed on the Stage. Lighten tints the colors in a sprite lighter as the background color gets darker. For both inks, the foreground color is added to the image to the degree allowed by the other color control. Neither ink has any effect on a sprite until you change the foreground or background color from the default settings of black and white.

Darken and Lighten are especially useful for animating unusual color effects. Because the Foreground and Background color properties of the sprite control the effects, you can animate color shifts to create dazzling effects without having to manually edit colors in a cast member. See **Tweening other sprite properties**.

Ink definitions

Following are definitions of all available ink types.

Copy displays all the original colors in a sprite. All colors, including white, are opaque unless the image contains alpha channel effects (transparency). Copy is the default ink and is useful for backgrounds or for sprites that do not appear in front of other artwork. If the cast member is not rectangular, a white box appears around the sprite when it passes in front of another sprite or is displayed on a nonwhite background. Sprites with the Copy ink animate faster than sprites with any other ink.

Matte removes the white bounding rectangle around a sprite. Artwork within the boundaries is opaque. Matte functions much like the Lasso tool in the Paint window in that the artwork is outlined rather than enclosed in a rectangle. Matte, like Mask, uses more RAM than the other inks, and sprites with this ink animate more slowly than other sprites.

Background Transparent makes all the pixels in the background color of the selected sprite appear transparent and permits the background to be seen.

Transparent makes all light colors transparent so you can see lighter objects beneath the sprite.

Reverse reverses overlapping colors. When applied to the foreground sprite, where colors overlap, the upper color changes to the chromatic opposite (based on the color palette currently in use) of the color beneath it. Pixels that were originally white become transparent and let the background show through unchanged. Reverse is good for creating custom masks.

Ghost, like Reverse, reverses overlapping colors, except nonoverlapping colors are transparent. The sprite is not visible unless it is overlapping another sprite.

Not Copy reverses all the colors in an image to create a chromatic negative of the original.

Not Transparent, Not Reverse, and **Not Ghost** are all variations of other effects. The foreground image is first reversed, then the Copy, Transparent, Reverse, or Ghost ink is applied. These inks are good for creating odd effects.

`Mask` determines the exact transparent or opaque parts of a sprite. For Mask ink to work, you must place a mask cast member in the Cast window position immediately following the cast member to be masked. The black areas of the mask make the sprite opaque, and white areas are transparent. Colors between black and white are more or less transparent; darker colors are more opaque. See **Using Mask ink to create transparency effects**.

Blend ensures that the sprite uses the color blend percentage specified in the Sprite tab of the Property Inspector. See **Setting blends**.

Darkest compares RGB pixel colors in the foreground and background and uses whichever pixel color is darkest.

Lightest compares RGB pixel colors in the foreground and background and uses whichever pixel color is lightest.

Add creates a new color that is the result of adding the RGB color value of the foreground sprite to the color value of the background sprite. If the value of the two colors exceeds the maximum RGB color value (255), Director subtracts 256 from the remaining value so that the result is between 0 and 255.

Add Pin is similar to Add. The foreground sprite's RGB color value is added to the background sprite's RGB color value, but the value of the new color cannot exceed the maximum color value (255).

Subtract subtracts the RGB color value of the foreground sprite's color from the RGB value of the background sprite's color to arrive at the new color. If the color value of the new color is less than 0,

Director adds 256 so the remaining value is between 0 and 255.

Subtract Pin subtracts the RGB color value of pixels in the foreground sprite from the value of the background sprite. The value of the new color cannot be less than 0.

Darken changes the effect of the Foreground and Background color properties of a sprite to create dramatic color effects that generally darken and tint a sprite. Darken ink makes the background color equivalent to a color filter through which the sprite is viewed on the Stage. White provides no filtering; black darkens all color to pure black. The foreground color is then added to the filtered image, creating an effect similar to shining light of that color onto the image. Choosing Darken ink has no effect on a sprite until you select nondefault foreground and background colors. See **About Darken and Lighten inks**.

Lighten changes the effect of the Foreground and Background color properties of a sprite so that it is easy to create dramatic color effects that generally lighten an image. Lighten ink makes the colors in a sprite lighter as the background color gets darker. The foreground color tints the image to the degree allowed by the lightening. See **About Darken and Lighten inks**.

Note: Mask and Matte use more memory than other inks because Director must duplicate the mask of the artwork.

Assigning a cast member to a sprite with Lingo

Several Lingo properties specify the cast member assigned to a sprite. You can use these properties to determine a sprite's cast member and switch the sprite's cast members as the movie plays.

To specify the cast member, including its cast:

Set the member sprite property. See **member (sprite property)**.

Setting this property is the most reliable way to specify a sprite's cast member. You can also set the `memberNum` sprite property, but this is reliable only when the new cast member is in the same cast as the current cast member.

To determine which cast contains the cast member assigned to a sprite:

Test the `castLibNum` sprite property. See **castLibNum**.

This procedure can be useful for updating movies that serve as templates.

Behaviors: Overview

A behavior is prewritten Lingo script that you use to provide interactivity and add interesting effects to your movie. You drag a behavior from the Library palette and drop it on a sprite or frame to attach it.

If the behavior includes parameters, a dialog box appears that lets you define those parameters. For example, most navigation behaviors let you specify a frame to jump to. You can attach the same behavior to as many sprites or frames as necessary and use different parameters for each instance of the behavior.

Most behaviors respond to simple events such as a click on a sprite or the entry of the playback head into a frame. When the event occurs, the behavior performs an action, such as jumping to a different frame or playing a sound.

Director comes packaged with customizable, reusable behaviors for many basic functions; you and other developers can also create your own behaviors by writing Lingo script. To modify behaviors, you use the Behavior Inspector or Property Inspector.

For more information about using included behaviors, see **Using Director 8 Behaviors** in the Director Support Center.

Attaching behaviors

You use the Library palette to display behaviors included in Director.

Director allows you to attach the same behavior to several sprites or several frames at the same time. You can attach as many behaviors as you want to a sprite, but you can attach only one behavior to a frame. If you attach a behavior to a frame that already has a behavior, the new behavior replaces the old one. Behaviors attached to frames are best suited to actions that affect the entire movie. For example, you might attach `Loop Until Media in Frame is Available` to make the movie wait while the media for a particular frame downloads.

When you attach a behavior, and the Parameters dialog box appears, note that the parameters you specify apply to the behavior only as it is attached to the current sprite or frame. These settings do not affect the way the behavior works when attached elsewhere. Use the Behavior Inspector to change parameters for behaviors attached to sprites or frames.

Once you attach a behavior to a sprite or frame, Director copies the behavior from the Behavior Library to the currently selected cast in the movie. This means you don't have to include the Behavior Library when you distribute the movie.

To attach a behavior to a single sprite or frame using the Library palette:

1. Choose Window > Library Palette.
2. Choose a library from the Library pop-up menu in the upper left corner of the palette.

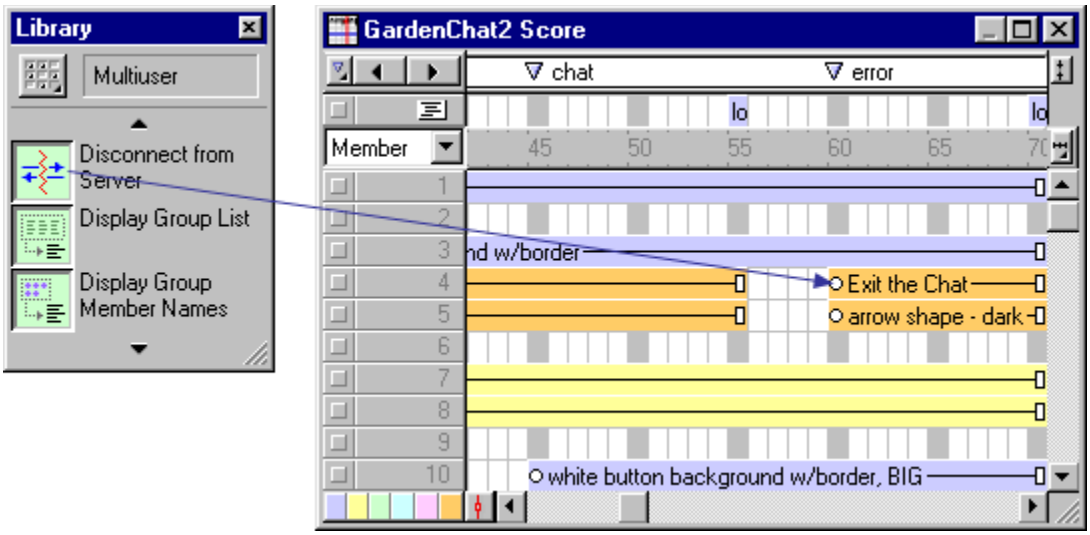
3. To view a brief description of included behaviors, move the pointer over a behavior icon.

   If the behavior includes a longer description, you can view it in the Behavior Inspector. See **Getting information about behaviors**. The behaviors included with Director come with descriptions. Behaviors from other sources may not.

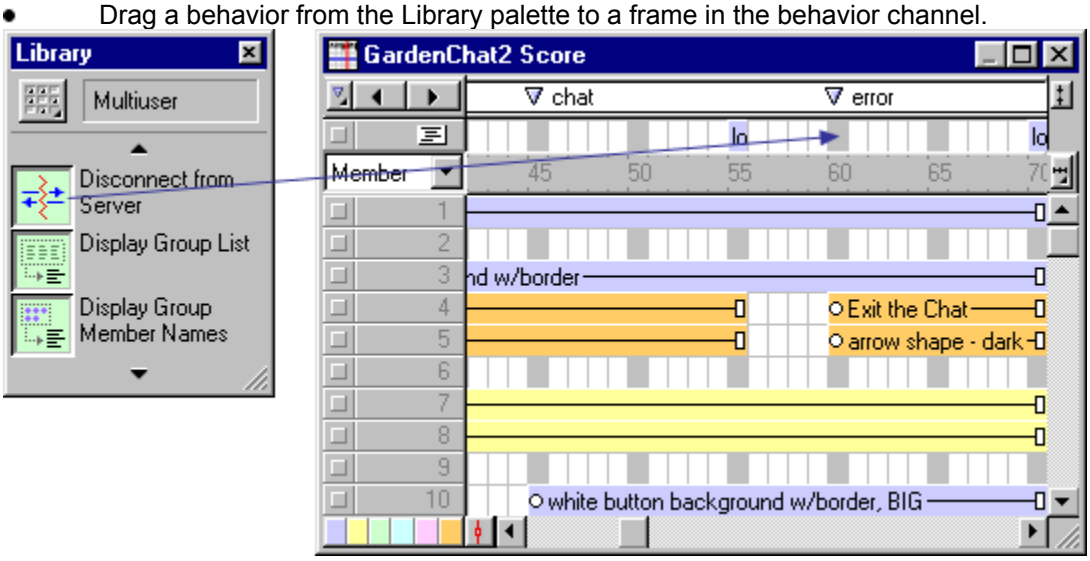
   Choose Show Names from the Library pop-up menu to turn the display of behavior names off or on.

4. To attach a behavior to a single sprite, do one of the following:

- Drag a behavior from the Library palette to a sprite on the Stage or in the Score.



- Drag a behavior from the Library palette to a frame in the behavior channel.



5 Enter parameters for the behavior in the Parameters dialog box.

Note: If you attach a behavior from a Director library of behaviors, the behavior is copied to an internal cast, to prevent you from accidentally changing the original behavior.

To attach the same behavior to several sprites at once using the Library palette:

Select the sprites on the Stage or in the Score and drag a behavior to any one of them.

To attach behaviors that are already attached to a sprite or frame:

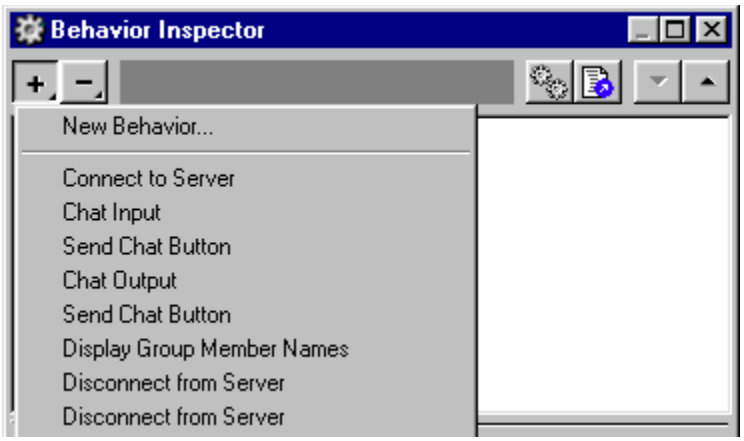
1 Choose Window > Inspectors > Behavior to open the Behavior Inspector.

2 Do one of the following:

- Select a sprite or several sprites.
- Select a frame or several frames.

3 Choose a behavior from the Behaviors pop-up menu.

Director attaches the behavior you choose to the sprite(s) or frame(s).



Note: Some behaviors work only when applied to either a sprite or a frame; read the behavior descriptions to learn more.

To change parameters for a behavior attached to a sprite or frame:

1. Select the sprite or frame to which the behavior is attached.
2. In the Behavior tab of the Property Inspector, use the pop-up menus or text fields to change any parameters.

   The Behavior tab includes the same fields for the behavior as those included in the Parameters dialog box.

Changing the order of attached behaviors

Director executes behaviors in the order they were attached to a sprite, and they are listed in this order in the Property and Behavior Inspectors. It's sometimes necessary to change the sequence of behaviors so that actions occur in the proper order.

To change the order of the behaviors attached to a sprite:

1. Select the sprite in the Score or on the Stage.
2. Click the Behavior tab in the Behavior or Property Inspector.
3. Select a behavior from the list.
4. Click the arrows in the toolbar to move the selected behavior up or down on the list. 

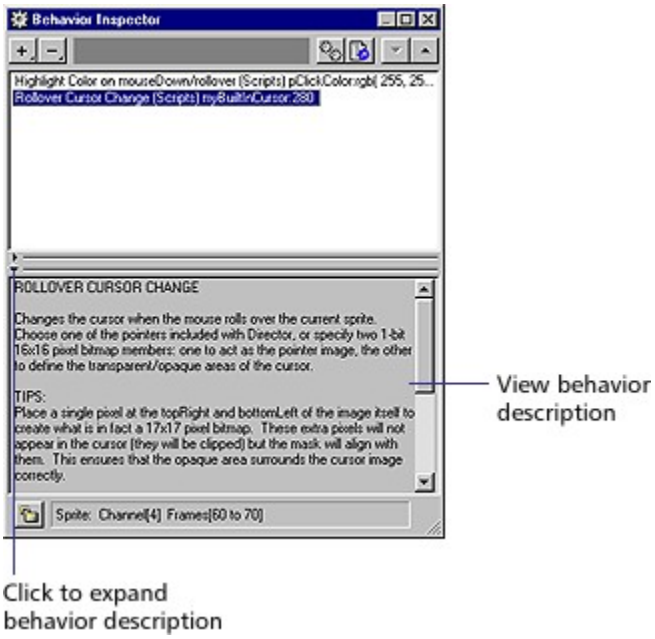
Getting information about behaviors

Behaviors included with Director have pop-up descriptions that appear when you hold the pointer over a behavior in the Library palette. Some behaviors, however, have longer descriptions and instructions, which you can view in the Behavior Inspector. A scrolling pane in the Behavior Inspector displays the complete description provided by the behavior's author. The Behavior Inspector only displays information about a behavior attached to a sprite or frame.

To view a behavior description:

1. Open the Behavior Inspector.
2. Select a sprite or frame to which a behavior has been attached.
3. Click the arrow that expands the Behavior Inspector's description pane.

   You can leave the description pane expanded and select different behaviors to see their descriptions.

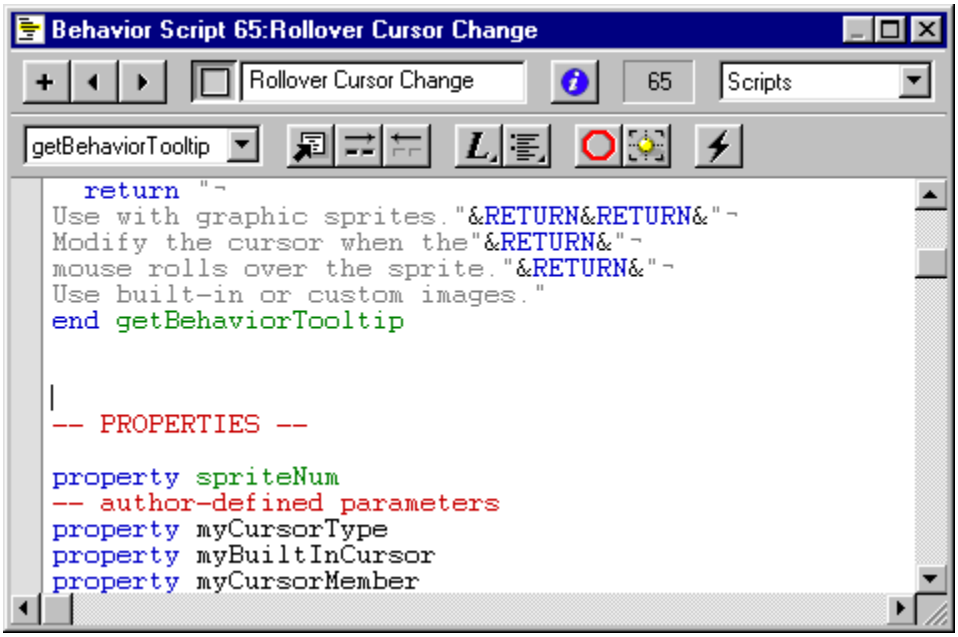


All of the behaviors included in the Director library have descriptions. Behaviors from other developers may not.

Creating and modifying behaviors

Without any scripting or programming experience, you can use the Behavior Inspector to create and modify behaviors to perform basic actions. To create behaviors with more complex structures, you need to understand Lingo.

Using the Behavior Inspector is a good way to learn Lingo. You can examine the scripts created by the Behavior Inspector to see how basic functions are assembled. Select any behavior and click the Script button to view the associated Lingo script.



All behaviors detect an event and then perform one or more actions in response. The Behavior Inspector lists the most common events and actions used in behaviors.
For experienced Lingo programmers, the Behavior Inspector also provides a shortcut for writing simple scripts.

Note: To always edit behaviors in the Script window instead of the Behavior Inspector, choose File > Preferences > Editors. In the Editors Preferences dialog box, choose Behaviors from the list and then click Editor. In the Select Editor box, choose Script Window.

To create or modify a behavior:

1. Do one of the following:

- To create a new behavior, click the Behaviors pop-up menu, choose New Behavior, and enter a name for the new behavior.
- The behavior appears in the currently selected Cast window, in the first empty position. Select an empty cast position first if you want the behavior to appear in a different place.
- To modify a behavior, select it in the Behavior Inspector.

2. Click the arrow in the lower left of the Behavior Inspector to expand the editing pane.

The editing pane shows the events and actions in the current behavior. If you're creating a new behavior, no events or actions appear.

- To add a new event or action group to the behavior, choose an event from the Events pop-up menu and then choose actions for the event from the Actions pop-up menu.

  You can choose as many actions as you need for a single event.

- To change an existing event or action group, choose an event from the list and then add or remove actions in the Actions list.
- To delete an event or action group, choose the event and press Delete.
- To change the sequence of actions in an event or action group, choose an event from the Events list, choose an action from the Actions list, and then click the up and down arrows above the Actions list to change the order of actions.
- To lock the current selection so nothing changes in the Behavior Inspector when new sprites are selected, click the Lock Selection button in the lower left of the expanded Behavior Inspector.

If you are familiar with Lingo, you can also edit a behavior's script directly.

Events and actions in the Behavior Inspector

The actions and events included with Director are basic building blocks you can use to create simple or complex behaviors.

The Behavior Inspector makes the following events available:

Mouse Up indicates that the mouse button was released.

Mouse Down indicates that the mouse button was clicked.

Right Mouse Up indicates that the right mouse button was released. (On the Macintosh, Director treats a Control-click the same as a right mouse click on a Windows system.)

Right Mouse Down indicates that the right mouse button was clicked.

Mouse Enter indicates that the pointer entered a sprite's region.

Mouse Leave indicates that the pointer left a sprite's region.

Mouse Within indicates that the pointer is within the sprite's region.

Key Up indicates that a key was released.

Key Down indicates that a key was pressed.

Prepare Frame indicates that the playback head has left the previous frame but has not yet entered the next frame.

Enter Frame indicates that the playback head has entered the current frame.

Exit Frame indicates that the playback head has exited the current frame.

New Event indicates that a specified message was received from a script or behavior. You must specify a name for this event.

The Behavior Inspector makes the following actions available:

Go to Frame moves the playback head to the specified frame.

Go to Movie opens and plays the specified movie.

Go to Marker moves the playback head to the specified marker.

Go to Net Page goes to the specified URL.

Wait on Current Frame waits at the current frame until another behavior or script advances to the next frame.

Wait until Click waits at the current frame until the mouse button is clicked.

Wait until Key Press waits at the current frame until a key is pressed.

Wait for Time Duration waits at the current frame for the specified time.

Play Cast Member plays the specified sound cast member.

Play External File plays the specified external sound file.

Beep plays the current system beep.

Set Volume sets the system volume level to the specified setting.

Change Tempo changes the movie's tempo to the specified setting.

Perform Transition performs the specified transition.

Change Palette changes to the specified palette.

Change Location moves the current sprite to the specified coordinates.

Change Cast Member switches the sprite's cast member to the specified cast member.

Change Ink switches to the specified ink.

Change Cursor changes the pointer to a shape you choose from the pop-up menu.

Restore Cursor restores the current system pointer.

New Action executes any Lingo function or sends a message to a handler. You specify the new handler's name.

Writing behaviors with Lingo

If you are familiar with Lingo, you can author your own behaviors.

From the perspective of Lingo, a behavior is a script with these additional features:

- Each instance of the behavior has independent values for properties. Lingo uses a `property` statement to declare properties that can have independent values in each instance of the behavior. See **property**.
- The same set of handlers can be shared by multiple sprites or frames.

  The handlers in a behavior are basically the same as other handlers. Include as many handlers as appropriate to implement the behavior.

  A behavior is usually attached to multiple sprites or frames. As a result, the sprites and frames share the same handlers. Director tracks which instance of the behavior is which by assigning each instance a reference number. The variable `me` contains the reference for the object that the instance of the behavior is attached to.

  In many cases it's most efficient to create behaviors dedicated to specific tasks and then attach a set of behaviors that perform the variety of actions you want.
- The behavior can have parameters that users edit from the Parameters dialog box. The optional `on getPropertyDescriptionList` handler sets up the Parameters dialog box. See **on getPropertyDescriptionList**.
- A description of the behavior can be added to the Behavior Inspector. The optional `on getBehaviorDescription` handler displays a description of the behavior in the Behavior Inspector. See **on getBehaviorDescription**.
- A brief description appears as a tooltip for the behavior in the Library palette if the optional `on getBehaviorToolTip` handler that creates the tooltip has been written. See **on getBehaviorTooltip**.

Setting up a Parameters dialog box

It's impossible to predict exactly what a user will want behaviors to do. You can make behaviors more flexible by letting the user customize the behavior's parameters.

For example, this handler moves the sprite 5 pixels to the right each time the playback head enters a new frame:

```
on enterFrame me
  if the locH of sprite the spriteNum of me > the ¬
  stageRight then
    set the locH of sprite the spriteNum of me = the stageLeft
  else
    set the locH of sprite the spriteNum of me to ¬
    (the locH of sprite the spriteNum of me + 5)
  end if
end
```

However, users could adjust the speed of each sprite if they could specify how far individual sprites move to the right in each frame.

To allow users to set different values for a property in different instances of the behavior, the behavior's script needs two types of Lingo:

- A `property` statement that allows each instance to maintain a separate value for the property
- An `on getPropertyDescriptionList` handler that sets up the property

Setting behavior properties with Lingo

Behaviors usually have properties for which each instance of the behavior maintains its own values. (An instance is each sprite or frame that the behavior is attached to.) These properties are shared among handlers in a behavior's script the same way that properties are shared among handlers in an object.

To declare which properties can have independent values in each instance of the behavior:

Put the `property` statement at the beginning of the behavior's script.

A `property` statement starts with the word `property` followed by the names of the individual properties. For example, the statement `property movement` declares that `movement` is a property of the behavior.

Customizing a behavior's property

If a behavior's script includes an `on getPropertyDescriptionList` handler, Director lets users set the property's initial values from the Parameters dialog box. The behavior's Parameters dialog box opens in three circumstances:

- After the user drags a behavior to a sprite or frame
- When the user double-clicks the behavior in the Behavior Inspector dialog box
- When the user clicks the Parameters button in the Behavior Inspector

The `on getPropertyDescriptionList` handler generates a property list that specifies these attributes of the property:

- The default initial value
- The type of data the property contains, such as Boolean, integer, string, cast member, or a specific type of cast member
- A comment in the Parameters dialog box to describe what the user is setting

The definition of a behavior's property must include the property's name, default value, and data type and the descriptive string that appears in the Parameters dialog box. The definition can also include an optional specification for the range of values allowed for the property.

The name of the property comes first in the definition. The remainder of the definition is a property list that assigns a value to each of the property's attributes.

For example, to define the property `movement` as an integer that can be set to a value from 1 to 10 and whose default value is 5, use a phrase similar to this:

```
#movement: [#default: 5, #format:#integer, ¬
#comment: "Set motion to the right:", #range: [#min:1, #max:10]]
```

- **#**`movement` is the property's name. A symbol (`#`) operator must precede the name in the property definition. A colon separates the name's definition and the list of parameters.
- `#default` specifies the property's default value. This example sets 5 as the default.
- `#format` specifies the property's type. This example sets the type as an integer. Some other possible types are Boolean, string, cast member, event, and sound. For a complete list of possible values for `#format`, see `on getPropertyDescriptionList` in Director Help or the *Lingo Dictionary*.
- `#comment` specifies a string that appears next to the parameter in the Parameters dialog box. This example makes "Set motion to the right" the comment that appears in the Parameters dialog box.
- `#range` specifies a range of possible values that the user can assign to the property. Specify the possible values as a list.

  To specify a range between a minimum and maximum number, use the form [`#min:minimum, #max:maximum`]**.** The example sets the range from 1 to 10. When the range is between a maximum or minimum number, the Parameters dialog box provides a slider that sets the value.

  To specify no range, omit the `#range` parameter. If the property's definition doesn't include `#range`, a text entry field appears for the user to enter a value in the Parameters dialog box.

  To specify a set of possible choices, use a linear list. For example, the list `[#mouseUp, #mouseDown, #keyUp, #keyDown]` makes these four events possible choices for a parameter. When you specify values in a linear list, the choices appear in a pop-up menu in the Parameters dialog box. (For this example list, you need to specify `#format: #symbol` for the list to display correctly.)

As another example, this statement defines the property `whichSound`:

```
addProp description, #whichSound, [#default: "", #format:#sound¬
, #comment: ¬
"Which cast member"]
```

The value `#sound` assigned to `#format` provides a pop-up menu in the Parameters dialog box that includes every sound cast member available in the movie.

If the behavior includes a command that plays a sound, this property can be used to specify a sound cast member to play. For example, if the user chooses Growl from the pop-up menu in the Parameters dialog box, the statement `puppetSound whichSound` would play the sound cast member Growl.

Creating an on getPropertyDescriptionList handler

To build a list of properties for a behavior, add each property to the list that the `on getPropertyDescriptionList` handler returns. Then use the `return` command to return the list.

For example, this handler creates a property list named Description that contains the definitions for `movement` and `whichSound`:

```
on getPropertyDescriptionList
  set description = [:]
  addProp description, #Movement, [#default: 5,¬
    #format:#integer, #comment: ¬
  "Set motion to the right:", #range: [#min:1, #max:10]]
  addProp description, #noise, [#default:"", format: #sound, ¬
  #comment:"Sound cast member name"]
  return description
end
```

Alternatively, you can use this syntax to do the same as the previous handler:

```
on getPropertyDescriptionList
  return [¬

  #Movement: [#default: 5, #format:#integer, #comment: ¬
  "Set motion to the right:", #range: [#min:1, #max:10]],
  #noise: [#default:"", format: #sound, ¬
  #comment:"Sound cast member name"]
  ]

end
```

Including a description for the Behavior Inspector

An `on getBehaviorDescription` handler in a behavior's script provides a description that appears in the bottom pane of the Behavior Inspector when the behavior is selected. For example, this handler displays the phrase "This changes sprite color and position" in the Behavior Inspector:

```
on getBehaviorDescription
  return "This changes sprite color and position"
end
```

Example of a complete behavior

If the handlers described here were in one behavior, the script would look like this (the `puppetSound` command was added to the `on mouseUp` handler in this example):

```
property movement, noise
on getPropertyDescriptionList
  set description = [:]
  addProp description, #movement, [#default: 5, ¬
  #format:#integer, #comment: "Set motion to ¬
  the right:", #range: [#min:1, #max:10]]
  addProp description, #noise, [#default:"", ¬
  #format: #sound, #comment:"Sound cast ¬
  member name"]

  return description
end
on getBehaviorDescription
  return "This changes sprite color and position"
end
on mouseUp me
  set the foreColor of sprite the spriteNum of me ¬
  to random(255)
  puppetSound noise
end
on enterFrame me
  if the locH of sprite the spriteNum of me > ¬
  the stageRight then
    set the locH of sprite the spriteNum ¬
    of me = the stageLeft
  else
    set the locH of sprite the spriteNum of me to ¬
    (the locH of sprite the spriteNum of me + ¬
    movement)
  end if
end
```

When this behavior is attached to a sprite, each time the playback head enters a frame, the sprite moves to the right by the amount the user specifies. When the user clicks a sprite, its color changes and a specified sound plays.

Sending messages to behaviors attached to sprites

Lingo can run handlers in behaviors attached to specific sprites by sending messages to the behaviors attached to one sprite, all sprites, or several specific sprites.

Sending messages to a sprite

The `sendSprite` command sends a message to a specified sprite. If none of the sprite's behaviors has a handler that corresponds to the message, the message passes to the cast member script, the frame script, and then the movie script. See **sendSprite**.

For example, this handler sends the custom message `bumpCounter` and the argument 2 to sprite 1 when the user clicks the mouse:

```
on mouseDown me
  sendSprite (1, #bumpCounter, 2)
end
```

Note: The symbol (`#`) operator must precede the message in the `sendSprite` command.

Sending messages to all sprites

The `sendAllSprites` command sends a message to every sprite in the frame. If no behavior of the specified sprite has a handler that corresponds to the message, the message passes to the cast member script, the frame script, and then the movie script. See **sendAllSprites**.

For example, this handler sends the custom message `bumpCounter` and the argument 2 to all sprites in the frame when the user clicks the mouse:

```
on mouseDown me
  sendAllSprites (#bumpCounter, 2)
end
```

Note: The symbol (`#`) operator must precede the message in the `sendAllSprites` command.

Sending messages to specific behaviors only

The `call` command sends an event to specific behaviors. Unlike the `sendSprite` command, the `call` command doesn't pass the message to frame scripts, scripts of the cast member, or movie scripts.

Before sending a message to a specific behavior, check the `scriptInstanceList` sprite property to find a behavior script reference to use with the `call` command.

The `scriptInstanceList` property provides a list of references for the behaviors attached to a sprite while a movie is playing.

For example, this handler displays the list of references for all behaviors attached to the same sprite as this behavior's handler:

```
on showScriptRefs me
  put the scriptInstanceList of sprite the ¬
  spriteNum of me
end
```

This handler sends the message `bumpCounter` to the first script reference attached to sprite 1 (the

`getAt` function identifies the first script reference in the `scriptInstanceList`):

```
on mouseDown me
  xref = getAt (the scriptInstanceList of sprite 1,1)
  call (#bumpCounter, xref, 2)
end
```

Note: The symbol (`#`) operator must precede the message in the `call` command.

To remove instances of a sprite while the movie is playing:

Set the sprite's `scriptInstanceList` property to an empty list([]). See **scriptInstanceList**.

Using inheritance in behaviors

Behaviors can have ancestor scripts in the same way that parent scripts do. (Ancestor scripts are additional scripts whose handlers and properties a parent script can call on and use.)

- The ancestor's handlers and properties are available to the behavior.
- If a behavior has the same handler or property as an ancestor script, Lingo uses the property or handler in the behavior instead of the one in the ancestor.

For more information about the concept of ancestors and inheritance, see **Parent scripts: Overview**.

To make a script an ancestor:

- Declare that `ancestor` is a property in the `property` statement at the beginning of the behavior's Score script.

  For example, the statement `property ancestor` declares that `ancestor` is a property.

- Include a statement that specifies which script is the ancestor. Put the statement in an `on beginSprite` handler in the behavior.

  For example, this handler makes the script Common Behavior an ancestor of the behavior when Director first enters the sprite:

```
on beginSprite
  set the ancestor of me to new (script "Common Behavior")
end
```

  This handler will let the behavior also use the handler in the script Common Behavior.

Writing scripts with Lingo: Overview

Lingo, Director's scripting language, adds interactivity to a movie. You can use Lingo to control a movie in response to specific conditions and events. For example, Lingo can play a sound after a specified amount of the sound has streamed from the Internet.

Scripting basics

The information in this section introduces and explains basic Lingo scripting concepts that Director uses. If you are new to scripting, review this section before you begin writing scripts in Lingo.

Types of scripts

Director uses four types of scripts: behaviors, movie scripts, parent scripts, and scripts attached to cast members. Behaviors, movie scripts, and parent scripts all appear as cast members in the Cast window.

Behaviors are scripts that are attached to sprites or frames in the Score, and are referred to as sprite behaviors or frame behaviors. The Cast window thumbnail for each behavior contains a behavior icon in the lower right corner.

When used in this chapter, the term "behavior" refers to any Lingo script that you attach to a sprite or a frame. This differs from the behaviors that come in Director's Library Palette. For more information about Director's built-in behaviors, see **Behaviors: Overview**.

All behaviors that have been added to the cast appear in the Behavior Inspector's Behavior pop-up menu. (Other types of scripts don't appear there.)

You can attach the same behavior to more than one location in the Score. When you edit a behavior, the edited version is applied everywhere the behavior is attached in the Score.

Movie scripts respond to events such as key presses and mouse clicks, and can control what happens when a movie starts, stops, or pauses. Handlers in a movie script can be called from other scripts in the movie as the movie plays.

A movie script icon appears in the lower right corner of the movie script's Cast window thumbnail.

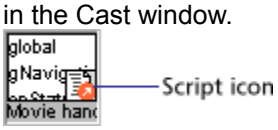
Movie scripts are available to the entire movie, regardless of which frame the movie is in or which sprites the user is interacting with. When a movie plays in a window or as a linked movie, a movie script is available only to its own movie.

Parent scripts are special scripts that contain Lingo used to create child objects. You can use parent scripts to generate script objects that behave and respond similarly yet can still operate independently of each other. A parent script icon appears in the lower right corner of the Cast window thumbnail.

For information about parent scripts, see **Parent scripts: Overview**.

Scripts attached to cast members are attached directly to a cast member, independent of the Score. Whenever the cast member is assigned to a sprite, the cast member's script is available.

Unlike behaviors, movie scripts, and parent scripts, cast member scripts don't appear in the Cast window. However, if Show Cast Member Script Icons is selected in the Cast Window Preferences dialog box, cast members that have a script attached display a small script icon in the lower left corner of their thumbnails in the Cast window.



How scripts flow

Director always executes Lingo statements starting with the first statement and continuing in order until it reaches the final statement or a statement that instructs Lingo to go somewhere else.

To set up statements so that they run when specific conditions exist, you use `if...then`, `case`, and `repeat` loop structures. For example, you can create an `if...then` structure that tests whether text has finished downloading from the Internet and, if it has, then attempts to format the text. See

Controlling flow in scripts.

The order in which statements are executed affects the order in which you should place statements. For example, if you write a statement that requires some calculated value, you need to put the statement that calculates the value first. For instance, in the following example, the first statement adds two numbers, and the second assigns a string representation of the sum to a field cast member to be displayed on the Stage:

```
x = 2 + 2
put string(x) into member "The Answer"
```

About planning and debugging scripts

When you write scripts for an entire movie, the quantity and variety of scripts can be very large. Deciding which Lingo commands to use, how to structure scripts effectively, and where scripts should be placed requires careful planning and testing, especially as the complexity of your movie grows.

Before you begin writing scripts, formulate your goal and understand what you want to achieve. This is as important—and typically as time consuming—as developing storyboards for your work.

When you have an overall plan for the movie, you are ready to start writing and testing scripts. Expect this to take time. Getting scripts to work the way you want often takes more than one cycle of writing, testing, and debugging.

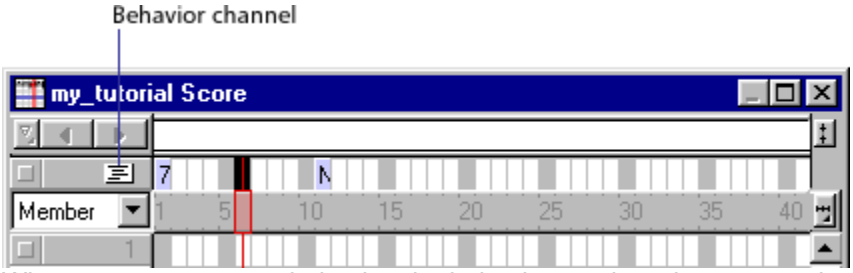
The best approach is to start simple and test your work frequently. When you get one part of a script working, start writing the next part. This approach helps you identify bugs efficiently and ensures that your Lingo is solid as you write more complex scripts.

Performing common tasks

The following are ways to perform common tasks for creating, attaching, and opening scripts.

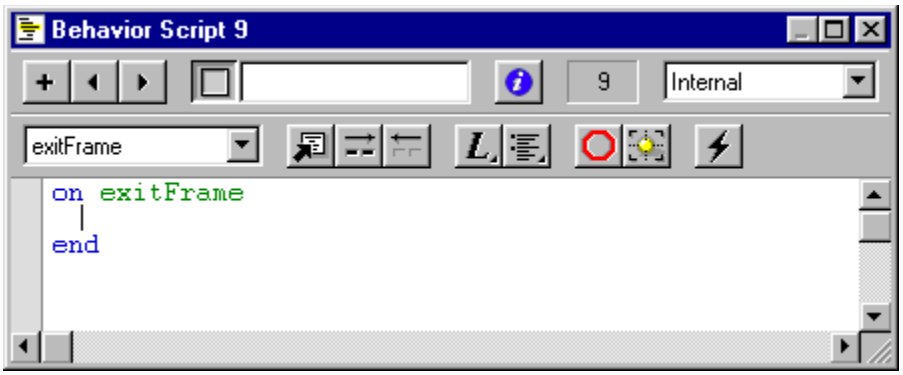
To create a frame behavior (script attached to a frame):

Double-click the behavior channel in the frame that you want to attach a behavior to.



When you create a new behavior, the behavior receives the cast number of the first available location in the current Cast window.

When you create a new frame behavior, the Script window opens and already contains the line `on exitFrame`, followed by a line with a blinking insertion point, and then a line with the word `end`. This makes it easy for you to quickly attach a common behavior to the frame.



To create a sprite behavior (script attached to a sprite):

In the Score or on the Stage, select the sprite that you're attaching the behavior to. Then choose Window > Inspectors > Behavior and choose New Behavior from the Behavior pop-up menu.

When you create a new sprite behavior, the Script window opens and already contains the line `on mouseUp`, followed by a line with a blinking insertion point, and then a line with the word `end`. This makes it easy for you to quickly attach a common behavior to the sprite.

To open a behavior for editing:

1. Double-click the behavior in the Cast window.

   The Behavior Inspector opens.

2. Click the Script Window icon in the Behavior Inspector.

   The Script window displays the behavior.

Alternatively, you can open the Script window and cycle through the scripts until you reach the behavior.

To remove a behavior from a Score location:

Select the location and then delete the script from the list displayed in the Property Inspector (Behavior tab).

To attach existing behaviors to sprites or frames, do one of the following:

- Drag a behavior from a cast to a sprite or frame in the Score or (for sprites) to a sprite on the Stage.
- In the score, select the sprites or frames that you're attaching the behavior to. Then choose Window > Inspectors > Behavior and choose the existing behavior from the Behavior pop-up menu.

To create a movie script (script attached to a movie), do one of the following:

- If the current script in the Script window is a movie script, click the New Script button in the Script window. (Clicking the New Script button always creates a script of the same type as the current script.)
- If the current script in the Script window is not a movie script, click the New Script button and then change the new script's type with the Script Type pop-up menu in the Script tab of the Property Inspector.
- If no sprites or scripts are selected in the cast, Score, or Stage, then open a new Script window; this will create a new movie script by default.

To open a movie script or parent script for editing:

Double-click the script in the Cast window.

To change a script's type:

1. Select the script in the Cast window or open it in the Script window.
2. Click the Script tab of the Property Inspector and choose a script type from the Type pop-up menu.

To cycle through the scripts in the Script window:

Use the Previous Cast Member and Next Cast Member arrows at the top of the Script window to advance or back up to a script.

To duplicate a script:

Select the script in the Cast window and choose Duplicate from the Edit menu.

To create and open scripts attached to cast members, do one of the following:

- Right-click (Windows) or Control-click (Macintosh) on a cast member in the Cast window and choose Cast Member Script from the context menu.
- Select a cast member in the Cast window and then click the Cast Member Script button.
- If the Script window is not open, choose Window > Script to open it. Then click the Previous Cast Member or Next Cast Member buttons until the script for the desired cast member appears in the window.

Note that the first two methods also let you create a new script if none is attached to the cast member.

Lingo terminology

Like any programming language, Lingo uses specific terminology and has rules of grammar and punctuation that you must follow. This information is summarized in this section.

Important Lingo terms are listed here in alphabetical order. References are included for terms that are discussed in more detail elsewhere in this chapter.

Arguments are placeholders that let you pass values to scripts (see **Using arguments to pass values to a handler**). For example, the following handler, called `addThem`, adds two values it receives in the arguments `a` and `b`.

```
on addThem a, b
  c = a + b
end
```

Commands are terms that instruct a movie to do something while the movie is playing. For example, `go to` sends the playback head to a specific frame, a marker, or another movie.

Constants are elements that don't change. For example, the constants `TAB`, `EMPTY`, and `RETURN` always have the same meaning.

Events are actions that occur while a movie is playing. For example, when a movie stops, a sprite starts, the playback head enters a frame, or the user types at the keyboard, these actions are events.

Expressions are any part of a statement that produces a value. For example, `2 + 2` is an expression.

Functions are terms that return a value. For example, the `date()` function returns the current date set in the computer. The `key()` function returns the key that was pressed last. Parentheses occur at the end of a function.

Handlers are sets of Lingo statements within a script that run when a specific event occurs in a movie (see **Using handlers**). For example, the following statements comprise a handler that plays a beep sound when the mouse is clicked:

```
on mouseDown
  beep
end
```

Keywords are reserved words that have a special meaning. For example, `end` indicates the end of a handler.

Lists are ordered sets of values used to track and update an array of data, such as a series of names or the values assigned to a set of variables (see **Using lists**). A simple example is a list of numbers such as `[1, 4, 2]`.

Messages are notices that Director sends to scripts when specific events occur in a movie (see **Using messages to identify events**). For example, when the playback head enters a specific frame, the `enterFrame` event occurs and Directors sends an `enterFrame` message. If a script contains an `on enterFrame` handler, the statements within that handler will run, because the handler received the `enterFrame` message.

Operators are terms that calculate a new value from one or more values. For example, the addition (`+`) operator adds two or more values together to produce a new value.

Properties are attributes that define an object. For example, `picture` is a property of a bitmap cast member.

Statements are valid instructions that Director can execute (see **Writing Lingo statements**). For example, `go to frame 23` is a statement.

Variables are elements used to store and update values (see **Storing and updating values in variables**). To assign values to variables or change the values of many properties, you use the equals (=) operator or the `set` command. For example, the statement `set startValue = 0` places a value of 0 into a variable named `startValue`.

Lingo syntax

Lingo supports a variety of data types, including references to sprites and cast members, `TRUE` and `FALSE` (Boolean) values, strings, constants, integers, floating-point numbers, points, rects, colors, and dates.

The following are general syntax rules that apply to all Lingo. Most Lingo terms also have their own individual requirements about terms that they must be combined with. For the rules for a specific Lingo term, see the term's syntax in the *Lingo Dictionary*.

Parentheses

Functions that return values require parentheses. When you define functions in handlers, you need to include parentheses in the calling statement.

Use parentheses after the keyword `sprite` or `member` to refer to the object's identifier: for example, `member("Patrice Lumumba")` refers to the member named Patrice Lumumba.

You can also use parentheses to override Lingo's order of precedence in math operations or to make your Lingo statements easier to read.

For example, this math expression will yield a result of 13:

```
5 * 3 - 2
```

while this expression will yield a result of 5:

```
5 * (3 - 2)
```

Character spaces

Words within expressions and statements are separated by spaces. Lingo ignores extra spaces.

In strings of characters surrounded by quotation marks, spaces are treated as characters. If you want spaces in a string, you must insert them explicitly.

You can see Lingo that uses strings in **Writing strings**.

Uppercase and lowercase letters

Lingo is not case sensitive—you can use uppercase and lowercase letters however you want. For example, the following statements are equivalent:

```
Set the hiLite of member "cat" to True
set the hilite of member "Cat" to True
SET THE HILITE OF MEMBER "CAT" TO TRUE
Set The Hilite Of Member "Cat" To True
```

However, it's a good habit to follow script writing conventions, such as the ones that are used in this book, to make it is easier to identify names of handlers, variables, and cast members when reading Lingo code.

Also, note that literal strings are case sensitive. See **Writing strings**.

Comments

Comments in scripts are preceded by double hyphens (`--`). You can place a comment on its own line or after any statement. Lingo ignores any text following the double hyphen on the same line. For more

information about comments in Lingo, see **Troubleshooting Lingo** in the Director Support Center.

Comments can consist of anything you want, such as notes about a particular script or handler or notes about a statement whose purpose might not be obvious. Comments make it easier for you or someone else to understand a procedure after you've been away from it for a while.

Double hyphens can also be used to make Lingo ignore sections of code you want to deactivate for testing or debugging purposes. By adding double hyphens rather than removing the code, you can temporarily turn it into comments. Select the code you want to turn on or off and then use the Comment or Uncomment buttons in the Script window to add or remove double hyphens easily.

Optional keywords and abbreviated commands

You can abbreviate some Lingo statements. Abbreviated versions of a command are easier to enter but may be less readable than the longer versions. The `go` command is a good example. All the following statements are equivalent, but the last one uses the fewest keystrokes.

```
go to frame "This Marker"
go to "This Marker"
go "This Marker"
```

It is good practice to use the same abbreviations throughout a movie so your Lingo is easier to read.

Describing conditions

A script often needs to determine whether a certain condition exists before carrying out a set of instructions. For example, a script may need to check whether a network operation is finished before doing something that requires the operation's result.

The term `TRUE` or the number 1 indicates that the condition you're testing for exists. The term `FALSE` or the number 0 indicates that a condition doesn't exist.

Writing Lingo statements

When you are writing statements in a Lingo script, you can choose between two types of syntax: verbose syntax and dot syntax.

Verbose syntax

Verbose syntax is similar to English. Because of this, verbose syntax is an excellent way to learn to program for the first time: as a new programmer, you can read verbose Lingo and get a fairly good idea of what it is doing. Most users will start out writing Lingo exclusively with verbose syntax because it is so easy to understand.

Here are three examples of verbose Lingo that is very English-like and has a literal meaning:

```
set the stageColor to 255
```

```
put the text of member "Instructions" after ¬
member "Introduction"
```

```
if x=5 then
  go to frame 22
end if
```

Almost all of Lingo's functionality is available through verbose syntax, but there are a few exceptions. Most of these exceptions are found in Lingo used for manipulating text.

The disadvantage of verbose Lingo is that it can quickly become very long when you write complex scripts. Longer scripts are harder to read and debug. Once your scripts reach a certain level of complexity, you may find it easier to use dot syntax.

Dot syntax contains several examples that compare verbose and dot syntax.

Dot syntax

Dot syntax is a concise form of Lingo that makes longer scripts easier to read and comprehend for users who have at least a novice understanding of the language. By understanding and using dot syntax, you can make your scripts shorter and easier to read and debug.

If you are just beginning to learn Lingo, you will probably want to start with verbose syntax and then begin using dot syntax as your understanding of Lingo improves. You can use verbose syntax and dot syntax in combination. You may want to do this as you begin the process of learning dot syntax.

Because most users will want to use dot syntax after they achieve a basic understanding of Lingo, most of the Lingo examples in this book are written with dot syntax. However, this chapter will provide extensive examples of both syntaxes.

Almost any Lingo statement can be written with either verbose syntax or dot syntax. The following example demonstrates how the two types of syntax relate to each other.

This statement sets the forecolor of sprite 12 to 155 using verbose syntax:

```
set the forecolor of sprite 12 to 155
```

The following statement does the same thing by using dot syntax. It also omits the `set` command, which is optional:

```
sprite(12).forecolor = 155
```

You can use dot syntax to express the properties or functions related to an object or to specify a chunk of text within a text object. A dot syntax expression begins with the name of the object, followed by a period (dot), and then the property, function, or text chunk that you want to specify.

For example, the `loc of sprite` property indicates a sprite's horizontal and vertical position on the Stage. The expression `sprite(15).loc` refers to the `loc of sprite` property of sprite 15.

As another example, the `number` cast member property specifies a cast member's number. The expression `member("Hot Button").number` refers to the cast member number of the Hot Button cast member.

Expressing a function related to an object follows the same pattern. For example, the `pointInHyperLink` text sprite function reports whether a specific point is within a hyperlink in a text sprite. In addition to the syntax demonstrated in the *Lingo Dictionary*, you can use the dot syntax *`textSpriteObject`*`.pointInHyperlink()` to express this function.

The following `put` statement will evaluate the specified expression and return `TRUE` or `FALSE` depending on whether the pointer is located over a hyperlink in the text sprite in channel 3:

```
put sprite(2).pointInHyperlink(mouseLoc)
```

This is how the same statement is written with verbose syntax:

```
put pointInHyperlink(sprite 2, the mouseLoc)
```

To identify chunks of text, include terms after the dot to refer to more specific items within text. For example, the expression `member("News Items").paragraph(1)` refers to the first paragraph of the text cast member News Items. The expression `member("News Items").paragraph(1).line(1)` refers to the first line in the first paragraph. These text chunk expressions are available only with dot syntax.

Using messages to identify events

To run the appropriate set of Lingo statements at the right time, Director must determine what is occurring in the movie and which Lingo to run in response to certain events.

Director sends messages to indicate when specific events occur in a movie, such as when sprites are clicked, keyboard keys are pressed, a movie starts, the playback head enters or exits a frame, or a script returns a certain result. Handlers within scripts contain instructions that run when a specific message is received.

Although you can define your own message names (see **Defining custom messages**), most common events that occur in a movie have built-in message names. See the following categories in the "Lingo by Feature" section of the *Lingo Dictionary* for the built-in messages that describe events.

- Keyboard and mouse events. See **Keyboard events** and **Mouse events**.
- Frame events. See **Frame events**.
- Browser and Internet events. See **Network Lingo**.
- Sprite events. See **Sprite events**.
- Movie in a window (MIAW) events. See **Movie in a window events**.
- Movie events. See **Movie events**.
- Synchronizing media events. See **Media synchronization**.
- Idle events. See **Memory management**.
- Timeout events. See **Time**.
- Authoring behavior events. See **Authoring behaviors**.

Defining custom messages

In addition to using built-in message names, you can define your own messages and corresponding handler names. A custom message can call another script, another handler, or the statement's own handler. When the called handler stops executing, the handler that called it resumes.

Director can send a custom message from any location. The message is first available to handlers in the script from which the message was sent. If no handler is found, the message is available to movie scripts.

If more than one movie script contains a handler for the message, the handler in the movie script that has the lowest cast member number is executed.

A custom handler name must meet the following criteria:

- It must start with a letter.
- It must include alphanumeric characters only (no special characters or punctuation).
- It must consist of one word or of several words connected by an underscore—no spaces are allowed.
- It must be different from the name of any predefined Lingo element.

Using Lingo keywords for handler names can create confusion. Although it is possible to explicitly replace or extend the functionality of a Lingo element by using it as a handler name, this should be done only in certain advanced situations.

When you have multiple handlers with similar functions, it is useful to give them names that have similar beginnings so they appear together in an alphabetical listing, such as the listing that can be displayed by the Edit > Find > Handler command.

Understanding the order of messages in a movie

Director follows a definite order when sending messages about events that occur during the course of a movie.

When the movie first starts, events occur in the following order:

1. `prepareMovie`
2. `beginSprite`

   This event occurs when the playback head enters a sprite span.

3. `prepareFrame`

   Immediately after the `prepareFrame` event, Director plays sounds, draws sprites, and performs any transitions or palette effects. This event occurs before the `enterFrame` event. An `on prepareFrame` handler is a good location for Lingo that you want to run before the frame draws.

4. `startMovie`

   This event occurs in the first frame that plays.

When Director plays a frame, events occur in this order:

1. `beginSprite`

   This event occurs only if new sprites begin in the frame.

2. `stepFrame`
3. `prepareFrame`

   Immediately after the `prepareFrame` event, Director plays sounds, draws sprites, and performs any transitions or palette effects. This event occurs before the `enterFrame` event.

4. `enterFrame`

   After `enterFrame` and before `exitFrame`, Director handles any time delays required by the tempo setting, idle events, and keyboard and mouse events.

5. `exitFrame`
6. `endSprite`

   This event occurs only if the playback head exits a sprite in the frame.

When a movie stops, events occur in this order:

1. `endSprite`

   This event occurs only if sprites currently exist in the movie.

2. `stopMovie`

Using handlers

As described in **Using messages to identify events**, Director sends messages to handlers within scripts when specific events occur. You attach a set of handlers to an object by attaching the handlers' script to the object. See **Creating and attaching scripts with the Script window**.

Each handler begins with the word `on` followed by the message that the handler should respond to. The last line of the handler is the word `end`. You can repeat the handler's name after `end`, but this is optional.

When an object receives a message that corresponds to a handler attached to the object, Director runs the Lingo statements within the handler. For example, the `mouseDown` message indicates that the user clicked the mouse button. To indicate in your script that an action should be performed when the mouse is clicked, you include a line in your script that begins with `onMouseDown`. You follow this line with the Lingo statements that should execute when the script receives the `mouseDown` message.

Using arguments to pass values to a handler

By using arguments for values, you can give the handler exactly the values that it needs to use at a specific time, regardless of where or when you call the handler in the movie. Arguments can be optional or required, depending on the situation.

To create arguments for a handler:

Put the arguments after the handler name. Use commas to separate multiple arguments.

For example, the following handler, called `addThem`, adds two values it receives in the arguments `a` and `b`, stores the result in local variable `c`, and uses the Lingo term `return` to send the result back to the original handler:

```
on addThem a, b
  -- a and b are argument placeholders
  c = a + b
  return c
end
```

When you call a handler, you must provide specific values for the arguments that the handler uses. You can use any type of value, such as a number, a variable that has a value assigned, or a string of characters. Values in the calling statement must be in the order they follow in the handler's arguments, and they must be surrounded by parentheses.

The following statement is a calling statement for the `on addThem` handler:

```
set mySum = addThem(4, 8)
```

Because `4` is first in the list of arguments, Lingo substitutes it for `a` in the handler. Likewise, because `8` is second in the list of arguments, Lingo substitutes `8` for `b` everywhere in the handler.

After the calling statement sends these parameters to the handler, the handler returns the value `12`, which corresponds to the variable `c` inside the `on addThem` handler. The variable `mySum` in the calling statement is then set to `12`.

You can also use expressions as values. For example, the following statement substitutes `3+6` for `a` and `8>2` (or `1`, representing `TRUE`) for `b`, and would return `10`:

```
set mySum = addThem(3+6, 8>2)
```

Returning results from handlers

Often you want a handler to report some condition or the result of some action.

To return results from a handler:

Use the `return` function to have a handler report a condition or the result of an action. For example, the following handler returns the current color of sprite 1:

```
on findColor
  return sprite(1).foreColor
end
```

When you define a handler that returns a result, you must use parentheses after the handler when you call it from another handler. For example, the statement `put findColor()` calls the `on findColor` handler and then displays the result in the Message window.

Deciding where to place handlers

You can place handlers in any type of script, and a script can contain multiple handlers. It's a good idea to group related handlers in a single place, though, for easier maintenance.

The following are some useful guidelines for many common situations:

- To set up a handler that affects a specific sprite or runs in response to an action on a specific sprite, put the handler in a behavior attached to the sprite. To set up a handler that should be available any time the movie is in a specific frame, put the handler in a behavior attached to the frame.

  For example, to have a handler respond to a mouse click while the playback head is in a frame, regardless of where the click occurs, place an `on mouseDown` or `on mouseUp` handler in the frame behavior rather than in a sprite behavior.

- To set up a handler that runs in response to messages about events anywhere in the movie, put the handler in a movie script.
- To set up a handler that runs in response to an event that affects a cast member, regardless of which sprites use the cast member, put the handler in a cast member script.

Determining when handlers receive a message

A movie can contain more than one handler for the same message. Director manages this situation by sending the message to objects in a definite order.

The general order in which messages are sent to objects is as follows:

1. Messages are sent first to behaviors attached to a sprite involved in the event. If a sprite has more than one behavior attached to it, behaviors respond to the message in the order in which they are attached to the sprite.
2. Messages are sent next to a script attached to the cast member assigned to the sprite.
3. Messages are then sent to behaviors attached to the current frame.
4. Messages are sent last to movie scripts.

When a message reaches a script that contains a handler corresponding to the message, Director executes the handler's instructions.

After a handler intercepts a message, the message doesn't automatically pass on to the remaining locations. (You can use the `pass` command to override this default rule and pass the message to other objects.) If no matching handler is found after the message passes to all possible locations, Director ignores the message.

The exact order of objects to which Director sends a message depends on the message. For details about the sequence of objects to which Director sends specific messages, see the *Lingo Dictionary* entry for the message.

Using lists

Lists provide an efficient way to track and update an array of data, such as a series of names or the values assigned to a set of variables. For example, if you know you will need to keep track of many names or numbers in your Director project, you may want to store them in a list. The list operator (`[ ]`) designates that the items within the brackets comprise a list.

You can create two types of lists with Lingo: linear lists and property lists.

- In a linear list, each element is a single value. For example, this linear list is a simple set of numbers:

```
[100, 150, 300, 350]
```

- In a property list, each element contains two values separated by a colon. One value is a property name, always preceded by a pound (#) sign; the other value is the value associated with that property. For example, the following statement sets the variable `myList` to a property list containing values for the properties #speed, #direction, and #weight. These could be the properties of an asteroid.

```
myList = [#speed: 155, #direction: 237, #weight: 8746]
```

  Properties can appear more than once in a property list.

Both kinds of lists can be empty, containing no values at all. An empty linear list consists of two square brackets (`[ ]`). An empty property list consists of two square brackets surrounding a colon (`[:]`).

It's usually easier to manipulate a list by assigning it to a variable when you create the list. The value contained in the variable is actually a reference to the list, not the list itself.

For more information on lists, see **list()**.

Creating linear lists

The most common way to create a linear list is to use the list operator (`[ ]`). You can also use the `list()` function to create a linear list.

To create a linear list, do one of the following:

- Place the list elements within the list operator (`[ ]`).
- Specify the list's elements as parameters of the `list()` function. (This function is useful when you use a keyboard that doesn't provide square brackets.)

In both cases, you use commas to separate items in the list.

For example, the following statements have the same effect; each statement creates a linear list of three names:

```
set workerList = ["Bruno", "Heather", "Carlos"]
set workerList = list("Bruno", "Heather", "Carlos")
```

To create an empty linear list:

Set the list to `[ ]`.

Creating property lists

The only way to create a property list is to use the list operator (`[ ]`). You cannot use the `list()` function to create a property list.

To create a property list:

Place the list elements within the list operator, and use commas to separate the elements. Precede each property with the pound (#) sign, and separate each property from its value with a colon.

For example, the following statements create two different property lists. Each list specifies the Stage coordinates of a sprite.

```
sprite1Location = [#left:100, #top:150, #right:300, #bottom:350]
sprite2Location = [#left:400, #top:550, #right:500, #bottom:750]
```

To create an empty property list:

Set the list to `[:]`.

Setting and retrieving items in a list

Lingo lets you set and retrieve individual items in a list. The syntax differs for linear and property lists.

To set a value in a linear list:

Use the equals (=) operator. (You can also use the `setAt` command introduced in earlier versions of Director.)

For example, the statement `workerList[2] = "Tiffany"` makes Tiffany the new value for the second item in the list workerList.

To retrieve a value in a linear list:

Use the list variable followed by the number that indicates the value's position in the list. Place square brackets around the number. (You can also use the `getAt` or `getaProp` commands, which were introduced in earlier versions of Director.)

For example, in the linear list `set workerList = ["Bruno ", "Heather ", "Carlos "]`, the expression `workerList[2]` represents the second value in the list workerList. The value is Heather.

To set a value in a property list, do one of the following:

- Use the equals (=) operator. (You can also use the `setAProp` command, which was introduced in earlier versions of Director.)

  For example, the statement `foodList[#Bruno] = "sushi"` makes sushi the new value associated with the property Bruno.

- Use dot syntax.

  For example, the statement `foodList.Bruno = "sushi"` makes sushi the new value associated with the property Bruno in the list foodList.

To retrieve a value in a property list, do one of the following:

- Use the list variable followed by the name of the property associated with the value. Place square brackets around the property. (You can also use the `getaProp` or `getAt` commands, or the `getPropAt()` function, which were introduced in earlier versions of Director.)

  For example, in the property list `foodList = [#breakfast:"Waffles", #lunch:"Tofu Burger", #dinner:"Hungarian Goulash"]`, the expression `foodList[#breakfast]` represents the value associated with the property `#breakfast`. The value is Waffles.

- Use dot syntax.

  For example, using the `foodList` property list above, `foodList.breakfast` represents the value Waffles.

Checking items in a list

You can determine the characteristics of a list and the number of items the list contains by using the following commands and functions.

- To display the contents of a list, use the `put` command followed by the variable that contains the list. See **put**.
- To determine the number of items in a list, use the `count()` function. See **count()**.
- To determine a list's type, use the `ilk()` function. See **ilk()**.
- To determine the maximum value in a list, use the `max()` function. See **max()**.
- To determine the minimum value in a list, use the `min()` function. See **min**.
- To determine the position of a specific property, use the `findPos`, `findPosNear`, or `getOne` command. See **findPos**, **findPosNear**, and **getOne()**.

Adding and deleting items in a list

You can add or delete items in a list by using the following commands.

- To add an item at the end of a list, use the `append` command. See **append**.
- To add an item at its proper position in a sorted list, use the `add` or `addProp` command. See **add** and **addProp**.
- To add an item at a specific place in a linear list, use the `addAt` command. See **addAt**.
- To add an item at a specific position in a property list, use the `addProp` command. See **addProp**.
- To delete an item from a list, use the deleteAt, deleteOne, or deleteProp command. See **deleteAt**, **deleteOne**, and **deleteProp**.
- To replace an item in a list, use the `setAt` or `setaProp` command. See **setAt** and **setProp**.

You do not have to explicitly remove lists. Lists are automatically removed when they are no longer referred to by any variable.

Copying lists

Assigning a list to a variable and then assigning that variable to a second variable does not make a separate copy of the list. For example, the statement `landList = ["Asia", "Africa"]` creates a list that contains the names of two continents. The statement `continentList = landList` assigns the same list to the variable `continentList`. However, adding Australia to `landList` using the statement `add landList, "Australia"` automatically adds Australia to `continentList` also. This happens because both variable names point to the same object in memory.

To create a copy of a list that is independent of the first list:

Use the duplicate() function. See **duplicate() (list function)**.

For example, this statement creates a list and assigns it to the variable `oldList`:

```
oldList = ["a", "b", "c"]
```

This statement uses the `duplicate()` function to make an independent copy of the list and assign it to the variable `newList`:

```
newList = duplicate(oldList)
```

After `newList` is created, editing either `oldList` or `newList` has no effect on the other.

Sorting lists

Lists can be unsorted. However, Lingo can sort a list in alphanumeric order, with numbers before strings. Strings are sorted according to their initial letters, regardless of how many characters they contain.

Lingo sorts a linear list according to the values in the list. Lingo sorts a property list according to the properties in the list.

To sort a list:

Use the `sort` command followed by the list's name. See **sort**.

About variables

Director uses variables to store and update values. As the name implies, a variable contains a value that can be changed or updated as the movie plays. By changing the value of a variable as the movie plays, you can do things such as store a URL, track the number of times a user takes part in an online chat session, or record whether a network operation is complete.

It's a good idea always to assign a variable a known value the first time you define the variable. This is known as initializing a variable. Initializing variables makes it easier to track and compare the variable's value as the movie plays.

Variables can be global or local. A global variable can exist and retain its value as long as Director is running, while a local variable exists only as long as the handler in which it is defined is running. See **Using global variables** and **Using local variables**.

Storing and updating values in variables

Variables can hold any of the types of information found in Director: numbers, strings, `TRUE` or `FALSE` values, symbols, lists, or the result of a calculation. To store and retrieve the values of properties and variables, Lingo uses the equals (=) operator and the `set` and `put` commands.

Also, a variable in Lingo can contain different types of data at different times. (The ability to change a variable's type distinguishes Lingo from other languages such as Java, in which a variable's type cannot be changed.)

For example, the statement `set x = 1` creates the variable `x`, which is an integer variable because you assigned the variable an integer. If you subsequently use the statement `set x = "one"`, the variable `x` becomes a string variable, because the variable now contains a string.

Some properties cannot be set, but can only be tested. Often these are properties that describe some condition that exists outside Director's control. For example, you cannot assign a value to the `numChannels` cast member property, which indicates the number of channels within a Shockwave movie. However, you can retrieve the number of channels by referring to the `numChannels` property of a cast member.

To assign a value to a variable:

Use the equals (=) operator. For improved readability, you can use the optional `set` command at the beginning of the statement.

For example, any of these statements will change the cast member assigned to sprite 2 by setting the sprite's `member` property to a different cast member. The last two statements use dot syntax (see **Dot syntax**):

```
set the member of sprite 2 = member "Big Flash"
set sprite (2).member = member ("Big Flash")
sprite (2).member = member ("Big Flash")
```

As another example, each of these statements assigns a URL to the variable `placesToGo`:

```
placesToGo = "http://www.macromedia.com"
set placesToGo = "http://www.macromedia.com"
```

Variables can also hold the results of mathematical operations. Both of these statements add the result of an addition operation to the variable `mySum`:

```
mySum = 5 + 5
set mySum = 5 + 5
```

It's good practice to use variable names that indicate what the variable is used for. This will make your Lingo easier to read. For example, the variable `mySum` indicates that the variable contains the sum of numbers.

To test the values of properties or variables:

Use the `put` command in the Message window or check the values in the Watcher window.

For example, the statement `put myNumber` displays the value assigned to the variable `myNumber` in the Message window.

As another example, the following statement returns the cast member assigned to sprite 2 by retrieving the sprite's `member` property:

```
put the member of sprite 2
```

Using global variables

Global variables can be shared among handlers and movies. A global variable exists and retains its value as long as Director is running or until you issue the `clearGlobals` command.

In Shockwave, global variables persist among movies displayed by the Lingo `goToNetMovie` command, but not among those displayed by the `goToNetPage` command.

Every handler that declares a variable as global can use the variable's current value. If the handler changes the variable's value, the new value is available to every other handler that treats the variable as global.

It's good practice to start the names of all global variables with a lowercase *g*. This helps identify which variables are global when you examine Lingo code.

Because you usually want global variables to be available throughout a movie, it is good practice to declare global variables in an `on prepareMovie` handler. This ensures that the global variables are available from the very start of the movie.

To declare that a variable is global, do one of the following:

- Use the term `global` before the variable name at the top of the Script window, before any individual handlers. This makes the variable global for every handler in the script.
- Declare the variable as global by using the term `global` before the variable name on a separate line in every handler that uses the global variable.

When you use the term `global` to define global variables, the variables automatically have `VOID` as their initial value.

The following statements make `gName` a global variable and give it the value Mary:

```
global gName
gName = "Mary"
```

To display all current global variables and their current values:

Use the `showGlobals` command in the Message window.

To clear all current global variables:

Use the `clearGlobals` command in the Message window to set the value of all global variables to `VOID`.

See **clearGlobals** and **showGlobals**.

Using local variables

A local variable exists only as long as the handler in which it is defined is running. However, after a local variable is created, you can use the variable in other expressions or change its value while Lingo is still within the handler that defined the variable.

Treating variables as local is a good idea when you want to use a variable only temporarily in one handler. This helps you avoid unintentionally changing the value in another handler that uses the same variable name.

To create a local variable:

Assign the variable a value using the equals (=) operator or the `set...=` command.

Unless the handler uses the term `global` to declare that a variable is global, the variable is automatically a local variable.

To display all current local variables in the handler:

Use the `showLocals` command.

You can use this command in the Message window or in handlers to help with debugging. The result appears in the Message window. The Director debugger can also track the value of local variables. For more information about using the debugger, see **Troubleshooting Lingo**, in the Director Support Center.

Expressing literal values

A literal value is any part of a statement or expression that is to be used exactly as it is, rather than as a variable or a Lingo element. Literal values that you encounter in Lingo are character strings, integers, decimal numbers, cast member names and numbers, frame and movie names and numbers, symbols, and constants.

Note: The `value()` function can convert a string into a numerical value. The `string()` function can convert a numerical value into a string.

Each type of literal value has its own rules.

Writing strings

Strings are characters that Lingo treats as characters instead of as variables. Strings must be enclosed in double quotation marks. For example, in the statement

```
member ("Greeting").text = "Hello"
```

“Hello” and “Greeting” are both strings. “Hello” is the literal text being put into a text cast member; “Greeting” is the name of the cast member.

Similarly, if you test a string, double quotation marks must surround each string, as in the following example:

```
if "Hello Mr. Jones" contains "Hello" then soundHandler
```

Lingo treats spaces at the beginning or end of a string as a literal part of the string. The following expression includes a space after the word *to*:

```
put "My thoughts amount to "
```

Although Lingo does not distinguish between uppercase and lowercase when referring to cast members, variables, and so on, literal strings are case sensitive. For example, the following two statements place different text into the specified cast member, because `"Hello"` and `"HELLO"` are literal strings.

```
member ("Greeting").text = "Hello"
member ("Greeting").text = "HELLO"
```

Using integers

An integer is a whole number, without any fractions or decimal places.

Director works with integers between -2,147,483,648 and +2,147,483,647. (For numbers outside this range, use decimal numbers, sometimes called floating-point numbers.) Enter integers without using commas. Use a minus (`-`) sign for negative numbers.

You can convert a decimal number to an integer by using the `integer()` function. For example, the statement `set theNumber = integer(3.9)` rounds off the decimal number 3.9 and converts it to the integer 4.

Some Lingo commands and functions require integers for their parameters. The requirements for specific Lingo elements can be found in Director Help or the *Lingo Dictionary*.

Using decimal numbers

A decimal number, sometimes called a floating-point number, is any number that includes a decimal point. The `floatPrecision` property controls the number of decimal places used to display these numbers. (However, Director always uses the complete number in calculations.) See the `floatPrecision` entry in Director Help or the *Lingo Dictionary* for information about setting the number of decimal places used for decimal numbers.

You can also use exponential notation with decimal numbers: for example, `-1.1234e-100` or `123.4e+9`.

You can convert an integer or string to a decimal number by using the `float()` function. For example, the statement `set theNumber = float(3)` stores the value `3.0` in the variable.

Identifying cast members and casts

Note: If you rearrange (and thus renumber) cast members while creating a movie, Director doesn't automatically update references to cast member numbers in Lingo scripts. Therefore, although some of the examples in this section illustrate how to reference cast members by number, the best practice is to always name cast members and refer to them by name in Lingo scripts.

Lingo refers to a cast member by using the term `member` followed by a cast member name or number in parentheses. (Cast member names are strings and follow the same syntax rules as other strings.) An alternative syntax is the term `member`, without parentheses, followed by the cast member name or number.

For example, the following all refer to cast member 50, which has the name Hammer:

```
member("Hammer")
member(50)
member "Hammer"
member 50
```

If more than one cast contains a cast member with the same name, you must use a second parameter to specify the cast member's cast. When your movie uses more than one cast and you identify a cast member by its number, you must also specify the cast. Otherwise, the second parameter is optional.

To specify a cast without parentheses when using `member`, include the term `of castLib` followed by the cast's name or number. When the cast member's name is unique in the movie, the cast's name or number isn't required, but you can include it for clarity.

For example, the following statements refer to cast member 50, which is named Hammer, in castLib 4, which is named Tools:

```
member(50, 4)
member 50 of castLib 4
member("Hammer", 4)
member "Hammer" of castLib 4
member(50, "Tools")
member 50 of castLib "Tools"
member("Hammer", "Tools")
member "Hammer" of castLib "Tools"
```

If more than one cast member has the same name and you use the name in a script without specifying the cast or cast member number, Lingo uses the first (lowest numbered) cast member in the lowest numbered cast that has the specified name.

Identifying frames and movies

Use these Lingo terms to refer to frames in a movie:

- The function `the frame` refers to the current frame.
- The keyword `frame` followed by the frame number or the frame marker label refers to a specific frame. For example, `frame 60` indicates frame 60.
- The keyword `loop` refers to the marker at the beginning of the current segment. If the current frame has a marker, `loop` refers to the current frame; if not, `loop` refers to the first marker before the current frame. If there are no markers in the movie, `loop` refers to the first frame.
- The word `next` or `previous` refers to the next marker or the marker before the current scene, respectively.
- The term `the frame` followed by a minus or plus sign and the number of frames before or after the current frame refers to a frame that's a specific number of frames before or after the current frame. For example, `the frame - 20` refers to the frame 20 frames before the current frame.
- The term `the frameLabel` identifies the label assigned to the current frame.
- The function `marker()`, with a positive or negative number of markers used as the parameter, refers to the marker that's a specific number of markers before or after the current frame. For example, `marker(-1)` returns the frame number of the previous marker and `marker(2)` returns the frame number of the second marker after the current frame. If the frame is marked, `marker(0)` returns the frame number of the current frame; if not, `marker(0)` gives the frame number of the previous marker.
- The term `movie` followed by the movie name refers to the beginning of another movie. For example, `movie "Navigation"` refers to the beginning of the movie called Navigation.
- The word `frame` plus a frame identifier, the word `of`, the word `movie`, and the movie name refers to a specific frame in another movie; for example, `frame 15 of movie "Navigation"` refers to frame 15 of the movie called Navigation.

Using symbols

A symbol is a string or other value that begins with the pound (`#`) sign.

Symbols are user-defined constants. Comparisons using symbols can usually be performed very quickly, providing more efficient code.

For example, the statement

```
userLevel = #novice
```

runs more quickly than the statement

```
userLevel = "novice"
```

Symbols can't contain spaces or punctuation.

Convert a string to a symbol by using the `symbol()` function. Convert a symbol back to a string by using the `string()` function.

See **# (symbol)** operator and **string()**function.

Expressing constants

A constant is a named value whose content never changes. For example, `TRUE`, `FALSE`, `VOID`, and `EMPTY` are constants because their values are always the same.

The constants `BACKSPACE`, `ENTER`, `QUOTE`, `RETURN`, `SPACE`, and `TAB` refer to keyboard characters. For example, to test whether the user is pressing the Enter key, use the following statement:

```
if the key = ENTER then beep
```

Using operators to manipulate values

Operators are elements that tell Lingo how to combine, compare, or modify the values of an expression. They include the following:

- Arithmetic operators (such as `+`, `-`, `/`, and `*`)
- Comparison operators (for example, `<>`, `>`, and `>=`), which compare two arguments
- Logical operators (`not`, `and`, `or`), which combine simple conditions into compound ones
- String operators (`&` and `&&`), which join strings of characters

Understanding operator precedence

When two or more operators are used in the same statement, some operators take precedence over others in a precise hierarchy that Lingo follows to determine which operators to execute first. This is called the operators' precedence order. For example, multiplication is always performed before addition. However, items in parentheses take precedence over multiplication. For example, without parentheses, Lingo performs the multiplication in this statement first:

```
total = 2 + 4 * 3
```

The result is 14.

When parentheses surround the addition operation, Lingo performs the addition first:

```
total = (2 + 4) * 3
```

The result is 18.

Descriptions of the operators and their precedence order follow. Operators with higher precedence are performed first. For example, an operator whose precedence order is 5 is performed before an operator whose precedence order is 4. Operations that have the same order of precedence are performed left to right.

Arithmetic operators

Arithmetic operators add, subtract, multiply, divide, and perform other arithmetic operations. Parentheses and the minus sign are arithmetic operators.

Operator	Effect	Precedence
( )	Groups operations to control precedence order.	5
-	When placed before a number, reverses the sign of a number.	5
*	Performs multiplication.	4
mod	Performs modulo operations.	4
/	Performs division.	4
+	Performs addition.	3
-	When placed between two numbers, performs subtraction.	3

Note: When only integers are used in an operation, the result is an integer. Using integers and floating-point numbers in the same calculation results in a floating-point number.

When dividing one integer by another doesn't result in a whole number, Lingo rounds the result down to the nearest integer. For example, the result of `4/3` is `1`.

To force Lingo to calculate a value without rounding the result, use `float()` on one or more values in an expression. For example, the result of `4/ float(3)` is 1.333.

Comparison operators

Comparison operators compare two values and determine whether the comparison is true or false. These are the comparison operators available in Lingo:

Operator	Meaning	Precedence
<	Is less than	1
<=	Is less than or equal to	1
<>	Is not equal to	1
>	Is greater than	1
>=	Is greater than or equal to	1
=	Equals	1

Logical operators

Logical operators test whether two logical expressions are true or false. These are the logical operators available in Lingo:

Operator	Effect	Precedence
and	Determines whether both expressions are true.	4
or	Determines whether either or both expressions are true.	4
not	Negates an expression.	5

The `not` operator is useful for toggling a `TRUE` or `FALSE` value to its opposite. For example, the following statement turns on the sound if it's currently off and turns off the sound if it's currently on:

```
set the soundEnabled = not (the soundEnabled)
```

String operators

String operators combine and define strings. These are the string operators available in Lingo:

Operator	Effect	Precedence
&	Concatenates two strings.	2
&&	Concatenates two strings and inserts a space between the two.	2
"	Marks the beginning or end of a string.	1

Controlling flow in scripts

Lingo uses `if...then...else`, `case`, and `repeat` statements to perform an action depending on whether a condition exists. See **Using if statements**, **Using case statements**, and **Repeating an action**.

Using if statements

Statements that check whether a condition is true or false begin with the Lingo term `if`. If the condition exists, Lingo executes the statement that follows `then`. If the condition doesn't exist, Lingo skips to the next statement in the handler.

To optimize your script's performance, test for the most likely conditions first.

The following statements test several conditions. The term `else if` specifies alternative tests to perform if previous conditions are false:

```
if the mouseMember = memberNum("map 1") then
  go to "Cairo"
else if the mouseMember = member ("map 2") then
  go to "Nairobi"
else
  alert "You're lost."
end if
```

When writing `if…then` structures, you can place the statement following `then` in the same line as `then`, or you can place it on its own line by inserting a carriage return after `then`. If you insert a carriage return, you must also include an `end if` statement at the end of the `if…then` structure.

For example, the following statements are equivalent:

```
if the mouseMember = member("map 1") then go to "Cairo"
```

```
if the mouseMember = member("map 1") then
  go to "Cairo"
end if
```

For more information, see **if** and **case** in.

Using case statements

The `case` statement is a shorthand alternative to repeating `if…then` statements when setting up a multiple branching structure. A `case` statement is often more efficient and easier to read than a large number of `if...then...else` statements.

The condition to test for follows the term `case` in the first line of the `case` structure. The comparison goes through each line in order until Lingo encounters an expression that matches the test condition. When a matching expression is found, Director executes the Lingo that follows the matching expression.

For example, the following `case` statement tests which key the user pressed most recently and responds accordingly:

```
case (the key) of
  "A": go to frame "Apple"
  "B", "C":
     puppetTransition 99
     go to frame "Oranges"
  otherwise beep
end case
```

- If the user pressed A, the movie goes to the frame labeled Apple.
- If the user pressed B or C, the movie performs the specified transition and then goes to the frame labeled Oranges.
- If the user pressed any other letter key, the computer beeps.

A `case` statement can use comparisons as the test condition.

For more information, see see **if** and **case**.

Repeating an action

Lingo can repeat an action a specified number of times or while a specific condition exists.

To repeat an action a specified number of times:

Use a `repeat with` structure. Specify the number of times to repeat as a range following `repeat with`.

This structure is useful for performing the same operation on a series of objects. For example, the following repeat loop makes Background Transparent the ink for sprites 2 through 10:

```
repeat with n = 2 to 10
  set the ink of sprite n = 36
end repeat
```

This example performs exactly the same action as above, but uses dot syntax:

```
repeat with n =2 to 10
  sprite(n).ink = 36
end repeat
```

To repeat a set of instructions as long as a specific condition exists:

Use a `repeat...while` statement.

For example, these statements instruct a movie to beep continuously whenever the mouse button is being pressed:

```
repeat while the mouseDown
  beep
end repeat
```

Lingo continues to loop through the statements inside the repeat loop until the condition is no longer true or until one of the instructions sends Lingo outside the loop. In the previous example, Lingo exits the repeat loop when the mouse button is released because the `mouseDown` condition is no longer true.

To exit a repeat loop:

Use the `exit repeat` command.

For example, the following statements make a movie beep while the mouse button is pressed, unless the mouse pointer is over sprite 1. If the pointer is over sprite 1, Lingo exits the repeat loop and stops beeping. (The term `rollover` followed by a sprite number indicates that the pointer is over the specified sprite.)

```
repeat while the stillDown
  beep
  if rollover (1) then exit repeat
end repeat
```

See **repeat with**, **repeat while**, and **exit repeat**.

Creating and attaching scripts with the Script window

To create scripts and write the Lingo statements that make up handlers, you use the Script window.

To open the Script window, do one of the following:

- Choose Window > Script.
- Double-click a script in a Cast window.

For more ways to create and open scripts, see **Performing common tasks**.

You can change the font of text in the Script window and define different colors for various code components. See **Setting Script window preferences**.

Setting Script window preferences

To change the default font of text in the Script window and the color of various code elements, you use Script window preferences. Director automatically colors different types of code elements unless you turn off Auto Coloring.

To set Script window preferences:

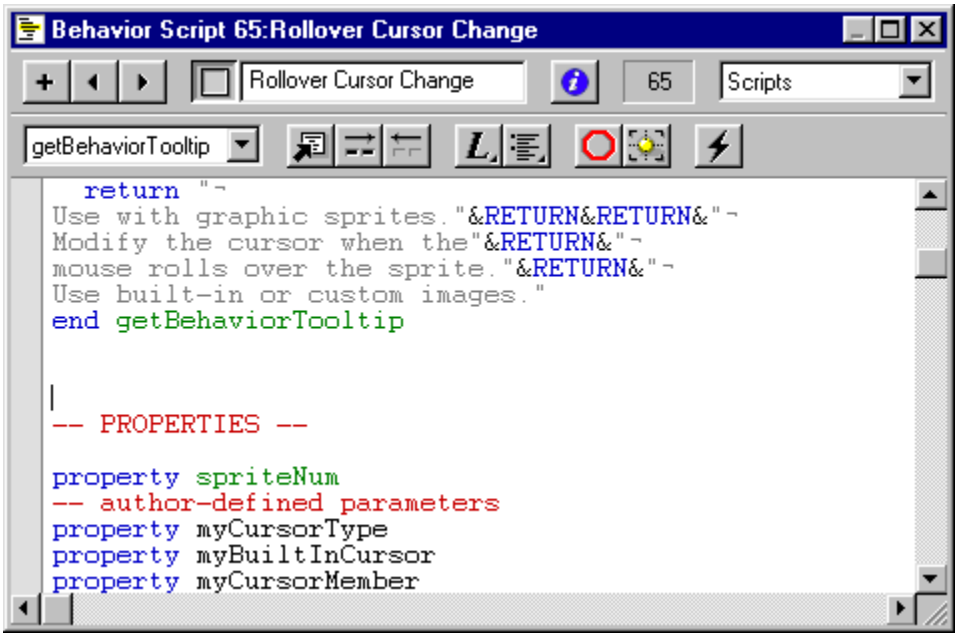
1. Choose File > Preferences > Script.
2. To choose the default font, click the Font button and choose settings from the Font dialog box.
3. To choose the default color of text in the Script window, choose a color from the Color menu.
4. To choose the background color for the Script window, choose a color from the Background color menu.
5. To make the Script window automatically color certain code elements, select Enable for Auto Coloring. This option is on by default.

   With Auto Coloring off, all text appears in the default color.
6. If Auto Coloring is on, choose colors for the following code elements from the corresponding color menus:

- Keywords
- Comments
- Literals
- Custom (terms you define in your own code)

Inserting common Lingo terms

The Script window provides a pop-up menu of common Lingo terms that you can use to insert Lingo in a script.



- The Alphabetical menu lists every element in alphabetical order.

- The Categories menu lists categories of elements according to the features they are often used for.

When you choose an element from the Lingo pop-up menu, Director inserts the element at the insertion point in the Script window.

When an element requires additional parameters, Lingo includes placeholder names that indicate the additional required information. When more than one argument or parameter is required, Lingo highlights the first one for you, so all you have to do is type to replace it. You must select and change the other parameters yourself.

Some cast member types and scripting Xtras provide Lingo terms that do not appear in the Lingo menus. These member types and Xtras often have their own documentation, and you can find some information from within Director.

To display a list of available Xtras:

Issue the command showXlib in the Message window.

To display a list of methods for an Xtra:

Issue the command put mmessageList("*XtraName*") in the Message window.

Entering and editing text

Entering and editing text in a Script window is similar to entering and editing text in any other field.

The following are common editing tasks you perform in the Script window:

- To select a word, double-click the word.
- To select an entire script, choose Select All from the Edit menu.
- To start a new line, enter a carriage return.
- To wrap a long line of code with a continuation symbol, press Alt+Enter (Windows) or Option+Return (Macintosh) where you want to insert a soft line break.

  The continuation symbol (¬) that appears indicates that the statement continues on the next line.

- To locate a handler in the current script, choose the handler's name from the Handler pop-up menu in the Script window.
- To compile the Lingo you have written, click the Script window's Recompile button or close the Script window.
- To reformat a script, press Tab in the Script window.

  Lingo automatically indents statements when the syntax is correct. If a line doesn't indent properly, there is a problem in the Lingo syntax on that line.

Finding handlers and text in scripts

The Find command in the Edit menu is useful for finding handlers and for finding and editing text and handlers.

To find handlers in scripts:

1. Choose Edit > Find > Handler.

   The leftmost column in the Find Handler dialog box displays the name of each handler in the movie. The middle column displays the number of the cast member associated with the handler's script. The rightmost column lists the cast that the cast member is in.

2. Select the handler that you want to find.

3. Click Find.

   The handler appears in the Script window.

   The title bar at the top of the Script window indicates the script's type.

To find text in scripts:

1. Make the Script window active.

2. Choose Edit > Find > Text.

   The Find Text dialog box appears.

3. Enter text that you want to find in the Find field, and then click Find.

   Find is not case sensitive: `ThisHandler`, `thisHandler`, and `THISHANDLER` are all the same for search purposes.

To specify which cast members to search:

Select the appropriate option under Search: Scripts.

To start the search over from the beginning after the search reaches the end:

Select the Wrap-Around option.

To search only for whole words and not fragments of other words that match the word:

Select the Whole Words Only option.

To find the next occurrence of the text specified in the Find field:

Choose Edit > Find Again.

To find occurrences of selected text:

1. Select the text.
2. Choose Edit > Find > Selection.

Using linked scripts

In addition to scripts stored as internal cast members, you can choose to keep scripts in external text files and link them to your Director movie. These linked scripts are similar to linked image or digital video files you can import into Director movies.

Advantages of using linked scripts include the following:

- One person can work on the Director file while another works on the script.
- You can easily exchange scripts with others.
- You can control the scripts separately from the Director file in a source code control application such as Microsoft Visual SourceSafe. Applications such as this prevent multiple programmers working on the same Director project from overwriting each other's work.

Linked scripts are used by Director only during authoring. At run time, Director projectors and Shockwave use a special internal copy of the script data stored in the movie. This way, your linked scripts need not be distributed with your movies and cannot be copied by end users.

To import a script as a linked text file:

1. Choose File > Import.
2. Choose Script as the type of file to import.
3. Select the script file(s) you want to import.

   You can import files with the file extensions .txt or .ls; the .ls extension is Director's linked script extension.

   To create a list of files you want to import, you can use the Add and Add All buttons. This is especially useful if you want to import scripts from multiple locations.
4. Select Link to External File from the Media pop-up menu.
5. Click Import.

You can edit linked scripts normally in Director's script window. Changes you make are written to the external files each time you save your Director movie. (If you imported the linked script from a UNIX server, UNIX line endings are preserved.) If you import a script whose text file is locked, you will not be able to edit the script in Director.

You cannot apply custom text colors to linked scripts in the script window. Script auto coloring, however, is enabled for linked scripts.

To turn an internal script cast member into an external, linked script cast member:

1. Select the internal cast member and click the Script tab of the Property Inspector.
2. Click Link Script As.
3. Enter a name for the script file in the Save As dialog box.
4. Click Save.

To reload a linked script after it is edited:

Use the `unloadMember` command.

If a linked script is edited outside of Director, you can reload it by using the `unloadMember` command

in the Message window. The following statement will cause the script `myScript` to be unloaded and then reloaded:

```
unloadMember member "myScript"
```

Color, tempo, and transitions: Overview

A number of behind-the-scenes functions in Director are important to the appearance and performance of a movie.

To control the way Director manages colors, it's important to understand the difference between RGB and index color and how to assign colors to various elements in your movie. See **Controlling color**.

To control the speed at which your movie plays, you use settings in the tempo channel. See **About tempo**.

To make scenes in your movie flow together without creating the animation yourself, you can use predefined transitions. See **Using transitions**.

All of these features involve using the channels at the top of the Score.

Controlling color

Choosing colors for movie elements is as simple as making a selection from a menu. To make sure that the colors you choose are displayed correctly on as many systems as possible, it helps to understand how Director controls color.

Director provides a variety of color controls. The following list describes the most important:

- Use the Movie tab in the Property Inspector to change modes for selecting colors. The modes are RGB values or palette index.
- Also use the Movie tab to turn on the Remap Palettes If Needed option, which causes Director to either dither or remap colors in bitmap images to the best available colors. If the option is off, Director assumes that all bitmaps use the movie's color palette and does not perform remapping or dithering, regardless of the settings for individual cast members. See **Setting Stage and movie properties**.
- Use the pop-up Color menu to choose colors for movie elements. The Color menu is available throughout the Director application—for example, in the Tool palette.
- Use Transform Bitmap to remap bitmap images to new palettes and change their color depth. You can also make the same changes when you import a bitmap. See **Changing size, color depth, and color palette for bitmaps**, and **About importing bitmaps**.
- Use the Score's palette channel to change the movie's color palette as a movie plays.
- Use the Color Palettes window to change the colors in a color palette or to create a custom color palette cast member.

Specifying palette index and RGB color

Director can use either palette index values or RGB values to specify colors. RGB values are much more reliable and accurate for specifying colors than palette index values. RGB is the system that most Web pages use.

Director identifies a palette index color by the number of its position in a set of colors called a color palette. Color number 12, for example, might be blue. If a different palette is active, color number 12 might be red. When a computer is set to display 256 colors or fewer, it can display only the colors in the palette currently active in the system. This means that images created to display with the colors of one palette do not appear correctly when a different palette is active. If you use palette index color in a movie and then switch palettes during the movie, or never make sure that the correct palette is active, the images in your movie may appear with the wrong colors.

Director identifies an RGB color as a set of hexadecimal numbers that specify the amounts of red, green, and blue required to create the color. When a computer is set to display thousands or millions of colors, Director always displays RGB colors accurately. When a computer is set to 256 colors, Director finds the closest color in the current color palette to approximate the RGB color.

To choose the color mode for the current movie, you use the Color Selection options in the Movie tab of the Property Inspector. When you choose RGB, all the colors you choose from the Color menu in Director are specified in RGB values. When you choose Palette Index, the colors you choose are specified according to their position in the current palette. The Color menu indicates which method is being used.

To change the color mode of a movie:

1. Display the Movie tab of the Property Inspector.
2. For Color Selection, choose either RGB or Palette Index.

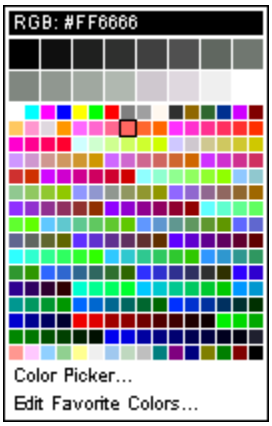
Changing the color depth of a movie

When you save a Director movie, it is set to the same color depth as the system on which you are authoring it. You can use Lingo to reset the system color depth to match the color depth of a movie. See **switchColorDepth**.

If you want to set the color depth of a movie without using Lingo, you can use system utilities to change the color depth of your system before you save the movie file. On the Macintosh, you can also make the movie reset the system color depth by choosing File > Preferences > General and selecting Reset Monitor to Movie's Color Depth.

Choosing colors for movie elements

Use the Color menu to choose colors for movie elements such as the Stage, vector shapes, and the foreground and background of sprites. For some elements, such as Stage and sprite colors, you can also enter hexadecimal values for any RGB color. The Color menu displays the colors in the current palette; the 16 larger color chips at the top of the menu identify your favorite colors.

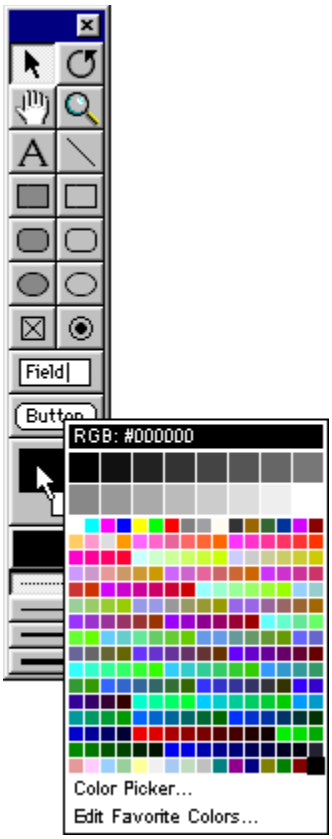


If the movie is set to specify colors as RGB values, choosing a color from the Color menu specifies the RGB value of the color, not its index value. (For an explanation of the difference between index and RGB color, see **Specifying palette index and RGB color**.) The bar at the top of the Color menu indicates whether the movie is set to RGB or index color.

If you want to choose a color that is not in the current palette (and therefore not available on the Color menu), you can use the system color picker to specify any color. You can also change the set of colors available on the Color menu by displaying a different color palette.

To open the Color menu:

1. Do one of the following:
   - Select a sprite and display the Sprite tab of the Property Inspector.
   - Choose Window > Tool Palette.
2. Click and hold the mouse button while pointing at the Foreground Color and Background Color buttons.



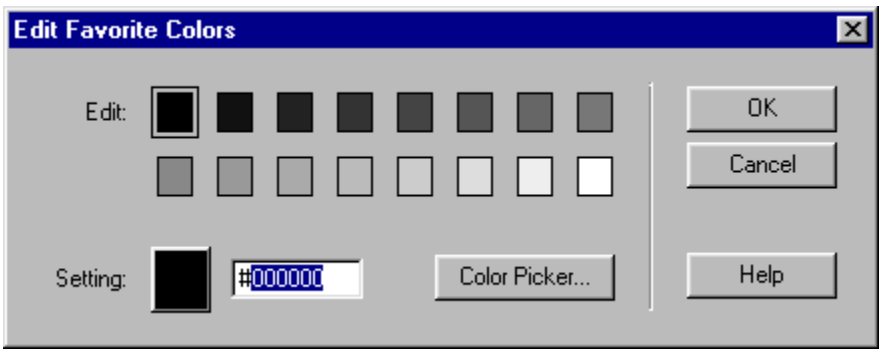
Note: To open the Color menu in the opposite mode (RGB or index), hold down the Alt key (Windows) or Option key (Macintosh) while clicking the color chip.

To choose colors not on the Color menu:

1. Open the Color menu.
2. Click Color Picker.
3. Use the color picker that is displayed to choose colors.

To edit the favorite colors on the Color menu:

1. Open the Color menu.
2. Choose Edit Favorite Colors.



3 Choose the color chip you want to change.

4 Choose a new color for the chip using one of the following options:

- Click the color box to open the Color menu and choose a color from the current palette.
- Enter an RGB value for a color in the box to the right of the color box.
- Click Color Picker and then use the system color picker to specify a new color.

5 Click OK.

To change the color palette displayed on the Color menu:

1 Choose Window > Color Palettes or double-click the mouse button on the Foreground Color and Background Color buttons in the Tool palette.

2 Choose a color palette from the Palettes pop-up menu.

Changing color palettes during a movie

The palette channel in the Score determines which palette is active for a particular frame in a movie. To define the palette that is active in a particular frame of a movie, use Modify > Frame > Palette. When the playback head reaches the frame with the palette change, Director switches to the new palette.

The settings in the palette channel have no effect on a movie playing in a Web browser. Do not use any of these settings for movies on the Web.

For a stand-alone disk-based movie that takes over the entire screen, changing palettes during a movie is a viable option for displaying 8-bit graphics with the best possible colors.

If you place a cast member that has its own custom palette on the Stage—and if it's the first cast member that has a different palette in the frame—Director automatically assigns the new palette to the palette channel. The new palette becomes the active palette unless you clear it from the palette channel or replace it with a different palette, and it remains in effect until you set a different palette in the palette channel.

Only one palette can be active at any time. If an 8-bit image appears with the wrong colors, it requires a different palette. See **Solving color palette problems**.

Director contains several color palettes. The Windows and Macintosh system palettes are the default selections. Web216 is nearly identical to the palettes used by Netscape Navigator and Microsoft Internet Explorer. Use it for any movie you plan to play in a browser. Any additional palettes you create or import appear as cast members.

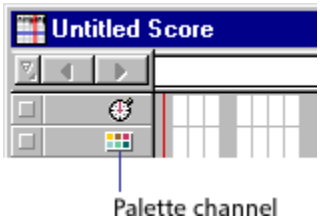
While working on a movie, you can change the active palette in the authoring environment by choosing a new palette in the Color Palettes window. The palette that is active in the authoring environment while you work does not change the palette in the movie you're working on. Any settings in the palette channel reset the active palette as soon as the movie plays.

To specify a palette:

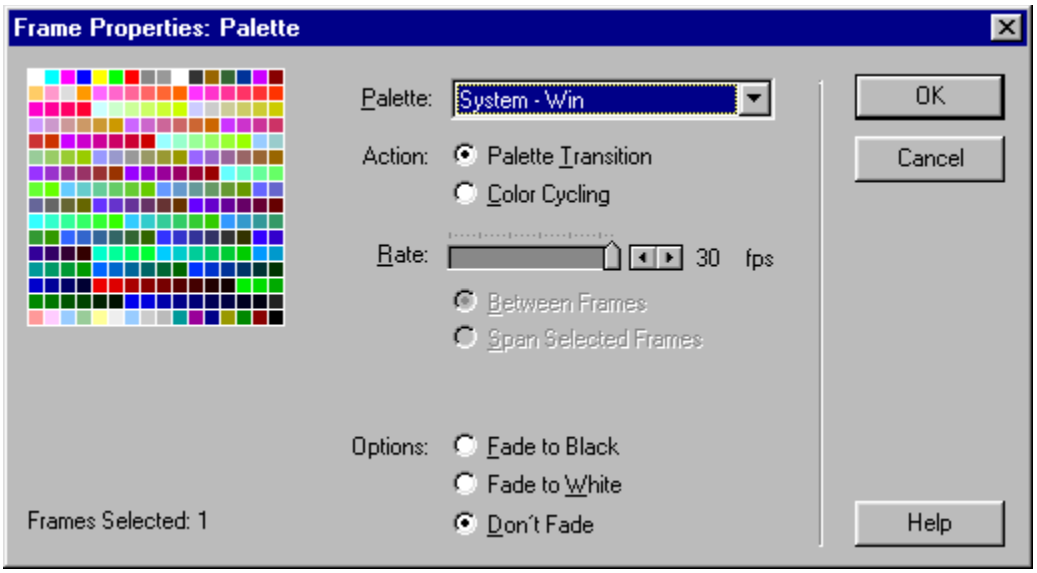
1. In the Score, do one of the following:

- Double-click the cell in the palette channel where you want the new palette setting to appear.
- Right-click (Windows) or Control-click (Macintosh) the cell in the effects channel where you want the new palette setting to appear, and then choose Palette from the Context menu.
- Select a frame in the palette channel and choose Modify > Frame > Palette.

  (If you don't see the palette channel, the effects channel is hidden. To display it, click the Hide/Show Effects Channel tool in the top right of the score window.)



2. Select the options you want to use in the Frame Properties: Palette dialog box.



- Choose a new palette.
- Specify how you want Director to manage the palette change. For example, to hide a palette change within a fade, first choose a new palette from the pop-up menu. Select the Palette Transition option and then select Fade to Black or Fade to White. Use the Rate slider to set the speed of the fade.

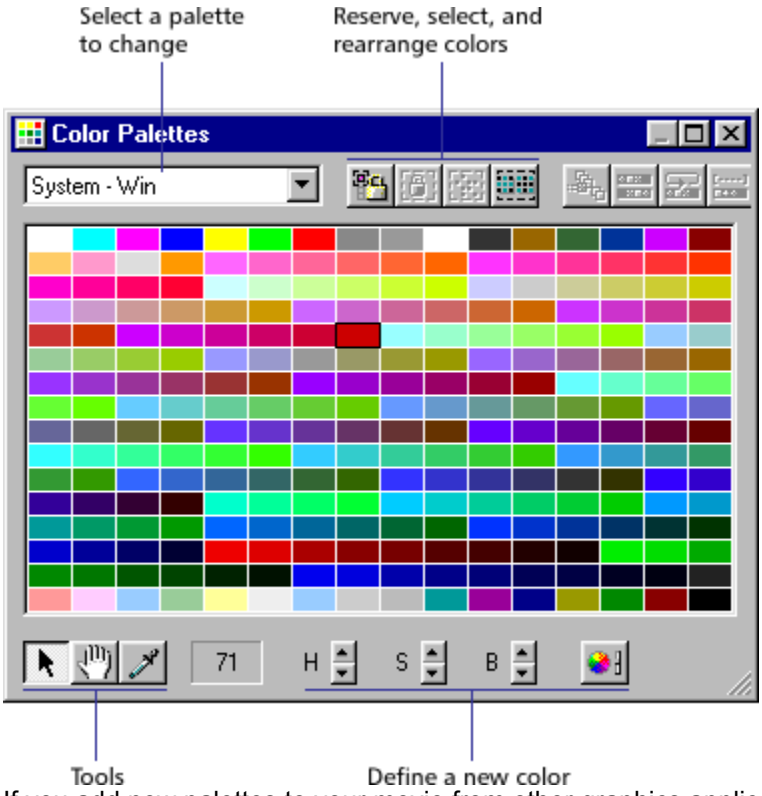
  To stop the movie while the palette changes, first choose a new palette from the Palettes pop-up menu. Select the Palette Transition option and then select Between Frames. Use the Rate slider to set the speed of the transition.

3 Click Set.

The palette you chose now appears in the cell you selected in the Score's palette channel. The setting remains in effect in the movie until you set a different palette in the palette channel.

Using the Color Palettes window

Use the Color Palettes window to change and rearrange color palettes and to determine which colors in a palette are used in an image. This section explains basic features of the Color Palettes window. For a description of specialized features, see **Special Color Effects** in the Director Support Center.



If you add new palettes to your movie from other graphics applications, those palettes appear in the palette list and in the Cast window.

The row of buttons on the right side of the Color Palettes window are for reserving, selecting, and rearranging colors in the current palette. If you attempt to change one of the nine built-in palettes, Director creates a copy of the palette for you to modify.

Note: Choosing a new palette in the Color Palettes window does not change the palette for the movie or any frame in the movie. Use the Movie tab in the Property Inspector to choose the movie color palette or choose Modify > Frame Palette to change the color palette at a particular frame.

When you modify a palette, all the cast members using the palette change as well, so make sure you always keep a copy of the original palette.

To open the Color Palettes window:

Choose Window > Color Palettes.

To edit a palette already used in a movie:

1. Choose Window > Color Palettes.
2. Select the palette you want to edit from the Palettes pop-up menu.
3. Double-click any color within the palette.

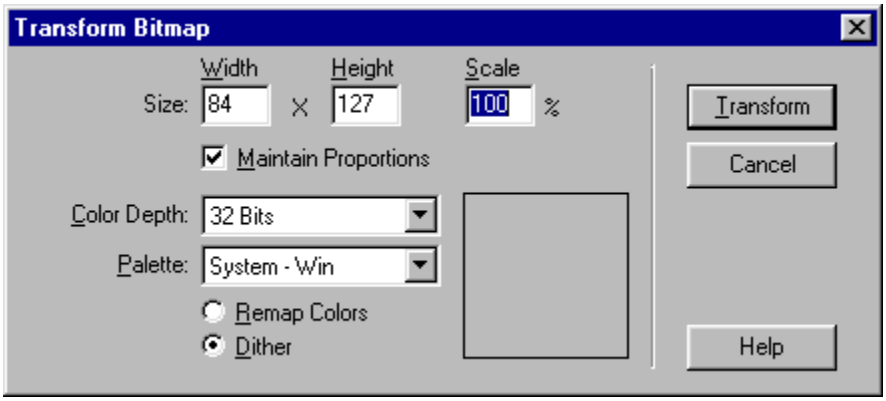
Director makes a copy of the palette and prompts you to enter a name.

4 Enter a name and press OK.

5 Edit the palette using any of the methods discussed later in this section.

6 Select all the cast members that use the old version of the palette, or use Find to locate all the cast members using a particular palette.

7 Choose Modify > Transform Bitmap and select the desired options.



Note: Be sure to select Remap Colors, not Dither.

8 Click Transform to remap all the cast members to the new palette.

To select one or more colors:

1 Click a color in the Color Palettes window. If the selection arrow is not active, click the Arrow tool at the bottom of the window. •

2 To select a range, drag across colors—or click the first color in the range, and then Shift-click the last.

3 Control-click (Windows) or Command-click (Macintosh) to select multiple discontiguous colors.

To match the color of any pixel on the Stage with the same color in the palette:

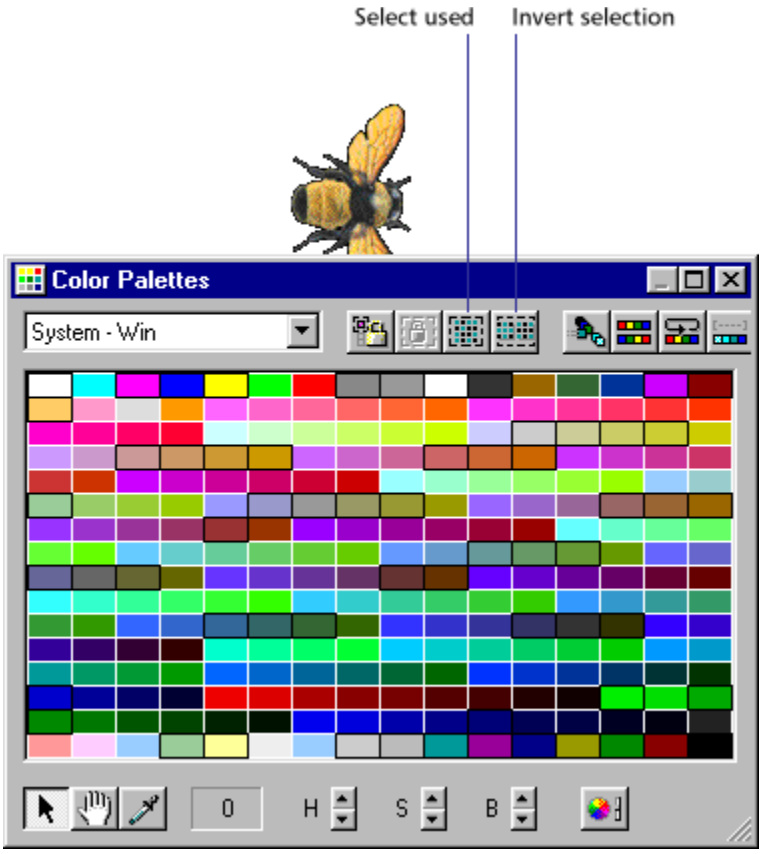
1 Click the Eyedropper tool. 

2 Drag any color in the Color Palettes window to any point on the Stage.

The selection in the Color Palettes window and the foreground color in the Tool palette changes to the color at the pointer location.

To select colors in the palette used by the current cast member:

1 Select the cast member or open the cast member in the Paint window.



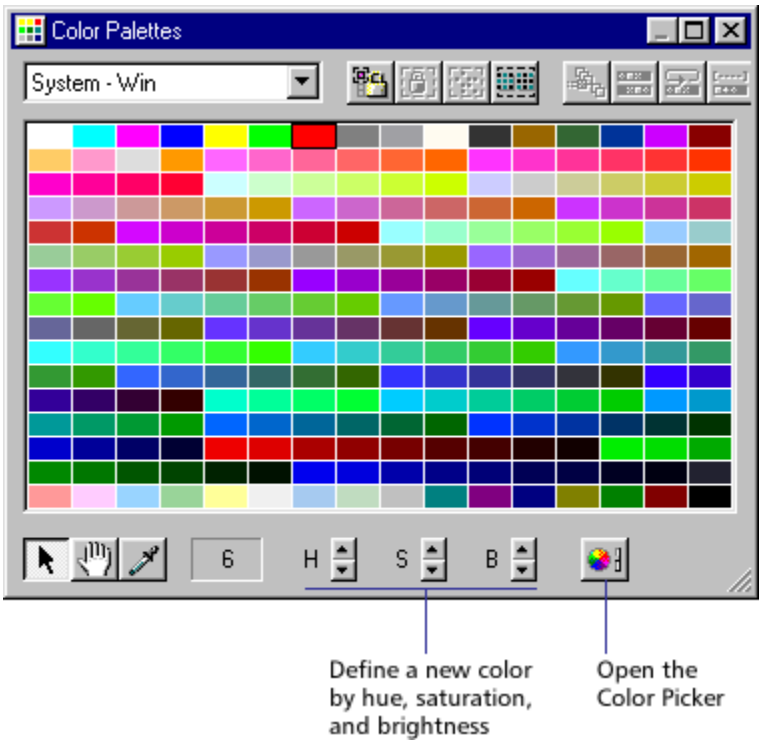
2 Click the Select Used button in the Color Palettes window. 

To select all colors not currently selected:

Click the Invert Selection button in the Color Palettes window. 

Changing colors in a color palette

You can define a new color for a color palette by selecting a color you want to change and then using either the controls at the bottom of the Color Palettes window or the system color.



To edit selected colors in the Color Palettes window:

1. Choose Window > Color Palettes.
2. Select the palette you want to change from the Palettes pop-up menu.
3. Select a color within the palette to change.

   If you attempt to change one of the default palettes, Director makes a copy of the palette and prompts you to enter a name.

4. To change the color using the H, S, and B (hue, saturation, and brightness) controls, click the arrows next to the controls.

- Hue is the color created by mixing primary colors.
- Saturation is a measure of how much white is mixed in with the color. A fully saturated color is vivid; a less saturated color is a washed-out pastel or, in the case of black, a shade of gray.
- Brightness controls how much black is mixed in with a color. Colors that are very bright have little or no black. As more black is added, the brightness is reduced, and the color gets darker. If brightness is reduced to 0, then no matter what the values are for Hue or Saturation, the color is black.

5. To change the color using the system color picker, click the Color Picker button.

   For instruction on using the Windows or Macintosh color picker, see your system documentation.

Controlling color palettes with Lingo

By using the `puppetPalette` command, you can change the current palette and specify how quickly a new palette fades in. This command is useful when you want to change the palette to suit changing conditions in the movie without entering a new frame. For example, you can change the palette when you switch a cast member assigned to a sprite.

The new palette remains in effect until a new `puppetPalette` command is issued, a new palette is set in the palette channel, or a new movie starts.

See **puppetPalette**.

Solving color palette problems

When images in your movie appear with the wrong colors, you probably have the wrong color palette active. Color palette problems occur only if you are using 8-bit bitmaps and you want your movie to be displayed correctly on 256-color systems (8-bit bitmaps always appear correctly on computers set to display thousands or millions of colors).

Eight-bit bitmaps don't store information about actual colors; they identify colors by referring to positions in the current color palette. When saving an 8-bit bitmap, a graphics program creates a palette with the colors required for that particular image. This palette is saved with the file and must be active when the bitmap appears in a Director movie for the bitmap to appear with the proper colors. Only one palette can be active at once. Whenever it's necessary to display more than one 8-bit bitmap on the screen at one time, as is often the case in Director movies, all the images must refer to the same palette.

To solve color palette problems, follow these guidelines:

- To avoid color problems in movies for the Web, map all 8-bit bitmaps in your movie to Director's built-in Web216 color palette. This is essentially the same palette used by Netscape Navigator and Microsoft Internet Explorer.
- Do not attempt to change palettes while a movie is playing in the browser. The browser, not the Director movie, controls the palette. Browsers ignore all palette channel settings.
- Make sure all 8-bit images that are on the Stage at the same time refer to the same palette.
- If bitmaps are not dithering or remapping to the current palette, make sure the Remap Palettes If Needed option in the Movie tab of the Property Inspector is selected. See **Setting Stage and movie properties**.
- Make sure there are no palette changes in the palette channel that you are unaware of. For example, when a cast member you are placing on the Stage has a palette different from the currently active palette, Director adds the new palette to the palette channel. If you don't realize that this has happened, you may find the palette changing unexpectedly when the movie plays.
- For disk-based movies, simplify your work and avoid frequent palette changes by mapping all the images in your movie to as few palettes as possible.
- Remap existing cast members to a new color palette using the Modify > Transform Bitmap command.
- If the Import option for Palette is not available while you are importing an image, the image's palette may not meet standard system requirements. Use an image editor to make sure the image's palette meets the following requirements: The palette must contain exactly 16 or 256 colors. The first and last colors in the palette must be black or white, and there must be only one black and one white in the entire palette.
- Don't change colors that are used by your system software for interface elements. In Windows, these colors always appear as the first ten and the last ten colors in the palette.

Setting palette cast member properties

When you create a color palette in the Color Palettes window or import a bitmap with its own palette, the palette appears in a cast as an ordinary cast member. Use cast member properties to name the palette and to specify how it is unloaded from memory.

To view or change color palette cast member properties:

1. Select a color palette cast member.
2. To display the Property Inspector, choose Modify > Cast Member > Properties or choose Window > Properties > Inspector.
3. If necessary, click the Member tab and display the Graphical mode.

   The following noneditable settings are displayed:

- The cast member size in kilobytes
- The cast member creation and edit dates
- The name of the last person who modified the cast member

4. To view or edit the cast member name, use the Name field.
5. To add comments about the cast member, use the Comments field.
6. To specify how Director removes the cast member from memory if memory is low, choose one of the following options from the Unload pop- up menu:

- 3—Normal sets the selected cast members to be removed from memory after any priority 2 cast members have been removed.
- 2—Next sets the selected cast members to be among the first removed from memory.
- 1—Last sets the selected cast members to be the last removed from memory.
- 0—Never sets the selected cast members to be retained in memory; these cast members are never unloaded.

7. To modify the colors in the palette, click Edit.

About tempo

Tempo is the number of frames per second that Director tries to play. You can control tempo using the Score tempo channel or Lingo's `puppetTempo` command.

Director tempo settings control the maximum speed at which the playback head moves from frame to frame. The tempo doesn't affect the duration of any transitions set in the transition channel, nor does it control the speed at which a sound or digital video plays. Note that tempo settings don't always control animated GIFs; see **Using animated GIFs**.

Settings in the tempo channel can also make a movie pause and wait for a mouse click or key press. For information on making a movie wait for a cue point in a sound or video, see **Synchronizing media**.

For simple movies, using the tempo channel is often the best way to define tempos. For more sophisticated control of the speed of a movie, use Lingo's `puppetTempo` command to control tempo.

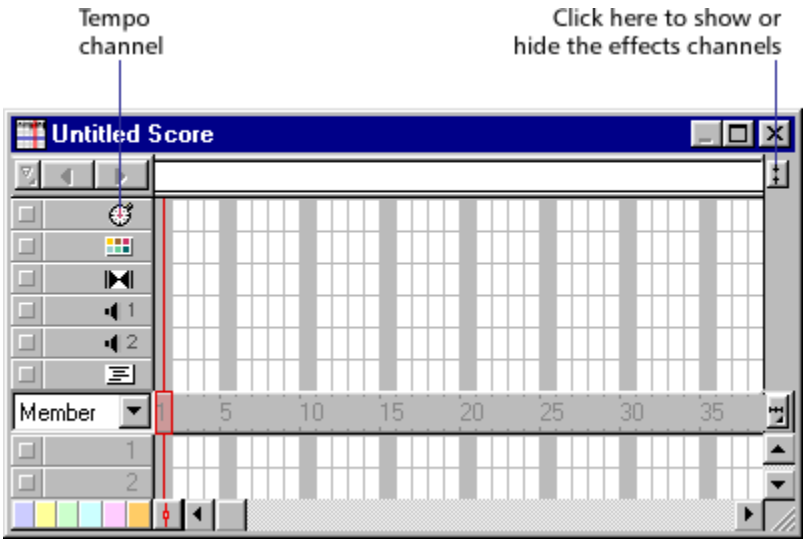
You can't make a movie go faster than the computer allows. Many factors can make movies play more slowly than the specified tempo, such as the following:

- Playing the movie on a slower computer
- Making the movie wait for cast members to download from a slow Internet connection
- Animating several large sprites at the same time
- Animating stretched sprites
- Color depth differences between the movie and monitor
- Animating sprites that have blend values

Specifying tempo properties

It's best to begin a movie with a tempo setting in the first cell of the tempo channel. If you don't set a tempo until later in the movie, the beginning tempo is determined by the setting in the Control Panel. Director plays a movie at the tempo you've set until it encounters a new tempo setting in the tempo channel or a `puppetTempo` command is issued.

Enter tempo changes in the tempo channel at the top of the Score. (If you don't see the tempo channel, the effects channel is hidden. To display it, click the Hide/Show Effects Channel tool in the top right of the score window.)



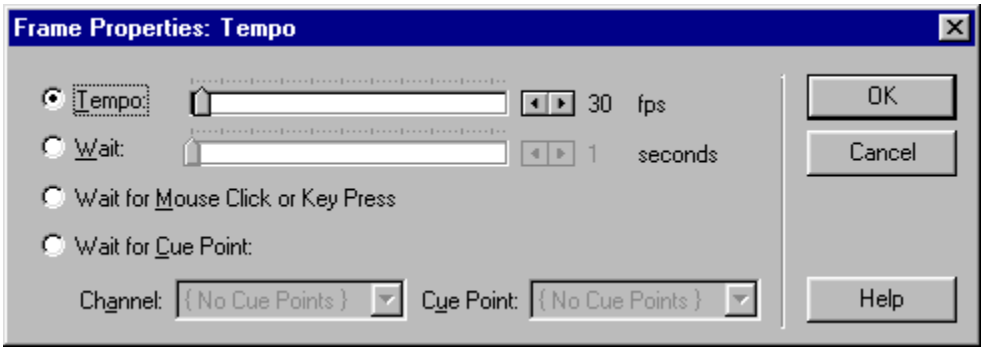
To specify a tempo setting:

1. In the score, do one of the following:

- Double-click the cell in the tempo channel where you want the new tempo setting to appear.
- Right-click (Windows) or Control-click (Macintosh) the cell in the effects channel where you want the new tempo setting to appear, and then choose Tempo from the Context menu.
- Select a frame in the tempo channel and choose Modify > Frame > Tempo.

  If you don't see the tempo channel, the effects channel is hidden. To display it, click the Hide/Show Effects Channel tool in the top right of the Score window.

2. Select the option you want to use in the Frame Properties: Tempo dialog box.



- To set a new tempo for the movie, use the Tempo arrows or drag the slider.
- To pause the movie at the current frame for a certain amount of time, use the Wait arrows or drag the slider.

- To pause the movie until the user clicks the mouse or presses a key, select Wait for Mouse Click or Key Press.
- To pause the movie until a sound or digital video cue point passes, select Wait for Cue Point and choose a channel and cue point. See **Synchronizing media**.

3. Click OK.

   A number that matches the setting you've chosen appears in the tempo channel. If you can't read the number, you may need to zoom the score. To do so, click the Zoom Menu button at the right edge of the sprite channel or choose View > Zoom. Then choose a percentage from the pop-up menu.

Comparing actual speed with tempos you've set

It's good practice to test the performance of your movie on a system similar to what your users have. Make sure the movie plays well on the slowest systems likely to be used.

The tempo you've set and the actual speed of a movie both appear in the Control Panel.

- **To compare the actual speed of a movie with the tempos you've set:**

1. Play the movie from start to finish, then rewind it to the beginning.
2. Use the Step Forward button to step through the movie frame by frame,
3. In each frame, compare the tempo setting shown in the Control Panel with the actual speed shown there.

   If you haven't recorded the actual speed of a movie in a particular frame, the Control Panel displays two dashes (--).

Locking frame durations

To make Director play a movie at the same tempo on all types of computers, use the Lock Frame Durations option in the Movie Playback Properties dialog box (**Setting movie playback options**). For frames without tempo settings, Director uses the current tempo. Lock Frame Duration prevents a movie from playing too fast on a fast system, but it cannot prevent a movie from playing slowly on a slow system.

To turn on Lock Frame Durations:

1. Choose Modify > Movie > Playback.
2. Select Lock Frame Durations.

Controlling tempo with Lingo

To override the tempo set in the movie's tempo channel, you use the `puppetTempo` command. This approach is useful when you want to change the movie's tempo in response to conditions that you can't control, such as the type of computer the movie is playing on or a user's action.

The `puppetTempo` command doesn't retain control of the tempo channel. If the movie encounters any tempo settings in the tempo channel, the `puppetTempo` settings will be overridden.

See **puppetTempo**.

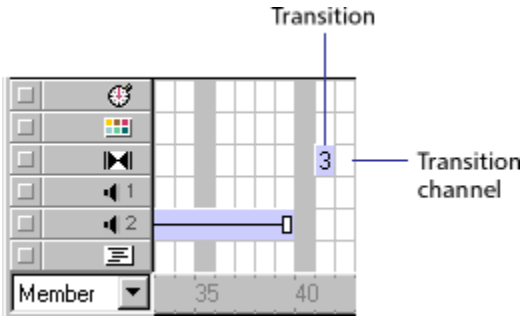
Using transitions

Transitions create brief animations that play between frames to create a smooth flow as sprites move, appear, or disappear or as the entire Stage changes. Director provides dozens of transitions built into the application, and many third-party Xtras include transitions as well. For example, you can dissolve from one scene to the next, display a new scene strip by strip, or switch to a scene as though revealing it through venetian blinds. You can also use many of the transitions to make individual elements appear or disappear from the screen.

Once they are defined, transitions appear in the Cast window as cast members. You can place them in the transition channel by dragging them from the cast to the Score.

Creating transitions

Transitions, like tempos, palettes, sounds, and behaviors, have a channel set aside for them in the Score.



A transition always takes place between the end of the current frame and the beginning of the frame where the transition is set. If you want to create a dissolve between two scenes, set the transition in the first frame of the second scene, not in the last frame of the first scene.

To add a transition:

1. In the transition channel, select the frame in which you want the transition to occur.
2. Choose Modify > Frame > Transition or double-click the frame in the transition channel.
3. In the Frame Properties Transition dialog box, choose a category if desired, then select the transition you want. You can quickly scroll through transitions by typing the first letter of the transition's name.

   Many transitions have default settings for Duration and Smoothness. You can adjust the sliders to change the settings.

   For many transitions, you can also select whether the transition affects the entire Stage or just the area that's changing.

   Xtra transitions may offer additional options provided by the developer. If the Options button is available when you choose an Xtra transition, click it to view and change the transition options.
4. Click OK.

   Director displays the cast member number that corresponds to the transition in the transition channel. The transition also appears in the cast.

Tips for using transitions

Here are some points to keep in mind when working with transitions:

- To play a sound while a transition occurs, place the sound in the frame immediately before the transition.
- The Dissolve Pixels, Dissolve Pixels Fast, or Dissolve Patterns transitions may look different on Windows and Macintosh systems. Test to ensure satisfactory results.
- If you export a movie that contains transitions as a digital video or PICS file, the transitions may not be preserved.
- A transition that occurs while a sound or digital video is decompressing may require more system resources than are available on less powerful systems. This may cause the sound to stop playing. If you notice this behavior while testing on low-end systems, try making the transition shorter, and avoid complex transitions such as Dissolve.
- Avoid looping on a frame that contains a transition. Playing a transition continuously may cause performance issues.
- Options will become available only when transition Xtras are available.

Using transition Xtras

You can add custom transitions that are available as transition Xtras. Transition Xtras appear in the Frame Properties: Transitions dialog box. Transition Xtras are often more complex than the transitions provided with Director and may include an additional dialog box for specialized settings.

To install a transition Xtra:

Place the transition Xtra in the Xtras folder in the Director application folder. The transition Xtra must be present when the movie runs.

Controlling transitions with Lingo

To set a transition with Lingo, you use the `puppetTransition` command. This command gives you the flexibility to select a transition appropriate for current movie conditions or to apply a transition to sprites before the playback head exits the current frame.

For example, use the `puppetTransition` command to specify one of several transitions, depending on which sprites are on the Stage when the playback head enters a new frame, or apply a transition to a new sprite when it appears but the playback head doesn't exit the frame.

The `puppetTransition` command applies only to the frame in which you issue the command. You do not need to explicitly return control of the transition channel to the Score after the transition occurs.

The `puppetTransition` command's parameters perform the same functions as the options in the Frame Properties: Transition dialog box.

See **puppetTransition**.

Setting transition cast member properties

Use the Property Inspector to set values for the transition cast member.

To view or change transition cast member properties:

1. Select a transition cast member.
2. To display the Property Inspector, choose Modify > Cast Member > Properties or choose Window > Properties > Inspector.
3. If necessary, click the Member tab and display the Graphical mode.

   The following noneditable settings are displayed:

- The cast member size in kilobytes
- The cast member creation and edit dates
- The name of the last person who modified the cast member

4. To view or edit the cast member name, use the Name field.
5. To add comments about the cast member, use the Comments field.
6. To specify how Director removes the cast member from memory if memory is low, choose one of the following options from the Unload pop- up menu:

- 3—Normal sets the selected cast members to be removed from memory after any priority 2 cast members have been removed.
- 2—Next sets the selected cast members to be among the first removed from memory.
- 1—Last sets the selected cast members to be the last removed from memory.
- 0—Never sets the selected cast members to be retained in memory; these cast members are never unloaded.

7. If you are using an Xtra transition, click Options to set values specific to the Xtra transition. The contents of the Options dialog box is determined by the developer of the Xtra. Refer to any documentation supplied with the Xtra.

Animation: Overview

Animation is the appearance of an image changing over time. The most common types of animation involve moving a sprite on the Stage (tweening animation) and using a series of cast members in the same sprite (frame-by- frame animation).

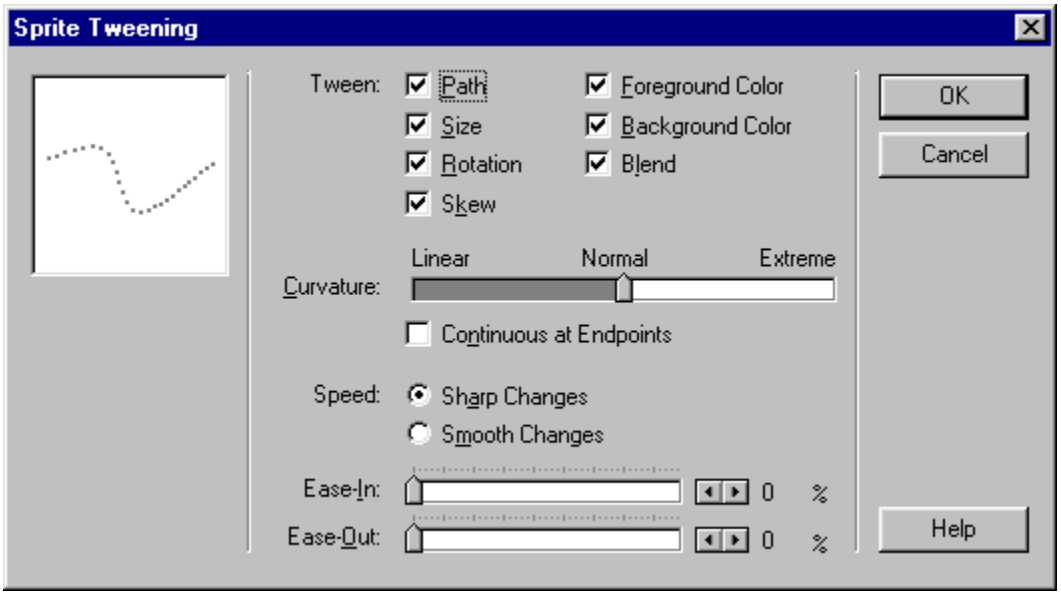
- Tweening is a traditional animation term that describes the process in which a lead animator draws the animation frames where major changes take place, called keyframes. Assistants draw the frames in between.
- Frame-by-frame animation involves manually creating every frame in an animation, whether that involves switching cast members for a sprite or manually changing settings for sprites on the Stage.

Other forms of animation include making a sprite change size, rotate, change colors, or fade in and out.

To specify tweening properties for a sprite, you use the Sprite Tweening dialog box.

To open the Sprite Tweening dialog box:

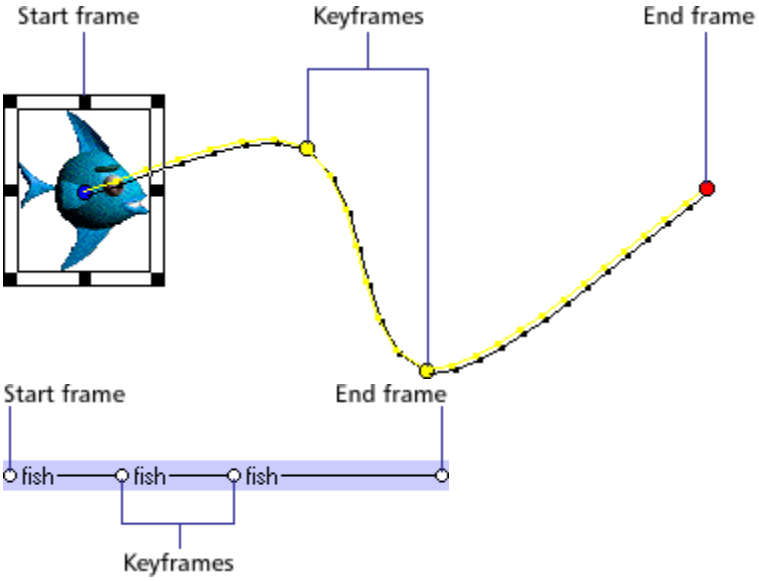
1. Select a sprite.
2. Choose Modify > Sprite > Tweening.



About tweening in Director

To use tweening in Director, you define properties for a sprite in frames called keyframes and let Director change the properties in the frames in between. Tweening is very efficient for adding animation to movies for Web sites, since no additional data needs to download when a single cast member changes.

A keyframe usually indicates a change in sprite properties. Properties that can be tweened are position, size, rotation, skew, blend, and foreground and background color. Each keyframe defines a value for all of these properties, even if you only explicitly define one.



Tweening the path of a sprite

Sprite paths are the lines Director displays on the Stage to show the movement of a sprite. Sprite paths are controlled by the Sprite Overlay Settings dialog box. You can change settings to make the paths appear for all sprites, for selected sprites, or only when the pointer rolls over a sprite. See **Using the Sprite Overlay**.

You can tween a sprite directly on the Stage by editing the sprite's path. Director displays the path of the selected sprite directly on the Stage. You can adjust the path by dragging keyframe indicators.

To tween the path of a sprite:

1. Place a sprite on the Stage where you want the path to start. If the sprite is already on the Stage, select it.

   This places the start frame of the sprite in the proper location. The start frame is also the first keyframe of the sprite.

2. If necessary, choose View > Sprite Overlay > Show Paths.

   The Show Paths option is on by default. With this option turned on, Director displays the paths of moving sprites on the Stage. Keyframes appear as hollow circles. Small tick marks show the sprite's position in tweened frames.

3. Insert keyframes in any additional frames where you want the sprite's animation path to change.

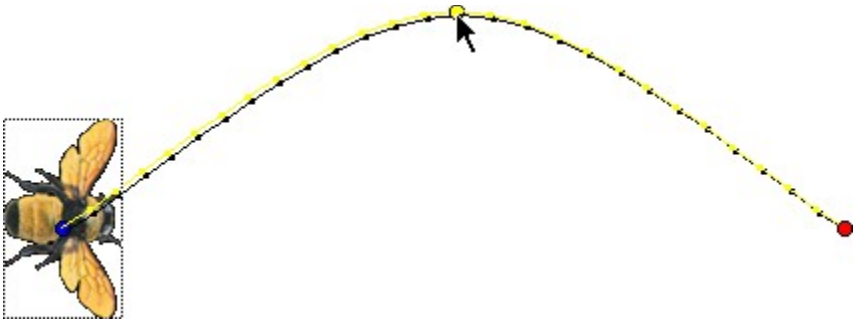
4. Drag the red handle within the sprite to the place on the Stage where you want the sprite's path to end.

   The red handle represents the sprite's location in the end frame. For bitmaps, the red handle is usually in the center of the image. For vector shapes and other media types, the handle is often in the upper left corner.

For an animated demonstration, see the **Drag Keyframe** movie.

5. Director displays the path the sprite will follow. The tick marks along the path show the sprite location in each frame in between.

6. To make the sprite's path curve between more points, hold down the Alt key (Windows) or Option key (Macintosh) and move the pointer on the Stage over a tick mark. When the pointer changes color, drag the tick mark to a new location.



For an animated demonstration, see the **Curve Keyframe** movie.
This creates a new keyframe and records the new location. Repeat this step to create additional keyframes.

7 To make the property changes defined by a keyframe occur at a different time, drag the keyframe in the Score to a new frame within the sprite.

8 To change the degree of curvature between keyframes, choose Modify > Sprite > Tweening and adjust the Curvature slider. To make the sprite move in the same direction at the beginning and end, select Continuous at Endpoints in the Sprite Tweening dialog box. This creates a circular motion. See **Changing tweening settings**.

Accelerating and decelerating sprites

To create more natural motion in tweened sprites, use the following settings in the Sprite Tweening dialog box:

- Ease-In and Ease-Out control how a sprite moves from its start frame to its end frame, no matter how many keyframes are in between. Ease-In makes a sprite move more slowly in the beginning frames; Ease-Out makes the sprite slow down in the ending frames. This setting makes the sprite move more like an object in the real world.
- The Speed settings control how Director moves a sprite between each keyframe. The Sharp Changes option is the default setting. Using this option, Director calculates how to move the sprite between each pair of keyframes separately. If a sprite's keyframes are separated by unequal numbers of frames in the Score, or by different amounts of space on the Stage, abrupt changes in speed may occur as the sprite moves between keyframe locations. Smooth out these speed changes by choosing the Smooth Changes option.

Sprite with modified ease-in and ease-out settings

To change the acceleration or deceleration of a sprite:

1. Use one of the tweening methods to create a moving sprite.
2. Turn on View > Sprite Overlay > Show Paths to see how far the sprite moves between each frame.
3. Select the sprite and choose Modify > Sprite > Tweening.
4. Use the Ease-In and Ease-Out sliders to specify the percentage of the sprite's path through which the sprite should accelerate or decelerate.
5. Choose one of the following speed settings:

- Sharp Changes moves the sprite between keyframe locations without adjusting the speed.
- Smooth Changes adjusts the sprite's speed gradually as it moves between keyframes.

Tweening other sprite properties

In addition to tweening a sprite's path, Director can tween the size, rotation, skew, blend, and foreground and background color of a sprite. Tweening size works best for vector-based cast members created in the Vector Shape window or Flash (bitmaps can become distorted when resized). Director can tween all of these properties at once.

To make a sprite fade in or out, you can tween blend settings. To make sprites spin or tilt, use rotation. To create gradual shifts in color, you can tween color settings.

For an animated demonstration of tweening color, see the **Tween Color** movie.

Note: To prevent Director from tweening a certain sprite property, choose Modify > Sprite > Tweening and turn off any of the tweening options.

To tween sprite properties:

1. If the Score isn't open, choose Window > Score.
2. Position a sprite on the Stage and make sure it spans all the frames in which you want the sprite to change.
3. Select the start frame of the sprite in the Score.
4. To tween size, scale the sprite or resize the sprite on the Stage. See **Resizing and scaling sprites**.
5. To define the beginning property settings, click the Sprite tab of the Property Inspector and do any of the following:

- To make the sprite fade in or out, enter a blend setting in the Property Inspector (in List view). Enter 0 to make the sprite fade in or 100 to make it fade out. For more information, see **Setting blends**.
- To tween rotation or skew, manually rotate or skew the sprite to the beginning position on the Stage or enter an angle in the Property Inspector. See **Rotating and skewing sprites**.
- To tween color, use the color boxes in the Property Inspector to open the color menu for foreground and background color, or enter the RGB values for a new color in the boxes at the right.

6. In the Score, select the end frame of the sprite.
7. Choose Insert > Keyframe.

   Note that the end frame is not a keyframe unless you create one there.
8. Make sure only the keyframe is selected (not the entire sprite), and then enter the ending values of the sprite properties you are tweening.

   For example, if you entered a blend setting of 0 in the first frame, you could enter a blend setting of 100 in this frame.
9. If necessary, create additional keyframes in the sprite and enter new values for the tweened properties.
10. To make the property changes defined by a keyframe occur at a different time, drag a keyframe in the Score to a new frame within the sprite.

To view the tweening, rewind and play the movie. Director gradually changes the value of the

tweened property in the frames between the keyframes.

Suggestions and shortcuts for tweening

Follow the suggestions listed here to improve results and productivity while tweening sprites.

- For smoother movements, tween across more frames, increasing the tempo if necessary.
- To achieve some types of motion, you may need to split the sprite and tween the sprites separately. See **Accelerating and decelerating sprites**.
- To quickly make duplicates, Alt-drag (Windows) or Option-drag (Macintosh) keyframes. This technique is useful when you want the start and end frames to have the same settings. This shortcut also provides a quick way to create a complex path. Insert a single keyframe, drag several duplicates to the proper frames, and then select the various keyframes and set positions on the Stage.
- To extend the sprite and leave the last keyframe in place, Alt-drag (Windows) or Option-drag (Macintosh) a keyframe at the end of a sprite.
- To move many keyframe positions at once, Control-click (Windows) or Command-click (Macintosh) multiple keyframes to select them and then move the sprite on the Stage.
- To make the animation look smoother, use an image editor to blur the edges of bitmaps.
- When tweening sprites that have a series of cast members, consider using a film loop instead. For more information, see **Using film loops**.
- Turn off all tweening options to make a sprite jump instantly between settings in different keyframes.

Changing tweening settings

To change tweening properties for sprites, you use the Sprite Tweening dialog box. You can turn tweening on and off for certain properties and control the curve of a tweening path and the way the speed changes as a sprite moves. For information on creating tweened animation, see **Tweening the path of a sprite**.

To change tweening settings:

1. Select a tweened sprite on the Stage or in the Score.
2. Choose Modify > Sprite > Tweening to open the Sprite Tweening dialog box.

   The diagram in the upper left corner shows the sprite's path as specified by the Curvature, Speed, Ease-In, and Ease-Out settings. This does not show the actual path of the sprite, just the type of curve it will follow.

   If the start and end points of the sprite are the same, the diagram is circular, indicating that the sprite travels in a circle when tweened. If the start and end points are not the same, the diagram describes a curved path, indicating that the sprite ends at a point different from the starting point.
3. To change which properties of the sprite are tweened, change the values for Tween.

   A check mark indicates that the property will be tweened. The available properties are Path, Size, Rotation, Skew, Foreground Color, Background Color, and Blend.
4. To change how the sprite curves between positions defined by keyframes, adjust the Curvature slider.

- Linear makes the sprite move in a straight line between the keyframe positions.
- Normal makes the sprite follow a curved path inside the keyframe positions.
- Extreme makes the sprite follow a curved path outside the keyframe positions.

5. To make the sprite move smoothly through start and end frames when it moves in a closed path, select Continuous at Endpoints.
6. To define how the tweened sprite positions change between keyframes, choose an option for Speed. See **Accelerating and decelerating sprites**.

- Sharp Changes makes the changes in position occur abruptly.
- Smooth Changes makes the changes in position occur gradually.

7. To define how tweened sprite positions change over the whole length of the sprite, use the sliders to change the values for Ease-In and Ease-Out.

- Ease-In defines the percentage of the sprite span through which the sprite accelerates.
- Ease-Out defines the percentage of the sprite span through which the sprite decelerates.

Switching a sprite's cast members

To show different content while maintaining all other sprite properties, you exchange the cast member assigned to a sprite. This technique is useful when you've tweened a sprite and you decide to use a different cast member. When you exchange the cast member, the tweening path stays the same.

To exchange cast members in the Score:

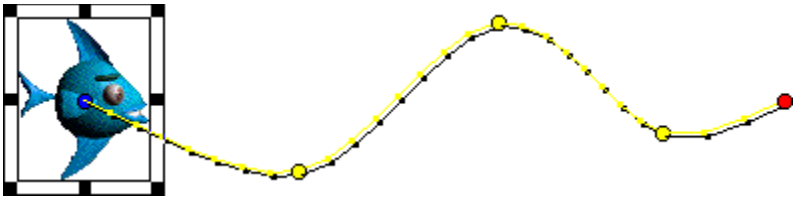
1. To change a cast member in every frame, select an entire sprite. To change a cast member only in certain frames, select part of a sprite.

   To select part of a sprite, press Alt and click the first frame that you want to select. Then press Control-Alt (Windows) or Option-Alt (Macintosh) and click each additional frame that you want to select.

2. Open the Cast window and select the cast member you want to use next in the animation.

3. Choose Edit > Exchange Cast Members.

If you selected an entire sprite, Director replaces the cast member for the entire sprite.



Before cast members are exchanged, the sprite moves like this.

After cast members are exchanged, the sprite still moves in the same way, but it displays a different cast member.

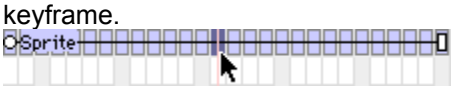
You can also use Lingo to switch the cast member assigned to a sprite. See **Assigning a cast member to a sprite with Lingo**.

Editing sprite frames

To change how a sprite is selected and how keyframes are created, you use the Edit Sprite Frames option. Use this option with sprites that have animation you need to adjust frequently; it's especially useful for cell animation in which each frame contains a different cast member in a different position.

Ordinarily, clicking a sprite on the Stage or in the Score selects the entire sprite.

When Edit Sprite Frames is turned on for a certain sprite, clicking the sprite selects a single frame. Any change you make to a tweenable property, such as moving a sprite on the Stage, defines a new keyframe.



To use Edit Sprite Frames:

Select sprites and choose Edit > Edit Sprite Frames. You can also Alt-double- click (Windows) or Option-double-click (Macintosh) a frame within the sprite.

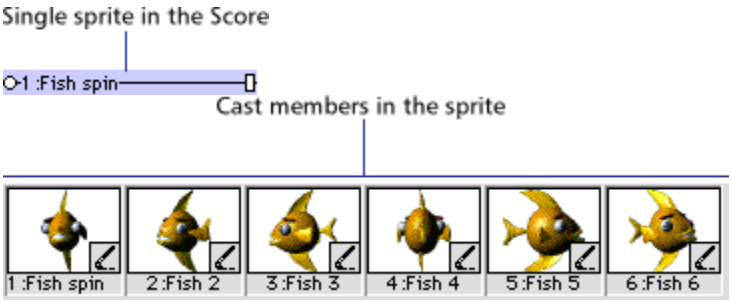
To return sprites to their normal state:

Select sprites and choose Edit > Edit Entire Sprite. You can also Alt-double- click (Windows) or Option-double-click (Macintosh) a frame within the sprite.

Frame-by-frame animation

To create animation that is more complex than is possible with simple tweening, you can use a series of cast members in frame-by-frame animation. Sprites usually refer to only one cast member, but they can refer to different cast members at different times during the life of the sprite.

For example, an animation of a man walking may display several cast members showing the man in different positions. By placing all the images in a sequence within a single sprite, you can work with the animation as if it were a single object.



A single sprite can display several cast members.

Sprite animating

For an example of frame-by-frame animation, see the **Frame-byFrame** movie.

Use this approach sparingly for movies that will be downloaded from the Internet, because all cast members must be downloaded before the animation can run. As an alternative to this type of animation, consider using vector shapes, rotation and skewing on bitmap cast members, or a Flash movie.

You can create multiple-cast-member animations in a variety of ways in Director. The following procedure explains a basic approach. The Cast to Time command provides an effective shortcut; see **Shortcuts for animating with multiple cast members**.

Note: The best way to prepare cast members for use in multiple-cast-member animation is with onion skinning in the Paint window. For more information, see **Using onion skinning**.

To animate a sprite with multiple cast members:

1. Create a sprite by placing the first cast member in the animation on the Stage in the appropriate frame.
2. Change the length of the sprite as needed.

   Drag the start or end frame in the Score, or enter a new start or end frame number in the Sprite Inspector.
3. Choose View > Display > Cast Member.

   This setting displays the name of the cast member on each sprite. For more information, see

Displaying sprite labels in the Score.

4 Choose View > Sprite Labels > Changes Only.

This setting changes the view of the Score to show the name of each sprite's cast member when it changes. This makes it easy to identify frames where the cast member changes. You may also want to zoom the score to 800% so the frames are wide enough to display the cast member information.

5 Choose Edit > Edit Sprite Frames.

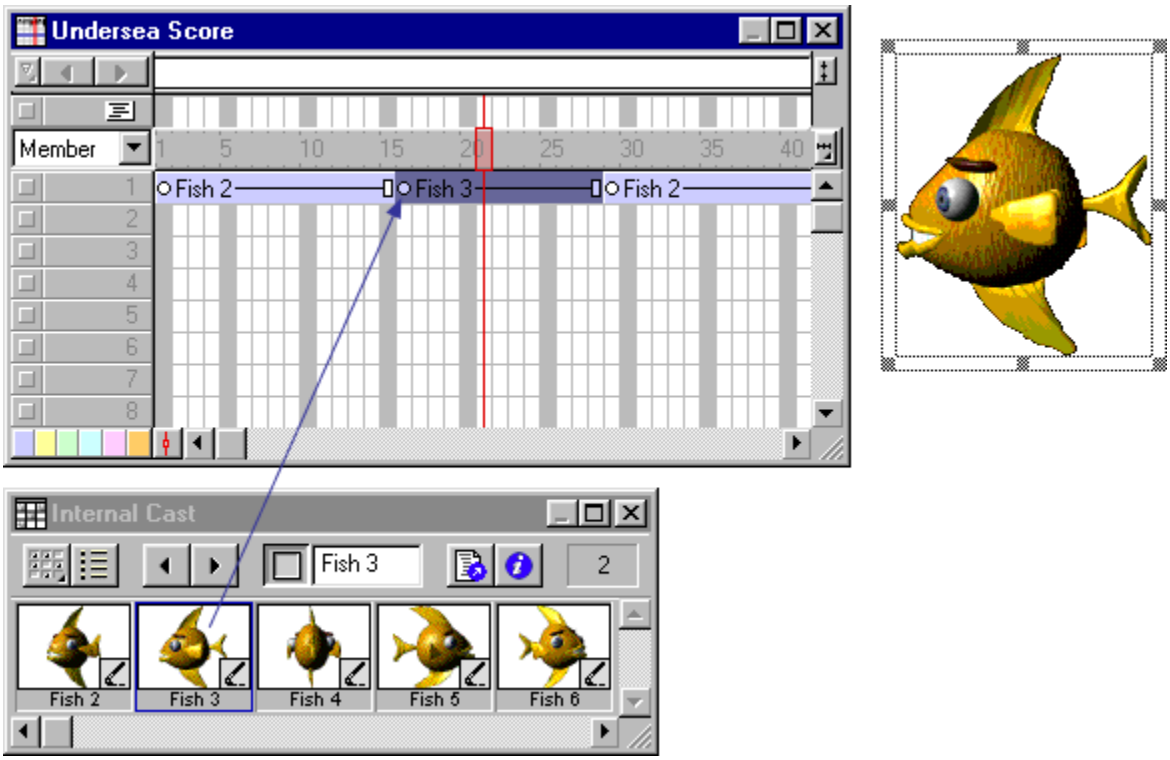
Edit Sprite Frames makes it easier to select frames within a sprite. See **Editing sprite frames**.

6 Select the frames in the sprite where you want a different cast member to appear.

7 Open the Cast window and select the cast member you want to use next in the animation.

8 Choose Edit > Exchange Cast Members.

Director replaces the cast member in the selected frame with the cast member selected in the Cast window.



9 Repeat these steps to complete the animation. Choose Edit Entire Sprite when you're done.

Sometimes a series of cast members placed in the Score jumps unexpectedly when you play the movie. This occurs because the cast members' registration points aren't aligned properly. When you exchange cast members, Director places the new cast member's registration point precisely where the previous cast member's registration point was. By default, Director places registration points in the center of a bitmap cast member's bounding rectangle.

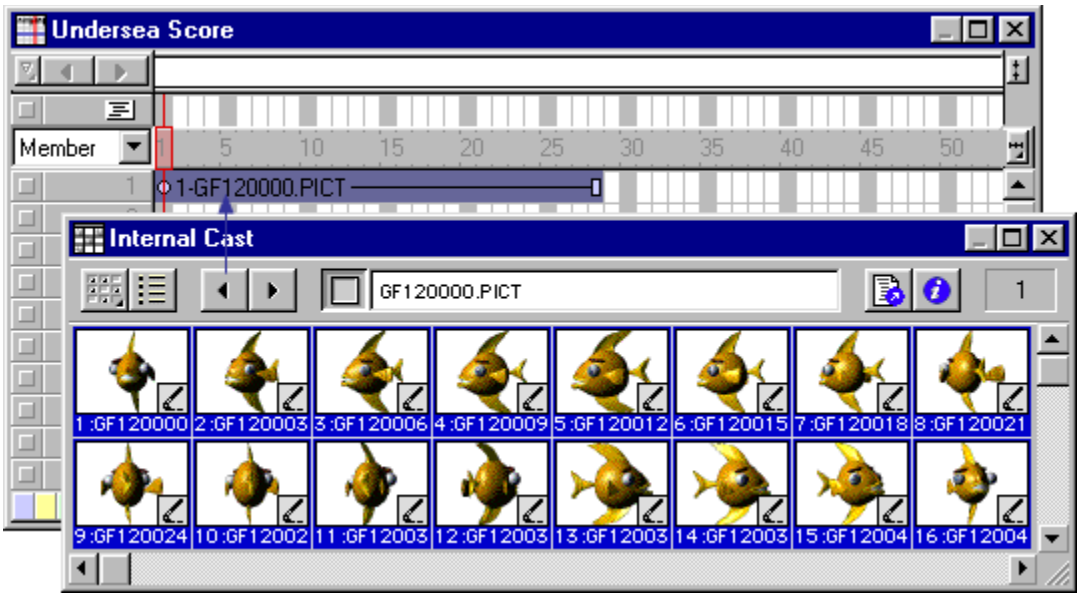
For information about aligning registration points, see **Changing registration points**. You can also align sprites relative to their bounding rectangles. See **Positioning sprites using guides, the grid, or the Align window**.

Shortcuts for animating with multiple cast members

The Cast to Time and Space to Time commands are both useful shortcuts for animating with multiple cast members.

Using the Cast to Time command

To move a series of cast members to the Score as a single sprite, you use the Modify > Cast to Time command, which is one of the most useful methods for creating animation with multiple cast members. Typically, you create a series of images and then use Cast to Time to quickly place them in the Score as a single sprite. Director's onion skinning feature is also useful for creating and aligning a series of images for use in animation. For more information, see **Using onion skinning**.



Cast to Time places selected cast members in the Score as a single sprite.

To create a sprite from a sequence of cast members:

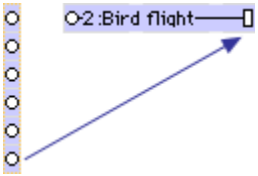
1. Select the frame in the Score where you want to place the new sprite.
2. Make the Cast window active.
3. Select the series of cast members to be placed in the new sprite.
4. Choose Modify > Cast to Time, or hold down Alt (Windows) or Option (Macintosh) and drag the cast members to the Stage.

   The selected series of cast members becomes a single sprite.

Using the Space to Time command

To move sprites from adjacent channels to a single sprite, you use the Modify > Space to Time command. This method is convenient when you want to arrange several images on the Stage in one frame and then convert them to a single sprite.

Arrange sprites on the Stage in a single frame.



Space to Time converts sprites from adjacent channels to a single sprite.

Onion skinning provides a benefit in the Paint window similar to that provided by Space to Time on the Stage. For more information, see **Using onion skinning**.

To use the Space to Time command:

1. Choose File > Preferences > Sprite and set Span Duration to 1 frame.

   Set the span duration to any setting you like, but Space to Time works best with shorter sprites.

2. Select an empty frame in the Score.

   This is usually at the end of the Score.

3. Drag cast members onto the Stage to create sprites where you want them to appear in the animation.

   As you position the sprites on the Stage, Director places each sprite in a separate channel. Make sure all the sprites are in consecutive channels.

4. Select all the sprites that are part of the sequence in the Score or on the Stage.

5. Choose Modify > Space to Time.

   The Space to Time dialog box appears. Set the number of frames you want between each cast member.

6. Enter an interval.

   Director rearranges the sprites so that instead of being arranged from top to bottom in a single frame, they're arranged in sequence from left to right in a single sprite.

Note: Space to Time is a fast way to set up keyframes for a sprite to move along a curve. Arrange the cast members in one frame, choose Modify > Space to Time, and add 10 to 20 cells between each cast member to produce a smooth curve.

Using film loops

A film loop is an animated sequence that you can use like a single cast member. For example, to create an animation of a bird flying across the Stage, you can create a film loop of the sequence of cast members that shows the bird flapping its wings. Instead of using the frame-by-frame technique, you create a sprite containing only the film loop and then animate it across as many frames as you need. When you run the animation, the bird flaps its wings and at the same time moves across the Stage.

You can also use film loops to consolidate Score data. Film loops are especially helpful when you want to reduce the number of sprite channels you're using. You can combine several Score channels into a film loop in a single channel.

To determine if a film loop is cropped or scaled within a sprite's bounding rectangle and to make the film loop repeat or mute its sounds, you use the Film Loop Cast Member properties. See **Setting film loop properties**.

- *Film loops are useful for animating repetitive motions and combining sprites to use fewer channels.*

To create a film loop:

1. In the Score, select the sprites you want to turn into a film loop.

   Use sprites in as many channels as you need in film loops—even in the sound channel. Select sequences in all the channels you want to be part of the film loop. You can select sprite fragments if you first select a sprite and choose Edit > Edit Sprite Frames. Control-click (Windows) or Command- click (Macintosh) to select sequences that aren't in adjacent channels.

2. Choose Insert > Film Loop.

   A dialog box appears asking you to name the film loop.

3. Enter a name for the film loop.

   Director stores all the Score data and cast member references as a new film loop cast member.

Note: Drag a selection from the Score to the Cast window to quickly create a film loop cast member in that position.

A film loop behaves just like any other cast member, with a few exceptions:

- When you step through an animation that contains a film loop (either by using Step Forward or Step Backward or by dragging the playback head in the Score), the film loop doesn't animate. Animation occurs only when the movie is running.
- You can't apply ink effects to a film loop. If you want to use ink effects with a film loop, you need to apply them to the sprites that make up the animation before you turn the animation into a film loop.
- Lengthening or shortening a sprite that contains a film loop doesn't affect how fast the film loop plays. It changes the number of times the film loop cycles.

Director provides three other ways of incorporating a completed animation into a movie as a discrete element: you can export it as a digital video (QuickTime or AVI), save and import it as a linked Director movie, or play it in a window in another Director movie.

Note: If you need to edit a film loop and you've deleted the original Score data it was based on, it's possible to restore the Score data for editing. Copy the film loop cast member to the Clipboard, select a cell in the Score, and then paste. Director pastes the original Score data instead of the film loop.

Setting film loop properties

To determine if a film loop is cropped or scaled within a sprite's bounding rectangle and to make the film loop repeat or mute its sounds, you set properties for the film loop cast member.

To set film loop properties:

1. Select a film loop cast member.
2. To display the Property Inspector, choose Modify > Cast Member > Properties or choose Window > Properties > Inspector.
3. If necessary, click the Member tab and display the Graphical view.

   The following noneditable settings are displayed:

- The cast member size in kilobytes
- The cast member creation and edit dates
- The name of the last person who modified the cast member

4. To view or edit the cast member name, use the Name field.
5. To add comments about the cast member, use the Comments field.
6. To specify how Director removes the cast member from memory if memory is low, choose one of the following options from the Unload pop- up menu:

- 3—Normal sets the selected cast members to be removed from memory after all priority 2 cast members have been removed.
- 2—Next sets the selected cast members to be among the first removed from memory.
- 1—Last sets the selected cast members to be the last removed from memory.
- 0—Never sets the selected cast members to be retained in memory; these cast members are never unloaded.

7. Click the Film Loop tab and display the Graphical view.
8. To determine how the film loop appears within the sprite bounding rectangle, choose Framing options:

- Crop makes the movie image appear at its default size. Any portions that extend beyond the sprite's rectangle are not visible.
- Center is available only if Crop is selected. It determines whether transformations occur with the cast member centered within the sprite or with the cast member's upper left corner aligned with the sprite's upper left corner.
- Scale fits the movie inside the bounding rectangle.

9. To determine how the film loop plays back, use the following settings:

- Play Sound plays the sound portion of the film loop. Turn this option off to mute sounds.
- Loop replays the film loop continuously from the beginning to the end and back to the beginning.

Step-recording animation

Step recording is a process of animating one frame at time. You record the position of a sprite in a frame, step forward to the next frame, move the sprite to its new position, step forward to the next frame, and so on until you've completed the animation. This method is useful for creating sprites that follow irregular paths.

To step-record animation:

1. Place sprites on the Stage where you want the animation to begin.
2. Select all the sprites you want to animate.
3. In the Score, click the frame where you want animation to begin.
4. Choose Control > Step Recording.

   The step-recording indicator appears next to the channel numbers for the sprites being recorded, and the selection border widens.
5. Press 3 on the numeric keypad (make sure Number Lock is off) or click  on the Control Panel.

   The movie advances to the next frame. If you reach the last frame of a sprite, Director extends the sprites being recorded into the new frame.

   > ***Note:*** As soon as you move the animation in any way other than stepping—such as using Rewind, Play, or Back—recording stops.

6. Drag the sprite to reposition it.

   You can also stretch the sprite, exchange cast members, or change any property.
7. Repeat steps 5 and 6 until you've completed the sequence you want to record.
8. Choose Control > Step Recording again to stop recording.

   You can also rewind the movie to stop recording.

Real-time recording animation

You can create animation by recording the movement of a sprite as you drag it across the Stage. The real-time recording technique is especially useful for simulating the movement of a pointer or for quickly creating a complex motion for later refinement.

For better control when you're recording in real time, use the Tempo control on the Control Panel to record at a speed that's slower than normal.

To use real-time recording:

1. Select one or more sprites on the Stage or in the Score.

   Recording will begin at the playback head. It's best to select a sprite in a channel that contains no other sprites later in the movie.

   To record in a specific range of frames, select the frames, and then click the Selected Frames Only button on the control panel. 

2. Choose Control > Real-Time Recording.

   The real-time recording indicator appears next to the channel numbers for the sprite being recorded, and a red and white selection frame appears around the sprite. Recording begins as soon as you drag the sprite on the Stage, so be prepared to move the mouse.

3. Drag the sprite on the Stage to record a path for the sprite.

   Director records the path.

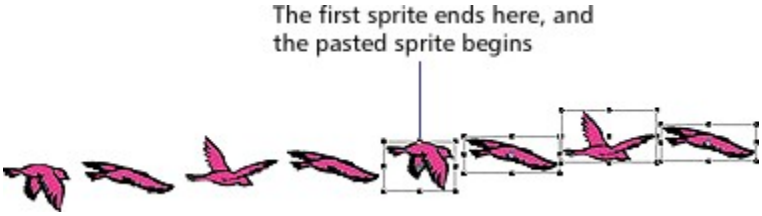
4. Release the mouse button to stop recording.

   The movie continues to play until you stop it.

Note: If you select Trails for the sprite, you can also use real-time recording to simulate handwriting.

Linking a sequence with Paste Relative

Paste Relative automatically aligns the start frame of one sprite with the end frame of the preceding sprite. It's useful for extending animations across the Stage.



To paste one sequence relative to another:

1. Select a sprite in the Score.
2. Choose Edit > Copy.
3. Select the cell immediately after the last cell in the sprite.
4. Choose Edit > Paste Special > Relative.

   Director positions the beginning of the pasted sprite where the previous sprite ends.

Repeat the process as many times as you need to create one continuous animation across the Stage.

Animating sprites with Lingo

Lingo can create animation regardless of the settings in the Score. This lets you create or modify animation depending on movie conditions.

To move a sprite on the Stage, you use Lingo that controls the sprite's location. See **bottom**, **left**, **right**, and **top**.

To animate a sprite by switching the sprite's cast members, change the sprite's `member` property. See **member (sprite property)**.

Navigation and user interaction: Overview

Adding interactivity lets you involve your audience in your movies. Using the keyboard, the mouse, or both, your audience can download content from the Internet, jump to different parts of movies, enter information, move objects, click buttons, and perform many other interactive operations.

Unless made to do otherwise, a movie plays through every frame in the Score from start to finish. Behaviors and Lingo can make the movie jump to a different frame, movie, or URL when a specified event occurs. With Lingo, you can include simple navigation instructions as part of more complex handlers; you can also place navigation Lingo in movie scripts and scripts attached to cast members such as buttons.

There are several other interactive features that you can add to your movie:

- Draggable sprites give your audience the ability to move sprites anywhere on the Stage. You can also create boundaries beyond which sprites cannot move.
- Editable fields are fields in which your audience can enter or edit information.
- Rollovers make certain sprites change in appearance when the mouse pointer passes over them, even if the user has not clicked the mouse. Using rollovers is an excellent way to give your audience feedback based on their actions.
- The cursor (that is, the mouse pointer) can be changed based on criteria you choose. Using Lingo, you can provide animated cursors or specify one of the standard cursors or a bitmap cast member as a cursor image. See **cursor (command)** and **cursor (sprite property)**.

Creating basic navigation controls with behaviors

Director provides behaviors that allows you to create basic navigation controls without knowing Lingo. You can use behaviors to move the playback head to a frame number or marker. You can also stop the playback head at any frame and wait for the user to act.

The following examples explain the basic use of the `Hold on Current Frame` and `Go Next Button` behaviors. You can also create your own navigation behaviors or get them from third-party developers.

To use basic navigation behaviors:

1. Create a movie that contains a sprite in frame 1, and at least one marker in a later frame.
2. Choose Window > Library Palette and select the Navigation library.
3. Drag `Hold on Current Frame` to frame 1 in the script channel.

   Typically, you use this behavior in a frame that requires user interaction such as selecting a menu command.
4. Play the movie.

   The playback head remains in frame 1 where you attached the behavior. Notice that the movie is still playing, but the playback head remains on the single frame. Use `Go Next Button` to send the playback head to a new frame and continue playing, as described in the following steps.
5. Stop the movie.
6. Drag the `Go Next Button` behavior from the Library palette to the sprite in frame 1.
7. Rewind and play the movie again.

   The playback head is again stopped in the first frame by the `Hold on Current Frame` behavior.
8. Click the sprite to which you attached the `Go Next Button` behavior.

   The playback head jumps to the frame containing the next marker and continues playing.

Jumping to locations with Lingo

Lingo's navigation features can make a movie jump to other frames, to other movies, or to Internet movies and Web pages. You can also use Lingo to make a movie appear to pause by looping in one frame or a group of frames.

For details about specifying the locations of frames, markers, and movies, see "Identifying frames with Lingo" on page 74.

Jumping to a different frame

Lingo lets you jump to a different frame in the current movie or in another movie.

- To jump to a specific frame in the current movie, use the `go to` command, followed by an identifier for the frame.

  For example, the statement `go to "Begin Over"` jumps to the frame labeled Begin Over.

- To jump to the beginning of a different movie, use the `go to` command followed by an identifier for the movie.

  For example, the statement `go to movie "Citizen_Kane"` goes to the beginning of the movie Citizen_Kane.dir.

- To jump to a frame in a different movie, use the `go to` command followed by an identifier for the frame and the movie; use `frame` followed by the frame identifier and `movie` followed by the name of the movie.

  For example, the statement `go to frame "Rosebud" of movie "Citizen_Kane"` goes to the frame labeled Rosebud in the movie Citizen_Kane.dir.

See **go**.

Jumping to a URL

Lingo lets you jump to a URL that represents an Internet movie or a Web page.

- To jump to an Internet movie, use the `gotoNetMovie` command.
For example, the statement `gotoNetMovie "http://www.yourserver.com/movies/movie1.dcr"` retrieves and plays the movie named movie1.dcr. See **gotoNetMovie**.
- To jump to a Web page, use the `gotoNetPage` command.
For example, the statement `gotoNetPage "http://www.yourserver.com/movies/intro.html"` displays the Web page named intro.html in a browser window. See **gotoNetPage**.

Looping in a group of frames

Looping within frames lets you create animation that recycles or makes a movie appear to pause. Such looping is useful for allowing a network operation to complete before the movie proceeds. Looping a movie by jumping from the current frame back to the first frame in the sequence can create a recycling animation effect.

- To loop within one segment of the Score, use the statement `go loop` to return to the first marker to the left of the frame containing the `go loop` statement. If there is no previous marker, the playback head jumps to frame 1.
- To pause a movie in one frame but keep it playing so the movie can react to events, use the statement `go to the frame` to loop in the current frame.
- To resume playing a movie that is looping in one frame, use the statement `go to the frame + 1`.

Jumping away and returning to the original location

You may want a movie to jump to a different frame or a separate movie and then return to the original frame. For example, at a Web site that describes the weather, you could jump to a movie segment that explains a weather term, and then return to the original location.

To jump away and return to the original location:

Use the `play` and `play done` commands.

The `play` command branches a movie to another frame, another movie, or a specified frame in another movie. The `play done` command remembers the original frame and returns to it without requiring that you specify where to return.

Use the `play` and `play done` commands in these situations:

- When the movie you want to play does not have instructions about where to return.
- When you want to play several movies sequentially from a single script. When one movie finishes, Lingo returns to the script that issued the `play` command.
- When you want to put one sequence inside another sequence and easily return to where you were in the outer sequence.
- When you want to jump to one loop from several different locations.

See **play** and **play done**.

Detecting mouse clicks with Lingo

Users can click the mouse button in several ways, each of which Lingo can detect. The following are ways that you can use Lingo to detect what the user does with the mouse.

- To determine the last place the mouse was clicked, use the `clickLoc()` function. See **clickLoc**.
- To determine the last active sprite (a sprite with a script attached) that the user clicked, use the `clickOn` function. See **clickOn**.
- To determine whether the last two clicks were a double-click, use the `doubleClick` function. See **doubleClick**.
- To determine the time since the mouse was last clicked, use the `lastClick()` function. See **lastClick()**.
- To determine whether the mouse button is pressed, check the `mouseDown` property. See **the mouseDown (system property)**.
- To determine whether the mouse button is released, check the `mouseUp` property. See **the mouseUp (system property)**.
- To determine whether the user presses the right mouse button (Windows) or Control+click (Macintosh), check the `rightMouseDown` property. See **rightMouseDown (system property)**.
- To determine whether the user releases the right mouse button (Windows) or Control+click (Macintosh), check the `rightMouseUp` property. See **rightMouseUp (system property)**.

For example, this handler checks whether the user double-clicked the mouse button and, if so, runs the handler `openWindow`:

```
on mouseDown
  if the doubleClick = TRUE then openWindow
end
```

Making sprites editable and draggable

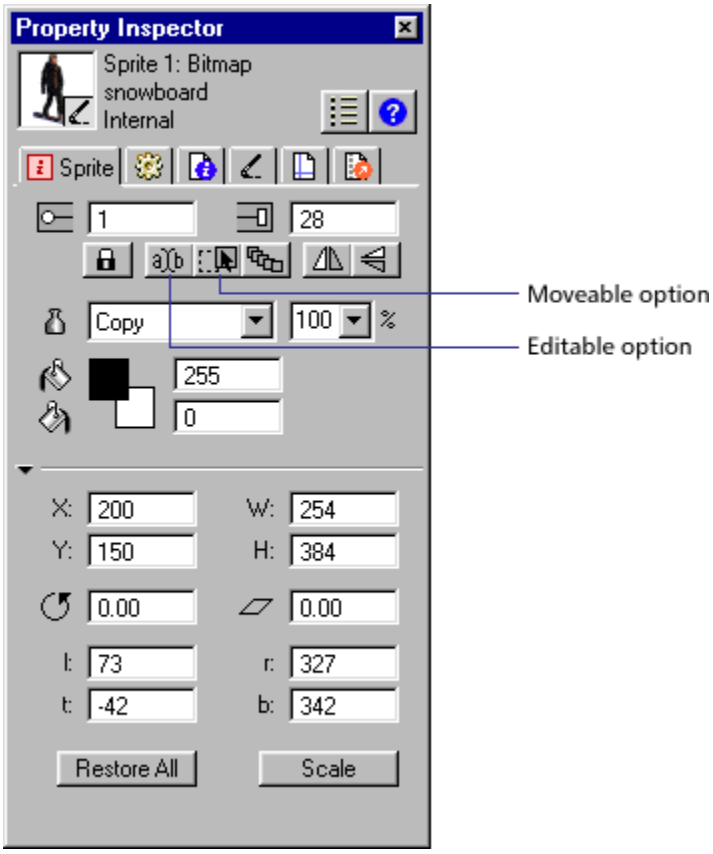
Using the Property Inspector, you can make a sprite editable, draggable, or both while your movie is running. See “Displaying sprite properties in the Property Inspector” on page 94.

To make a sprite draggable on the Stage:

Click the Moveable button in the Property Inspector.

To make a text sprite editable:

Click the Editable button in the Property Inspector.



Making sprites editable or moveable with Lingo

Lingo can make sprites editable or moveable regardless of the settings in the Score. You can also use Lingo to constrain a moveable sprite to a certain region. For example, you can create a draggable slider with an indicator that moves across a gauge.

- To make a text sprite editable with Lingo, set the text sprite's `editable` property to `TRUE`. For best results, set this property in a script attached to the sprite or the frame where the sprite is located. See **editable**.
- To make a sprite moveable with Lingo, set the `moveableSprite` sprite property to `TRUE`. For best results, set this property in a script attached to the sprite or the frame where the sprite is located. See **moveableSprite**.
- To restrict the registration point of a moveable sprite so it stays within the bounding rectangle of a second sprite, use the `constraint` sprite property. See **constraint**.
- To constrain a sprite along a horizontal or vertical path, use the `constrainH()` or `constrainV()` function. See **constrainH()**or **constrainV()**.

Checking which text is under the pointer with Lingo

Lingo can detect which text component in a text or field cast member is currently under the mouse pointer.

Use Lingo that applies to text and field cast members as follows:

- To detect which character in a text or field cast member is under the pointer, use the `pointToChar()` function. See **pointToChar()**.
- To detect which item in a text or field cast member is under the pointer, use the `pointToItem()` function. See **pointToItem()**.
- To detect which word in a text or field cast member is under the pointer, use the `pointToWord()` function. See **pointToWord()**.
- To detect which paragraph in a text or field cast member is under the pointer, use the `pointToParagraph()` function. See **pointToParagraph()**.

Use Lingo that applies only to text cast members as follows:

- To detect whether a specific point is in a hyperlink within a text cast member and is under the pointer, use the `pointInHyperlink()` function. See **pointInHyperlink()**.

Use Lingo that applies only to field cast members as follows:

- To detect which line in a field is under the pointer, use the `mouseLine` function. See **mouseLine**.
- To detect which word in a field is under the pointer, use the `mouseWord` function. See **mouseWord**.

Responding to rollovers with Lingo

You often want some action to occur when the user rolls the mouse pointer over a sprite or a particular place on the Stage. You can use Lingo to specify how the movie responds to such rollovers.

Director provides several event handlers that run when the pointer rolls over a sprite. Messages for each of these events are sent to the sprite script, the script of the cast member, the frame script, and then the movie script.

- To set up Lingo that runs when the mouse pointer enters a sprite's bounding rectangle, place the Lingo in an `on mouseEnter` event handler. See **on mouseEnter**.
- To set up Lingo that runs when the mouse pointer leaves a sprite's bounding rectangle, place the Lingo in an `on mouseLeave` event handler. See **on mouseLeave**.
- To set up Lingo that runs when the user clicks a sprite, rolls the pointer off the sprite, and then releases the mouse button, place the Lingo in an `on mouseUpOutside` event handler. See **on mouseUpOutside**.
- To set up Lingo that runs when the mouse pointer is within a sprite's bounding rectangle when the playback head enters the frame that contains the sprite, place the Lingo in an `on mouseWithin` event handler. See **on mouseWithin**.

  The `mouseWithin` event can occur repeatedly as long as the mouse pointer remains inside the sprite.

- To determine whether the cursor is over a specific sprite, use the `rollOver()` function. See **rollOver()**.

Finding mouse pointer locations with Lingo

Determining where the mouse pointer is on the Stage is a common need in Director.

To determine the mouse pointer's horizontal and vertical positions, use the `mouseH()` and `mouseV()` functions. See **mouseH** and **mouseV**.

The `mouseV()` function returns the distance, in pixels, between the mouse pointer and the upper left corner of the Stage. The `mouseH()` function returns the distance, in pixels, between the mouse pointer and the upper left corner of the Stage.

The statements `put the mouseH` and `put the mouseV` display the mouse pointer's location in the Message window.

For example, this handler directs the Message window to display the distance (in pixels) between the pointer and the upper left corner of the Stage:

```
on exitFrame
  put the mouseH
  put the mouseV
  go to the frame
end
```

Checking keys with Lingo

Lingo can detect the last key the user pressed.

- To obtain the ANSI value of the last key that was pressed, use the `key()` function. See **key()**.
- To obtain the keyboard's numerical (or ASCII) value for the last key pressed, use the `keyCode()` function. See **keyCode()**.

A common place for using `key` and `keyCode` is in an `on keyDown` handler. This instructs Lingo to check the value of `key` only when a key is actually pressed. For example, the following handler in a frame script sends the playback head to the next marker whenever the user presses Enter (Windows) or Return (Macintosh):

```
on keyDown
  if the key = RETURN then go to marker (1)
end
```

Equivalent cross-platform keys

Because of inherent differences between Windows and Macintosh keyboards, keys on Windows and Macintosh computers don't always correspond directly.

This can create confusion, because Lingo often uses the same term to refer to corresponding keys on Windows and Macintosh computers, even though the key's name differs on the two platforms.

The following table lists Lingo elements that refer to specific keys and the keys they represent on each platform.

Lingo term	Windows key	Macintosh key
RETURN	Enter	Return
commandDown	Control	Command
optionDown	Alt	Option
controlDown	Control	Control
ENTER	Enter key on the numeric keypad (during authoring, pressing Enter starts playing the movie)	Enter key on the numeric keypad (during authoring, pressing Enter starts playing the movie)
BACKSPACE	Backspace	Delete

Identifying keys on different keyboards

Characters can vary on different keyboards. Avoid possible confusion by identifying a character by its ASCII value.

- To obtain a character's ASCII value, use the `charToNum()` function. See **charToNum()**.

  For example, this statement finds the ASCII value for the letter *A* and displays it in the Message window:

  ```
  put charToNum("A")
  -- 65
  ```

- To find out which character corresponds to an ASCII value, use the `numToChar()` function. See **numToChar()**.

  For example, this statement finds the character that corresponds to the ASCII value 65. The result is the letter *A*:

  ```
  put numToChar(65)
  -- A
  ```

About animated color cursors

Director supports animated cursors. You can use any 8-bit bitmap source in your Director cast as an image in the cursor animation, automatically scale images, and generate masks for 16 x16 pixel and 32 x 32 pixel cursors. (Macintosh computers don’t support 32 x 32 pixel cursors.)

An animated cursor consists of a series of bitmap cast members. Each bitmap cast member is a frame of the cursor. You can control the rate at which Director plays the frames of an animated cursor. Using the Cursor Properties Editor, you designate one or more bitmap cast members as frames of a single cursor cast member.

Xtras that support animated cursors

The Director installation program places two animated color cursor files in the Media Support folder within the Director application's Xtras folder. The specific files depend on the platform you are using.

Windows	PowerPC	Purpose
Cursor Options.x32	Cursor Options	This file supports the creation of cursors while you author movies in Director. Do not distribute this file with projectors; it is not licensed for redistribution.
Cursor Asset.x32	Cursor Asset	Distribute this file with any movies or projectors that you create using the animated color cursors.

Requirements for animated color cursors

All cast members used for an animated color cursor must meet certain criteria:

- They must be bitmap cast members.
- They must have a color depth of 8 bits (256 colors).
- They must use only the first eight or the last eight colors that are in the standard System - Win palette. These provide the most predictable results when playing back across platforms. Other colors may not appear correctly.

The cast members need not be in sequence in the cast, and they need not be in the same cast.

A cursor's maximum size depends on the computer:

- In Windows 95 and 98, you can create cursors of either 16 x 16 pixels or 32 x 32 pixels (almost always 32 x 32 pixels, but some video cards may support only 16 x 16 pixels).
- In Windows NT, you can create cursors of 16 x 16 pixels or 32 x 32 pixels.
- On the Macintosh, you can create cursors of 16 x 16 pixels.

When you are creating cursors in the Cursor Properties Editor, Director dims any size option that is not available on your computer.

The 16 x16 pixel and 32 x 32 pixel sizes are the maximum sizes at which Director can display a cursor on the screen. The actual cast members you specify for the cursor can be larger than the maximum, and Director will scale the cast members to the appropriate size, maintaining the aspect ratio as it scales them. If you specify a cast member smaller than the maximum size, Director displays the cast member at its original size, without scaling. For example, if you select a maximum size of 16 x 16 pixels and then specify a cursor that is 12 x 14 pixels, Director displays the cursor at 12 x 14 pixels.

Creating an animated color cursor cast member

Before creating an animated color cursor cast member, make sure that the cast members you want to use in the cursor are stored in a cast linked to the movie. See **Managing external casts**.

To create an animated color cursor cast member:

1. Choose Insert > Media Element > Cursor.

   Director opens the Cursor Properties Editor, which you use to set up the cursor.

2. From the Cast pop-up menu, choose the cast that contains the cast member you want to add as a frame in your cursor.

   The cast members used for a single cursor can be stored in different casts.

3. Use the < and > buttons to find the cast member you want.

   As you click the buttons, the preview shows a thumbnail of the selected cast member. If you do not see the cast member you want, the cast member probably isn't a bitmap or has a color depth greater than 8 bits (256 colors). The Cursor Properties Editor shows only bitmaps that can be used in an animated color cursor.

   You can also type a cast member number in the Member box and press Tab; Director will select the cast member that has that number or the cast member with the number closest to it.

4. Select the cast member you want and click Add.

   You see the cast member in the Cursor Frames preview area. The Frame X of Y field shows where the cast member falls within an animated series of cursor frames.

5. Repeat steps 2 through 4 until you have added all the cast members for the cursor.

   In the Cursor Frames area, you can use the < and > buttons to review the order of the cursor frames. Click the Remove button to delete the currently selected frame from the cursor (this deletes the cast member only from the cursor animation, not from the cast).

6. In the Interval field, specify the number of milliseconds that elapse between each frame of the cursor animation.

   This interval affects all frames of the cursor and cannot vary for different frames. The cursor frame rate is independent of the frame rate set for the movie using the tempo channel or the `puppetTempo` Lingo command.

Note: By inserting the same bitmap in multiple frames of the cursor, you can create the illusion of variable-rate cursor animation.

7. In the Hotspot Position fields, specify the location of the mouse pointer's active point.

   Director uses this point to track the mouse pointer's position on the screen. For example, Director uses this point's location when it returns values for the `mouseH()` and `mouseV()` functions. The hotspot also determines the point where a rollover occurs.

   The first field specifies the horizontal (*x*) location, and the second field specifies the vertical (*y*) location. The upper left pixel is location 0,0. In a 16 x 16 pixel cursor, the lower right pixel is location 15,15. You can't enter a point that is beyond the bounds of the cursor.

8 Click one of the Size options to specify the maximum size of the cursor.

If a Size option is dimmed, your computer does not allow you to create cursors of that size.

9 Select the Automask option if you want the white pixels of the cursor frames to be transparent.

Note: The Automask option makes all white pixels transparent. If you want some white pixels to be opaque, you can't use white for those pixels, but you can achieve the same effect by instead using the lightest shade of gray available in the system palette.

10 Click OK to close the Cursor Properties Editor.

After you create a cursor cast member, use Lingo to switch to the cursor in a movie. See **Using an animated color cursor in a movie**.

Using an animated color cursor in a movie

Once you have added an animated color cursor to the cast, use Lingo to switch to the animated cursor just as you would any other cursor. You can set up an animated cursor as the movie's cursor or a sprite's cursor.

To switch to an animated color cursor, use this command:

```
cursor (member whichCursorCastMember)
```

For *whichCursorCastMember*, substitute a cast member name (surrounded by quotation marks) or a cast member number. For example, this sprite script changes the cursor to the cast member named myCursor when the cursor is over the sprite:

```
on mouseEnter
  cursor (member "myCursor")
end
```

To reset the cursor to the regular arrow cursor, specify a cursor type of -1 (and do not use parentheses). This sample sprite script resets the cursor:

```
on mouseLeave
  cursor -1
end
```

Note: Do not place an animated color cursor cast member on the Stage.

For more information, see **cursor (command)**.

Movies in a window: Overview

Director can play several movies simultaneously by creating windows that additional movies can play in. A movie in a window (MIAW) is a distinct Director movie that retains all its interactivity.

You can use a MIAW simply to play another movie in a separate window while the main movie plays on the Stage. In addition, movies in windows and the main movie can communicate and interact with each other. This lets you create a variety of interactive features, such as an interactive portfolio, a control panel for a second movie or digital video, or a status display window.

Here's the typical workflow for using a movie in a window:

- Create and set up the window.
- Assign a movie to the window.
- Open the window and play the movie.
- Delete the window when the reason for playing the movie no longer applies.

When creating a MIAW, first decide how you want it to behave and work. For example, decide how you want to display it, how users should be able to move the window around the screen and dismiss it, and how the window should appear. You can specify the window's size and whether the window is visible, has a frame and title, or is in front of or behind other windows on the screen.

Shockwave doesn't support MIAWs. Use MIAWs only with movies that you intend to distribute as projectors (see **About distribution formats** and **About projectors**). However, you can use Lingo to have a Shockwave movie target a URL in a browser window. See **Jumping to a URL**.

Creating a MIAW using Lingo

In Lingo, you create a MIAW by specifying a window's rectangle on the Stage and then specifying the file name for the movie assigned to the window. You can also make the window visible, change its type, set its title, and set the window's size and location.

The simplest way to create a MIAW is to simply open a window for an existing movie.

To create a new MIAW by opening a window for an existing movie:

Follow this example:

```
open window("movieName")
```

This statement creates a window, assigns it the movie `movieName`, and opens it on the Stage at the location where `movieName` was originally authored. At this point, you can use the commands discussed in the rest of this chapter to set various attributes of the MIAW.

You can also use a movie's file name as the argument for the `open window` command. This approach assigns that movie to a window and instructs Director to use the file name as the window title.

To create a MIAW using a file name and the Open Window command:

Follow this example:

```
on beginNewMovie theMovie
  global newWindow
  set newWindow to window theMovie
  set newWindow.titleVisible to FALSE
  open newWindow
end beginNewMovie
```

This version of the handler uses the movie's rectangle to determine the size of the window's rectangle.

You can also assign a MIAW to a variable, which makes it easier to write the handler and reuse it.

To assign a MIAW to a variable:

Follow this example:

```
on beginNewMovie theMovie
  global newWindow
  set newWindow to window "The Big Picture"
  set newWindow.rect to rect(0, 0, 250, 200)
  set newWindow.filename to theMovie
  set newWindow.titleVisible to FALSE
  open newWindow
end beginNewMovie
```

The variable `newWindow` contains a new window named The Big Picture. The handler specifies the coordinates of a rectangle, instructs Director to use that rectangle as the window named The Big Picture, and then assigns a movie file to the window. The handler makes the title bar at the top of the window invisible and then opens the window.

Opening and closing a MIAW

Use the commands in this section to open and close movies in windows.

Opening a MIAW

To open a MIAW, use the `open window` command. See **open window**.

Unless Lingo explicitly preloads the movie, Director doesn't load the movie into memory until the window is first opened, which can cause a noticeable pause. To load the first frame of the movie, use `preLoadMovie`.

You can specify other window characteristics before or after you open the window.

Closing a MIAW

You can close the window for a MIAW but leave the movie in memory, or you can close the MIAW and remove the movie from memory when it's no longer in use.

- If you leave a MIAW in memory, you'll get better performance if the window is reopened; however, the movie still takes up space in memory. You may want to use this option if you expect a MIAW to be reopened after it initially runs, or if other windows or global variables refer to the MIAW.
- If you remove a MIAW from memory, performance will slow down if the window is reopened, because the movie has to reload; however, memory is freed up until the movie is reloaded. You may want to use this option if you don't expect a MIAW to be reopened after it initially runs, or if you want to optimize memory on the computer running the MIAW.

To close a MIAW but keep it in memory:

Use the `close window` command. After the window is closed, the window becomes invisible, but the movie continues playing. See **close window**.

To close a MIAW and remove it from memory:

Use the `forget window` command. The window is closed and the movie is removed from memory. Use this command only if no other window or global variables still refer to the MIAW. See **forget window**.

Setting the window type for a MIAW

You can choose from seven styles of windows:

- Four document window styles: the standard document window, a document window with a zoom box and variable resize box, a document window with the variable resize box disabled, and a document window without a resize box
- An alert box style
- A plain box style
- A curved-border style

To specify the window type for a MIAW:

Assign a value for the `windowType` property.

Different numerical values for the `windowType` property specify different types of window styles. When you don't specify a window type, Director uses a moveable, sizable window without a zoom box and with a title bar, which is type -1. You can set the title property only for windows of type -1.

In most cases, it's best to specify window settings before you actually open the window, to avoid delays as the window redraws.

See **windowType**.

Setting the window size and location for a MIAW

Setting the screen coordinates for a MIAW lets you control how large the window is and where it appears. Setting the coordinates before the movie appears controls the initial position of the window; setting them after the window appears moves the window.

To specify the screen coordinates for a MIAW:

Set the `rect` property to the coordinates of the location where you want the window to appear, choosing from the following options:

- Define the coordinates as a rectangle in the order left, top, right, and bottom, as in this statement:

```
set window.rect "Sample" = [0, 0, 200, 300]
```

- Use the `rect()` function to define the window rectangle's four coordinates, as in these statements:

```
set aRect = rect(0, 0, 200, 300)
set window.rect "Sample" = aRect
```

For convenience, assign the coordinates to a variable and then use the variable in the statements you write.

See **rect (window)** and **rect()**.

Cropping and scaling a MIAW

You can use Lingo to crop or scale a MIAW.

To crop a MIAW:

Set the `rect` window property to an area smaller than the MIAW. See **rect (window)**.

To scale a MIAW:

Set the `drawRect` property to coordinates smaller than the MIAW's original size and apply the position to the window, as shown in the following example:

```
set aRect = [0, 0, 200, 300]
set window("Sample").drawRect = aRect -- sets window size to ¬
200 x 300
set window("Sample").drawRect = aRect/2 -- scales the window ¬
to half its original size
```

When the `drawRect` property specifies a window rectangle that is smaller than the movie, the window appears in the upper left corner, and the movie is compressed to fit within the window. See **drawRect**.

Controlling the appearance of a MIAW

You can use Lingo commands and properties to control whether a window is visible, is in front of or behind other windows, and has a title.

To specify whether the window is visible:

Set the window's `visible` window property. To avoid a potential time lag when the window opens, use `preLoadMovie` to preload the movie before it's needed and then open the window when it needs to be visible. See **visible (window property)**.

To control whether a movie appears in front of or behind other windows:

Use the moveToFront and moveToBack commands. See **moveToFront** and **moveToBack**.

To assign a title to a window:

Set the `title` window property. See **title**.

Listing the current movies in windows

The `windowList` property displays a list of all known MIAWs in the main movie.

For example, the following statement displays a list of current MIAW names in the Message window.

```
put the windowList
```

See **windowList**.

Controlling interaction between MIAWs

Movies can interact with each other by sending Lingo messages back and forth. This lets movies share current values for variables, share information about current conditions, and send each other Lingo instructions.

Global variables can be declared in the main movie (the Stage) or in a MIAW. No matter where they are declared, they are available to the main movie and to all movies playing in windows. For more information about global variables, see **Using global variables**.

At times, you may want only one movie to respond when the user clicks the mouse or types on the keyboard. To control when Director can respond to any events that occur outside a window, set the `modal` window property. When a window's `modal` property is set to TRUE, no other window, including the Stage, can respond to events such as mouse clicks and keystrokes.

To have a MIAW send a Lingo statement:

Use the `tell` command. See **tell**.

When using the `tell` command, be sure to specify the MIAW to which the instructions are directed. When you want a MIAW to send a Lingo message to the main movie, use `the stage` to refer to the main movie. For example, the statement `tell the stage to go to "Help"` instructs the main movie to go to the frame marked Help in the main movie.

To have a MIAW open another MIAW:

In Lingo, only the main movie (the Stage) can open a MIAW. Therefore, to have one MIAW open another MIAW, you must use the `tell` command in the running MIAW to tell the Stage to open another MIAW.

For example, this statement in a MIAW tells the Stage movie to open the movie menuMovie in its own window:

```
tell the stage to open window "menuMovie"
```

See **tell**.

Controlling events involving MIAWs

Lingo provides event handlers for typical events that can occur while a MIAW is playing, such as the movement of a window by the user. Such a handler is a good place for instructions that you want to run in response to an event that involves a window.

For example, to cause a sound to play whenever the user closes a MIAW, use the `queue()` and `play()` functions in an `on closeWindow` handler in a movie script within the movie that plays in the window. The `on closeWindow` handler will run whenever the MIAW that contains the handler closes.

See **Movie in a window** in the “Lingo by Feature” section in the *Lingo Dictionary*.

Parent scripts: Overview

Parent scripts provide the advantages of object-oriented programming within Director. These advantages include the ability to write less code and use simpler logic to accomplish tasks in Lingo. You can use parent scripts to generate script objects that behave and respond similarly yet can still operate independently of each other.

Lingo can create multiple copies (or instances) of a parent script. Each instance of a parent script is called a child object. You can create child objects on demand as the movie plays. Director doesn't limit the number of child objects that can be created from the same parent script. You can create as many child objects as the computer's memory can support.

Director can create multiple child objects from the same parent script, just as Director can create multiple instances of a behavior for different sprites. You can think of a parent script as a template and of child scripts as implementations of the parent template.

This chapter describes the basics of how to write parent scripts, and create and use child objects, and it provides script examples. It doesn't teach fundamental object-oriented programming concepts; however, to use parent scripts and child objects successfully, you must understand object-oriented programming principles. For an introduction to the basics of object-oriented programming, see one of the many third-party books on that subject.

Similarity with other object-oriented languages

If you are familiar with an object-oriented programming language such as Java or C++, you may already understand the concepts that underlie parent scripting but know them by different names.

Terms that Director uses to describe parent scripts and child objects correspond to the following common object-oriented programming terms:

Parent scripts correspond to classes.

Child objects correspond to instances.

Property variables correspond to instance variables or member variables.

Handlers correspond to methods.

Ancestor scripts correspond to the Super class or base class.

Parent script and child object basics

A parent script is a set of handlers and properties that will define a child object; it is not a child object itself. A child object is a self-contained, independent instance of a parent script. Children of the same parent have identical handlers and properties, so child objects in the same group can have similar responses to events and messages.

Typically, parent scripts are used to build child objects that make it easier to organize movie logic. These child objects are especially useful when a movie requires the same logic to be run several times concurrently with different parameters. You can also add a child object to a sprite's `scriptInstanceList` or the `actorList` as a way to control animation.

Because all the child objects of the same parent script have identical handlers, those child objects respond to events in similar ways. However, because each child object maintains independent values for the properties defined in the parent script, each child object can behave differently than its sibling objects— even though they are instances of the same parent script.

For example, you can create a parent script that defines child objects that are editable text fields, each with its own property settings, text, and color, regardless of the other text fields' settings. By changing the values of properties in specific child objects, you can change any of these characteristics as the movie plays without affecting the other child objects based on the same parent script.

Similarly, a child object can have a property set to either `TRUE` or `FALSE` regardless of that property's setting in sibling child objects.

Differences between child objects and behaviors

While child objects and behaviors are similar in that they both can have multiple instances, they have some important differences as well. The main difference between child objects and behaviors is that behaviors are associated with locations in the Score because they are attached to sprites. Behavior objects are automatically created from initializers stored in the Score as the playback head moves from frame to frame and encounters sprites with attached behaviors. In contrast, child objects from parent scripts must be created explicitly by a handler.

Behaviors and child objects differ in how they become associated with sprites. Director automatically associates a behavior with the sprite that the behavior is attached to, but you must explicitly associate a child object with a sprite.

Ancestor basics

Parent scripts can declare ancestors, which are additional scripts whose handlers and properties a child object can call on and use.

Ancestor scripting lets you create a set of handlers and properties that you can use and reuse for multiple parent scripts.

A parent script makes another parent script its ancestor by assigning the script to its `ancestor` property. For example, the following statement makes the script What_Everyone_Does an ancestor to the parent script in which the statement occurs:

```
set ancestor to new(script "What_Everyone_Does")
```

When handlers and properties are not defined in a child object, Director searches for the handler or property in the child's ancestors, starting with the child's parent script. If a handler is called or a property is tested and the parent script contains no definition for it, Director searches for a definition in the ancestor script. If a definition exists in the ancestor script, that definition is used.

A child object can have only one ancestor at a time, but that ancestor script can have its own ancestor, which can also have an ancestor, and so on. This lets you create a series of parent scripts whose handlers are available to a child object.

See **ancestor**.

Writing a parent script

A parent script contains the Lingo needed to create child objects and define their possible actions and properties. First you need to decide how you want the child objects to behave. Then you can write a parent script that does the following:

- Optionally declares any appropriate property variables; these variables represent properties for which each child object can contain a value independent of other child objects. See **Parent script and child object basics**.
- Sets up the initial values of the child objects' properties and variables in the `on new` handler.
- Contains additional handlers that control the child objects' actions.

Declaring property variables

Each child object created from the same parent script initially contains the same values for its property variables. A property variable's value belongs only to the child object it's associated with. Each property variable and its value persists as long as the child object exists. The initial value for the property variable is typically set in the `on new` handler; if it's not set, the initial value is `VOID`.

To declare a property variable:

Use the `property` keyword at the beginning of the parent script. See **property**.

To set and test property variables from outside the child object:

Set and test property variables in the same way you would any other property in Lingo, by using the syntax `the` *`propertyName`* `of` *`whichObject`* or *`whichObject.propertyName`*.

This statement sets the `speed` property of the object `car1`:

```
car1.speed = 55
```

Creating the on new handler

Each parent script typically uses an `on new` handler. This handler creates the new child object when another script issues a `new(script` *`parentScriptName`*`)` command, which tells the specified parent script to create a child object from itself. The `on new` handler in the parent script can also set the child object's initial property values, if you want. The `on new` handler always starts with the phrase `on new`, followed by the `me` variable and any arguments being passed to the new child object. See **new()**.

The following is a sample `on new` handler:

```
property spriteNum
on new me, aSpriteNum
  spriteNum = aSpriteNum
  return me
end
```

Adding other handlers

You determine a child object's behavior by including in the parent script the handlers that produce the desired behavior. For example, you could add a handler to the code above to make the sprite change color.

The following parent script defines a value for the property `spriteNum` and contains a second handler that will change the `forecolor` property of the sprite.

```
property spriteNum
on new me, aSpriteNum
  spriteNum = aSpriteNum
  return me
end
on changeColor me
  spriteNum.forecolor = random(255)
end
```

Using the me variable

Typically, one parent script creates many child objects, and each child object contains more than one handler. The term `me` is a special parameter variable. It must always be the first parameter variable stated in every handler definition in a parent script.

The `me` variable tells the handlers in the child object that they are to operate on the properties of that object and not on the properties of any other child object. This way, when a handler within a child object refers to properties, the handler will use its own child object's values for those properties.

This is why it is always important to define `me` as the first parameter for parent scripts and to pass the same parameter if you need to call other handlers in the same parent script, since these will be the handlers in each of the script's child objects.

When referring to properties defined in ancestor scripts, you must use the `me` parameter as the source of the reference. This is because the property, while defined in the ancestor script, is nevertheless a property of the child object. For example, these statements both use `me` to refer to an object and access properties defined in an ancestor of the object:

```
--access ancestor property
x = me.y
```

or

```
x = the y of me
```

Because the `me` variable is present in each handler of the child object, it indicates that all the handlers control that same child object.

See **me**.

Creating a child object

Child objects exist entirely in RAM; they are not saved with a movie. Only parent and ancestor scripts exist on disk.

To create a new child object, you use the `new` function and assign the child object a variable name or position in a list so you can identify and work with it later.

To create a child object and assign it to a variable, use the syntax:

```
set variableName = new(script "scriptName", argument1¬
, argument2, argument3...)
```

where *`scriptName`* is the name of the parent script and *`argument1`*, *`argument2`*, *`argument3`*... are any arguments you are passing to the child object's `on new` handler.

The `new()` function creates a child object whose ancestor is *`scriptName`*. It then calls the `on new` handler in the child object with the specified arguments.

You can issue a `new` statement from anywhere in a movie. You can customize the child object's initial settings by changing the values of the arguments passed with the `new` statement.

Each child object requires only enough memory to record the current values of its properties and variables and a reference to the parent script. Because of this, in most cases you can create and maintain as many child objects as you require.

You can produce additional child objects from the same parent script by issuing additional `new` statements.

To create child objects without immediately initializing their property variables, use the `rawNew()` function. The `rawNew()` function does this by creating the child object without calling the parent script's `on new` handler. In situations where large numbers of child objects are needed, `rawNew()` allows you to create the objects ahead of time and defer the assignment of property values until each object is needed.

This statement creates a child object from the parent script Car without initializing its property variables and assigns it to the variable `car1`:

```
car1 = script("Car").rawNew()
```

To initialize the properties of one of these child objects, call its `on new` handler:

```
car1.new
```

Checking child object properties

You can check the values of specific property variables in individual child objects by using a simple *objectName.PropertyName* syntax. For example, this statement assigns the variable `x` the value of the `carSpeed` property of the child object in the variable `car1`:

```
x = car1.carSpeed
```

Querying object properties from outside the objects themselves can be useful for getting information about groups of objects, such as the average speed of all the car objects in a racing game. You might also use the properties of one object to help determine the behavior of other objects that are dependent on it.

In addition to checking the properties you assign, you can check whether a child object contains a specific handler or find out which parent script an object came from. This is useful when you have objects that come from parent scripts that are similar but that have subtle differences.

For example, you may want to create a scenario in which one of several parent scripts might be used to create a child object. You can then determine which parent script a particular child object came from by using the `script()` function, which returns the name of an object's parent script.

These statements check whether the object `car1` was created from the parent script named Car:

```
if car1.script = script("Car") then
   beep
end if
```

You can also get a list of the handlers in a child object by using the `handlers()` function, or check whether a particular handler exists in a child object by using the `handler()` function.

This statement places a list of the handlers in the child object `car1` into the variable `myHandlerList`:

```
myHandlerList = car1.handlers()
```

The list would look something like this:

```
[#start, #accelerate, #stop]
```

These statements use the `handler()` function to check whether the handler `on accelerate` exists in the child object `car1`:

```
if car1.handler(#accelerate) then
   put "The child object car1 contains the handler named ¬
   on accelerate."
end if
```

Removing a child object

You can remove a child object from a movie by setting all variables that contain a reference to the child object to another value. If the child object has been assigned to a list, such as `actorList`, you must also remove the child object from the list. (The `actorList` property is useful for tracking and manipulating the child objects in a movie. For details, see **Using actorList**.)

To remove a child object and the variables that refer to it:

Set each variable to `VOID`.

Director deletes the child object when there are no more references to it. For example, if `ball1` contains the only reference to a specific child object, then the `statement set ball1 = VOID` deletes the object from memory.

To remove an object from actorList:

Use the `delete` command to delete the item from the list. See **delete**.

Using scriptInstanceList

You can use the `scriptInstanceList` property to dynamically add new behaviors to a sprite. Normally, `scriptInstanceList` is the list of behavior instances created from the behavior initializers defined in the Score. If you add child objects created from parent scripts to this list, the child objects receive the messages sent to other behaviors.

For example, this statement adds a child object to the `scriptInstanceList` property of sprite 10:

```
add sprite(10).scriptInstanceList, new(script "rotation", 10)
```

This is a possible parent script that the statement refers to:

```
-- parent script "rotation"
property spriteNum
on new me, aSpriteNum
  spriteNum = aSpriteNum
  return me
end
on prepareFrame me
  sprite(spriteNum).rotation = sprite(spriteNum).rotation + 1
end
```

When a child object is added to `scriptInstanceList`, you must initialize the child object's `spriteNum` property. Typically you do this from a parameter passed in to the `on new` handler.

Note: The `beginSprite` message is not sent to dynamically added child objects.

Using actorList

Lingo can set up a special list of child objects (or any other objects) that receives its own message each time the playback head enters a frame or the `updateStage` command updates the Stage.

The special list is `actorList`, which contains only objects that have been explicitly added to the list. See **actorList**.

The message is the `stepFrame` message that is sent only when the playback head enters a frame or the `updateStage` command is used. See **on stepFrame**.

Objects in `actorList` receive a `stepFrame` message instead of an `enterFrame` message at each frame. If the objects have an `on stepFrame` handler available, the Lingo in the `on stepFrame` handler runs each time the playback head enters a new frame or the `updateStage` command updates the Stage.

Some possible uses of `the actorList` and `stepFrame` are to animate child objects that are used as sprites or to update a counter that tracks the number of times the playback head enters a frame.

An `on enterFrame` handler could achieve the same results, but the `actorList` property and `on stepFrame` handler are optimized for performance in Director. Objects in `the actorList` respond to `stepFrame` messages more efficiently than to an `enterFrame` message or to a custom message sent after an `updateStage` command.

To add an object to `actorList`, use the following statement:

```
add the actorList, theObject
```

The object's `on stepFrame` handler in its parent or ancestor script will then run automatically each time the playback head advances. The object will be passed as the first argument (that is, the `me` argument) to the `on stepFrame` handler.

Director doesn't clear the contents of `actorList` when branching to another movie, which can cause unpredictable behavior in the new movie. If you don't want child objects in the current movie to be carried over into the new movie, insert a statement that clears `actorList` in the `on prepareMovie` handler of the new movie.

To clear child objects from `actorList`:

Set `actorList` to `[ ]`, which is an empty list.

For more information, see **actorList** and **on stepFrame**.

Creating timeout objects

You can create a timeout object—a script object that acts like a timer and sends a message when the timer expires. This is useful for scenarios that require specific things to happen at regular time intervals or after a particular amount of time has elapsed.

Timeout objects can send messages that call handlers inside child objects or in movie scripts. You create a timeout object by using the `new()` function. You must specify a name for the object, a handler to be called, and the frequency with which you want the handler to be called. Once a timeout object is created, Director keeps a list of currently active timeout objects, called `timeOutList`.

To create timeout objects, use the following syntax:

```
variableName = timeOut("theName").new(integerMilliseconds¬
, #handlerName, targetObject)
```

This statement uses the following elements:

- *`variableName`* is the variable you are placing the timeout object into.
- `timeOut` indicates which type of Lingo object you are creating.
- *`theName`* is the name you give to the timeout object. This name will appear in the `timeOutList`. It is the `#name` property of the object.
- `new` is the Lingo function that creates a new object.
- *`integerMilliseconds`* indicates the frequency with which the timeout object should call the handler you specify. This is the `#period` property of the object. A value of 2000 will call the specified handler every 2 seconds.
- *`#handlerName`* is the name of the handler you want the object to call. This is the `#timeOutHandler` property of the object. You represent it as a symbol by preceding the name with the # sign. For example, a handler called `on accelerate` would be specified as `#accelerate`.
- *`targetObject`* indicates which child object's handler should be called. This is the `#target` property of the object. It allows specificity when many child objects contain the same handlers. If you omit this parameter, Director will look for the specified handler in the movie script.

This statement creates a timeout object named `timer1` that will call the `on accelerate` handler in the child object `car1` every 2 seconds:

```
myTimer = timeOut("timer1").new(2000, #accelerate, car1)
```

To determine when the next timeout message will be sent from a particular timeout object, check its `#time` property. The value returned is the point in time, in milliseconds, when the next timeout message will be sent.

This statement determines the time when the next timeout message will be sent from the timeout object `timer1` and displays it in the Message window:

```
put timeout("timer1").time
```

Using timeOutList

When you begin creating timeout objects, you can use `timeOutList` to check the number of timeout objects that are active at a particular moment.

The following statement sets the variable `x` to the number of objects in `timeOutList`. See **count()**.

```
x = (the timeOutList).count
```

You can also refer to an individual timeout object by its number in the list.

The following statement deletes the second timeout object in `timeOutList`. See **forget()**.

```
timeOut(2).forget
```

Relaying system events with timeout objects

When you create timeout objects that target specific child objects, you enable those child objects to receive system events. Timeout objects relay these events to their target child objects. The system events that can be received by child objects include `prepareMovie`, `startMovie`, `stopMovie`, `prepareFrame`, and `exitFrame`. By including handlers for these events in child objects, you can make the child objects respond to them for whatever purposes you see fit. System events received by child objects are also received by movie scripts, frame scripts, and other scripts designed to respond to them.

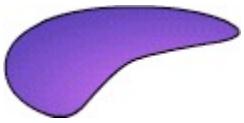
This parent script contains a handler for the system event `exitFrame` as well as a custom handler:

```
property velocity
on new me
  velocity = random(55)
end
on exitFrame
  velocity = velocity + 5
end
on slowDown mph
  velocity = velocity - mph
end
```

For information on specific timeout properties, see `timeout()` in the *Lingo Dictionary*.

Vector shapes and bitmaps: Overview

Vector shapes and bitmaps are the two main types of graphics used with Director. A vector shape is a mathematical description of a geometric form that includes the thickness of the line, the fill color, and additional features of the line that can be expressed mathematically. A bitmap defines an image as a grid of colored pixels, and it stores the color for each pixel in the image. A bitmap typically requires more RAM and disk space than a comparable vector shape. If not compressed, bitmaps take longer than vector shapes to download from the Internet. Fortunately, Director offers compression control to reduce the size of bitmaps in movies that you package to play on the Web. For more information about bitmap compression, see **Compressing bitmaps**.



Vector shape

- *Bitmap*

A vector shape is most appropriate for a simple, smooth, clean-looking image. It typically includes less detail than a bitmap, but you can resize it without distortion. You can use Lingo to dynamically control a vector shape. You can create a vector shape entirely with Lingo or modify an existing one as the movie plays. Because vector shapes are stored as mathematical descriptions, they require less RAM and disk space than an equivalent bitmap image and they download faster from the Internet.

You can create vector shapes in the Director Vector Shape window by defining points through which a line passes. The shape can be a line, a curve, or an open or closed irregular shape that can be filled with a color or gradient.

Bitmaps are suited for continuous tone images, like photographs. You can easily make minute changes to a bitmap by editing single pixels, but resizing the image can cause distortion as pixels are redistributed. Anti-aliasing is a Director feature that blends the bitmap's colors with background colors around the edges to make the edge appear smooth instead of jagged.

You can create bitmaps in the Paint window or import them from any of the popular image editors in most of the popular formats, including GIF and JPEG. Director can also import bitmaps with alpha channel (transparency) data and animated GIFs. The Paint window includes a variety of tools for editing and applying effects to bitmaps.

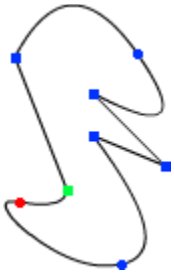
Drawing vector shapes

You create vector shapes with drawing tools in the Vector Shape window. You can use the Pen tool to create irregular shapes, or use shape tools to create rectangles and ellipses. A vector shape can include multiple curves, and you can split and join the curves. Shape properties such as fill color, stroke color, and stroke width are set at the cast-member level and not for individual curves.

When you create vector shapes, you create vertices, which are fixed points. You can also create handles, which are points that determine the degree of curvature between vertices. These curves are known as Bézier curves. A vertex without a handle creates a corner.

As you draw vector shapes, control handles appear on the vertices: round curve points for vertices with handles and square corner points for vertices without handles.

- The first vertex in a curve is green.
- The last vertex in a curve is red.
- All other vertices are blue.
- Unselected vertices are solid.
- Selected vertices are unfilled.



For an animated introduction to vector shapes, watch the **Vector Shape** movie.

To open the Vector Shape window:

Choose Window > Vector Shape.

Zooming in and out in the Vector Shape window

You can use the Magnify tool or the Zoom commands on the View menu to zoom in or out at four levels of magnification.

To zoom in or out, do one of the following:

- Choose View > Zoom and then choose the level of magnification.
- Right-click (Windows) or Control-click (Macintosh) and choose Zoom In or Zoom Out from the context menu.
- Press Control + the plus (+) key (Windows) or Command + the plus (+) key (Macintosh) to zoom in, or Control + the minus (-) key (Windows) or Command + the minus (-) key (Macintosh) to zoom out.

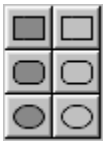
To return to normal view:

Choose View > Zoom > 100%.

Using vector shape drawing tools

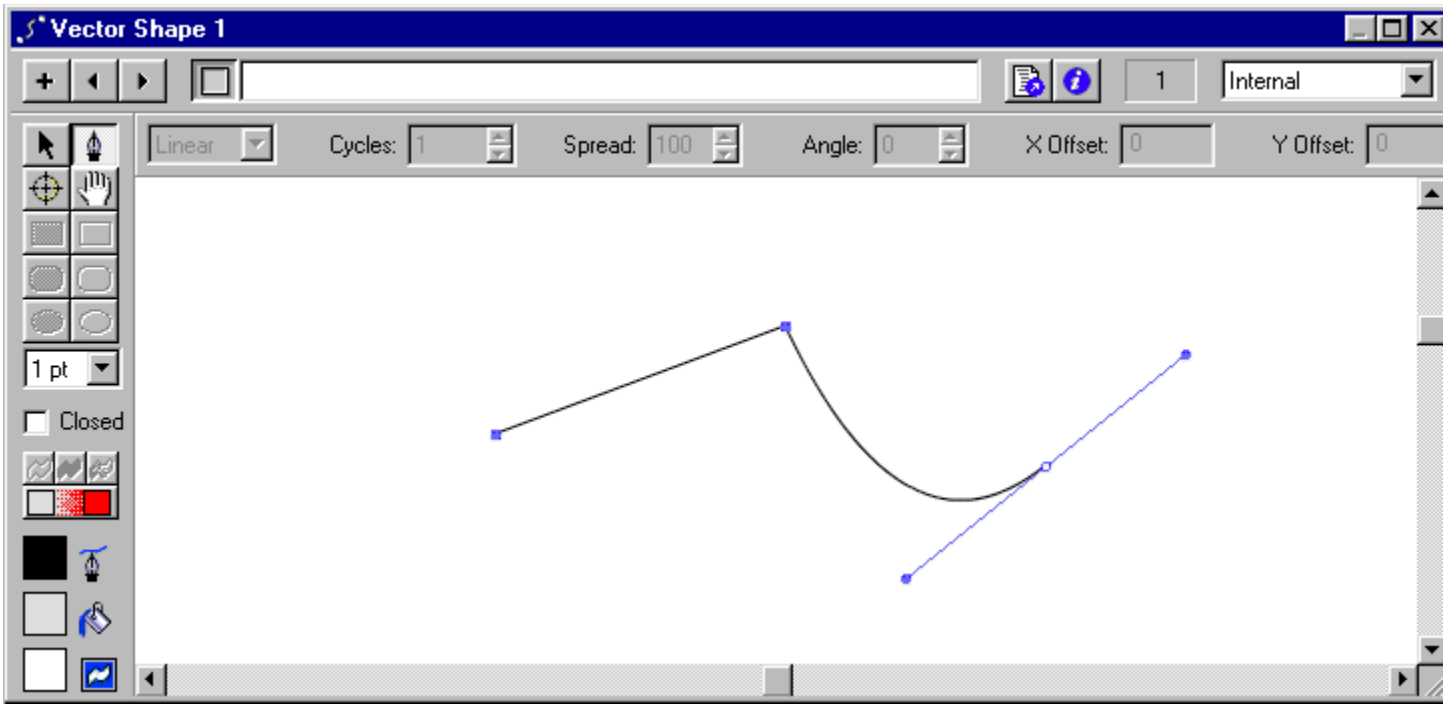
You use the tools in the Vector Shape window to draw free-form shapes or geometric figures. You can define a shape with the Pen tool by creating curve or corner points through which a line passes.

To draw regular shapes, you use the Rectangle, Rounded Rectangle, and Ellipse tools.



To create a vector shape using the Pen tool:

1. In the Vector Shape window, click the New Cast Member button.
2. Click the Pen tool and begin to draw:



- To create a corner point, click once.
- To create a curve point, click and drag. Dragging creates control handles that define how the line curves through the point you define.
- To constrain a new point to vertical, horizontal, or a 45° angle, hold down Shift while clicking.

To draw using a basic shape tool:

1. In the Vector Shape window, click the New Cast Member button.
2. Select the Filled or Unfilled Rectangle, Rounded Rectangle, or Ellipse tool.
3. Hold down the mouse button to start a shape, drag to draw, and release the mouse button to end the shape.

   To constrain a rectangle to a square, or to constrain an ellipse to a circle, hold down Shift while dragging.

To select a vertex or vertices:

- To select one vertex, select the Arrow tool and click the vertex.
- To select multiple vertices, either select the Arrow tool and hold Shift while clicking the vertices, or click and drag a selection rectangle over the vertices (marquee-select).
- To select all the vertices in a curve, select the Arrow tool and double-click one of the vertices in the curve.

To create multiple curves, do one of the following:

- If using the Pen tool, double-click the last vertex drawn. The next vertex will start a new curve.
- With no vertices selected, use the Pen tool to start a new curve.
- To create two separate curves from one, select two adjacent vertices in a curve and choose Modify > Split Curve.
- If the current shape is empty or closed, select one of the shape tools and draw a new shape.

Note: If you create multiple shapes in the Vector Shape window, Director treats all of the shapes as one if you change shape attributes. If, for example, you create ten open shapes in one Vector Shape window and select Close, Director closes all ten shapes.

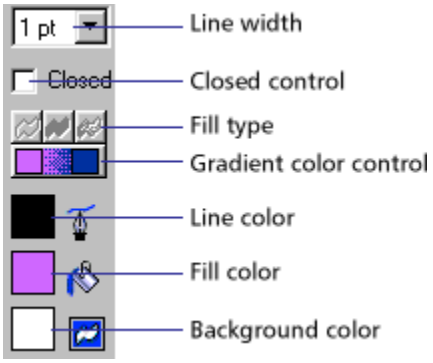
Choosing fill and line settings for vector shapes

You can use either controls in the Vector Shape window or Lingo to choose a vector shape's fill color, line width and color, and background color. The background is the area outside of a vector shape but within the cast member's bounding rectangle.

Because a vector shape is a single object, you don't need to select any part of the vector shape to make the following changes.

To choose the fill and line settings:

1. Open a vector shape in the Vector Shape window.
2. Choose fill and line settings using the appropriate controls at the left of the window.



- To choose the fill color, choose a color from the Fill Color menu.
- To choose the line color, choose a color from the Line Color menu.
- To set the line width, choose a point size option from the Line Width menu.
- To set the background color, choose a color from the Background Color menu. Choosing a background color that matches the color of the background results in better performance than using Background Transparent ink.

Specifying vector shape fills and strokes with Lingo

You can use Lingo to specify a vector shape's fills and strokes.

To specify the strokes that form a vector shape with Lingo:

Set the strokeColor and strokeWidth cast member properties. See **strokeColor** and **strokeWidth**.

To specify a vector shape's fill with Lingo:

Set the fillColor, fillMode, fillOffset, and fillScale cast member properties. See **fillColor**, **fillMode**, **fillOffset**, and **fillScale**.

Editing vector shapes

To edit vector shapes, you use the Vector Shape window. You change vector shapes by moving, adding, or deleting control points and changing the way they control curves. You can also change the way a vector shape is placed on the Stage by moving its registration point using either the Vector Shape window or Lingo.

To adjust the outline of a vector shape:

1. Open a vector shape in the Vector Shape window.
2. Click the Arrow tool and make any of the following changes:

- To move a curve or corner point, drag it to any location.
- To move multiple points, Shift-click or drag a selection rectangle around all the points you want to move and then drag any one of the selected points.
- To drag a single curve within a shape, select the Arrow tool and drag the curve. If the curve is filled, you can click anywhere within the filled area and drag the curve.
- To adjust a curve, select a curve point and drag a control handle.

  By default, the two control handles remain at a 180° angle from each other. If you want to drag one control handle independently from the other one, hold down Control (Windows) or Command (Macintosh) when you drag it. To constrain the control handles to vertical, horizontal, or a 45° angle, hold down Shift as you move them.
- To change a corner point to a curve point, Alt-click (Windows) or Option- click (Macintosh) and drag away from the handle to extend a control handle.
- To change a curve point to a corner point, drag the control handles directly over the curve point.
- To delete a point, select the point and press Backspace or Delete.
- To move the window view without using the scroll bars, click the Hand tool and drag anywhere inside the shape.

To add a point in the middle of a shape:

1. Open a vector shape in the Vector Shape window.
2. Click the Pen tool.
3. If the shape is closed, move the pointer over a line until it changes and then click the mouse button. If the shape is open, hold down Alt (Windows) or Option (Macintosh) and move the pointer over a line until it changes; then click the mouse button.

To add a new point that is connected to a certain end point:

1. Click the Arrow tool and select an end point.
2. Click the Pen tool and then click the location where you want to add the next point.

To join two curves:

1. Select a vertex in each curve.

   If you select two endpoint vertices, you will join them. If you select points in the middle of the curve, you will join the start of the second curve to the end of the first curve.
2. Choose Modify > Join Curves.

To split two curves:

Select two adjacent vertices and choose Modify > Split Curves.

To change the registration point:

1 Click the Registration Point tool. 

The dotted lines in the window intersect at the registration point. The default registration point is the center of the cast member.

The pointer changes to a cross hair when you move it to the window.

2 Click to set the new registration point.

You can also drag the dotted lines around the window to reposition the registration point.

3 To reset the default registration point at the center of the cast member, double-click the Registration Point tool.

To change a vector shape cast member's registration point with Lingo:

Set the `regPoint` or `regPointVertex` cast member property. You can test the `centerRegPoint` property to determine whether Director automatically recenters the registration point when the cast member is edited. (If you specify a value for `regPointVertex`, any values in the `regPoint` and `centerRegPoint` properties are ignored.) See **centerRegPoint**, **regPoint**, and **regPointVertex**.

To close or open vector shapes:

Select or deselect the Close box at the left side of the window.

If the shape is closed, Director draws a line between the last and first points defined; if it is open, Director removes the line between the last and first points.

To close a shape with Lingo:

Set the `closed` cast member property to `TRUE`. See **closed**.

To scale a vector shape:

Control-Alt-drag (Windows) or Command-Option-drag (Macintosh) to proportionally resize a vector shape.

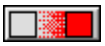
You can also enter a scaling percentage for a vector shape using the Cast Member Properties dialog box. See **Setting vector shape properties**.

Defining gradients for vector shapes

You can use controls in the Vector Shape window or Lingo to specify the type of gradient, how it is placed within a shape, and how many times it cycles within the shape. A gradient for a vector shape shifts between the fill color and the end color you define. You can create linear or radial gradients. Changes you make to vector shape gradients have no effect on gradients for bitmaps in the Paint window. You can fill only closed vector shapes with gradients.

To define a gradient for a vector shape:

1. Create a closed vector shape in the Vector Shape window.
2. Click the Gradient button.



3. To choose colors for the gradient, click the color box on the left side of the Gradient Colors control and choose a starting color from the Color menu. To choose the ending color, repeat this step using the color box on the right side of the Gradient Colors control.

4. Choose Linear or Radial from the Gradient Type pop-up menu at the top of the window.

5. To define the number of times the gradient should change colors within the shape, use the Cycles control.
6. To specify the rate at which the gradient shifts between colors, use the Scale control to enter a percentage.

   A setting of 100% uses the entire width or height of the shape to gradually shift colors. Lower settings make the shift more abrupt. For settings over 100%, the end color is reached at a theoretical location beyond the edges of the shape.
7. To rotate the gradient within the shape, use the Angle control to enter the number of degrees.

   This setting affects only linear gradients.
8. To offset the gradient within the shape, enter X Offset (horizontal) and Y Offset (vertical) values.

To specify a gradient with Lingo:

Set the fillColor, fillDirection, fillMode, fillOffset, fillScale, gradientType, and endColor cast member properties. See **fillColor**, **fillDirection**, **fillMode**, **fillOffset**, **fillScale**, **gradientType**, and **endColor**.

Controlling vector shapes with Lingo

You can use Lingo to modify a vector shape by setting properties and using commands and functions related to the shape's vertices.

- To display a list that contains the location of each vertex and control handle in a vector shape, test the `vertexList` property. See **vertexList**.
- To access a vertex directly, use the `vertex` chunk expression. See **vertex**.
- To add or delete a vertex, use the addVertex() or deleteVertex() command. See **addVertex** and **deleteVertex()**.
- To move a vertex or a vertex handle, use the moveVertex() or moveVertexHandle() command. See **moveVertex()**and **moveVertexHandle()**.
- To display the vertex list for a vector shape, test the `curve` property. See **curve**.
- To add a new shape to the vector shape, use the `newCurve()` command. See **newCurve()**.
- To display or specify the registration point for the vector shape's cast member, test or set the `regPointVertex` property. See **regPointVertex**.
- To display or specify the point around which a vector shape scales and rotates, test or set the `originMode` property. See **originMode**.

Setting vector shape properties

Use the Property Inspector to view and change settings for selected vector shape cast members. In addition to setting standard name and unload properties, you can specify anti-aliasing based on system performance and how the shape fits within the bounding rectangle.

To view or change vector shape cast member properties:

1. Select a vector shape cast member and click the Cast Member tab on the Property Inspector.
2. To specify how Director removes the cast member from memory if memory is low, be sure you're in Graphical view and choose an option from the Unload pop-up menu. See **Controlling cast member unloading**.
3. To set specific vector shape settings, click the Vector tab.
4. To change the setting for anti-aliasing, click Anti-alias.

   A check mark indicates that Anti-alias is on.

5. In the scaleMode field, you can control how the vector shape sprites are scaled on the Stage:

- Show All maintains the vector shape's aspect ratio and, if necessary, fills in any gap along the horizontal or vertical dimension using the vector shape's background color.
- No Border maintains the vector shape's aspect ratio by cropping the horizontal or vertical dimension as necessary without leaving a border.
- Exact Fit stretches the vector shape to fit the sprite exactly, disregarding the aspect ratio.
- Auto Size adjusts the vector shape's bounding rectangle to fit the movie when it is rotated, skewed, or flipped.
- No Scale places the vector shape on the Stage with no scaling. The movie stays the same size no matter how you resize the sprite, even if it means cropping the vector shape.

6. To change the size of the cast member, either enter a percentage in the Scale field (Graphical view) or use the slide bar (List view) to determine a percentage.

About importing bitmaps

Importing bitmaps is similar to importing other types of media. If you import a bitmap with a color palette or depth different from that of the current movie, the Image Options dialog box appears. You must choose to import the bitmap at its original color depth or at the current system color depth. If you are importing an 8-bit image, you have the choice of importing the image's color palette or remapping the image to a palette already in Director. See **Choosing import image options**.

Director can import images with alpha channel (transparency) effects, which are 32 bits. If you reduce the image to a lower color depth, Director removes all the alpha channel data.

When importing bitmaps, you should always consider that they will be displayed on the screen at your monitor's resolution (generally 72 to 96 dots per inch). Higher-resolution images that you place on the Stage in Director may appear much larger than you expect. Other applications, particularly those focused on creating images for print, allow you to work on the screen with high-resolution images at reduced sizes. Within Director, you can scale high- resolution images to the right size, but this may reduce the quality of the image. Also, high-resolution images use extra memory and storage space, even after they've been scaled.

If you are working with a high-resolution image, convert it to between 72 and 96 dots per inch with your image editing program before you import it into Director.

Director supports JPEG compression at run time for internal cast members imported through the Standard or Include Original Data for Editing import options. A JPEG file imported with either of these options contains both the original compressed bits and decompressed bits. Once imported, the JPEG file decompresses in the authoring environment. The cast member size displays the member's size in RAM after it has been decompressed. The amount of RAM required to display a JPEG file is larger than its size on disk, so do not be surprised that your cast member size is larger than its original size on disk in the Cast Properties window.

Director takes advantage of compressed JPEG data at run time. The original compressed data bits are saved in a Shockwave movie or a projector (if the Shockwave compression option is on). If you edit the member within Director in the Paint window, the compressed data will be lost. An alert appears before the data is overwritten.

If the Shockwave compression option is on, Director also compresses bitmaps into the JPEG format. For more information about bitmap compression, see **Compressing bitmaps**.

Using animated GIFs

You can import an animated GIF into Director with File > Import, similar to the way in which you import any other bitmap cast member. The only difference is that when the Select Format dialog box appears, you select Animated GIF.

Director supports both the GIF89a and GIF87 formats. GIFs must have a global color table to be imported. You can import an animated GIF within a movie file or link to an external file. You also have the choice of importing the first frame of an animated GIF as a still image. Just as with an ordinary bitmap, you place an animated GIF in the Score in a sprite channel and extend it through all the frames in which you want it to appear. An animated GIF can play at the same frame rate as the Director movie, at a different rate that you specify, or at its original rate.

Director does not support the following inks for animated GIFs: Background Transparent, Reverse, Not Reverse, Darkest, Lightest, Add, Add Pin, Subtract, and Subtract Pin.

You can make an animated GIF play direct-to-Stage, meaning that it is immediately displayed on the Stage instead of being first composed in an offscreen buffer with other sprites. A direct-to-Stage GIF takes less time to load, but you cannot place other sprites in front of it or use any ink effect.

To set properties for an animated GIF:

1. To specify how Director removes the cast member from memory if memory is low, choose an option from the Unload pop-up menu on the Cast Member tab of the Property Inspector (Graphical view). See **Controlling cast member unloading**.

2. To achieve the fastest playback rate, click the Animated GIF tab and select Direct to Stage.

   When Direct to Stage is on, you can use only Copy ink and you cannot place any sprites on top of the animated GIF sprite.

3. Choose an option from the Rate pop-up menu.

- Normal plays at the GIF's original rate, independent of the Director movie. The GIF cannot exceed Director's frame rate.
- Fixed plays at the frame rate you enter on the right.
- Locked-Step plays at the same rate as the Director movie.

4. To set additional animated GIF settings, click More Options.

- To change the file of a linked external cast member, enter a new pathname in the Import field or click Browse to choose a new file.
- To import a file from the Internet, click Internet and enter a new URL.

Using the Paint window

The Paint window has a complete set of paint tools and inks for creating and changing bitmap cast members for movies. Anything you draw in the Paint window becomes a cast member. When you make a change to a cast member in the Paint window, the image in the Cast window is instantly updated—as is the cast member wherever it appears on the Stage.

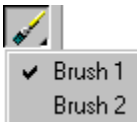
To open the Paint window, do any of the following:

- Choose Window > Paint.
- Click the Paint window icon on the toolbar.

- Press Control-5 (Windows) or Command-5 (Macintosh).
- Double-click a bitmap sprite on the Stage or in the Score or double-click the sprite's cast member in the Cast window.

Using Paint window tools and controls

If you see an arrow in the lower right corner of a tool, click it and hold down the mouse button to display a pop-up menu of options for that tool.



To select an irregular area:

- Click the Lasso tool in the Paint window and drag to enclose the pixels you want to select.

The Lasso selects only those pixels of a color different from the color the Lasso was on when you first started dragging it.

- Press Alt (Windows) or Option (Macintosh) while dragging to create a polygon selection. Every time you click, you create a new angle in the selection polygon.
- Click the Lasso tool and hold down the mouse button to choose new settings from the pop-up menu.

See **Using the Lasso tool**.

To select a rectangular area:

- Click and drag the Marquee tool in the Paint window.

- Double-click the Marquee tool to select the entire bitmap.
- Click the Marquee tool and hold down the mouse button to choose new settings from the pop-up menu.

See **Using the Marquee tool**.

To change the location of the registration point:

- Click the Registration Point tool and click the spot where you wish to set the registration point.

- Double-click the Registration Point tool to set the registration point in the center of the image.

See **Changing registration points**.

To erase:

- Click and drag the Eraser tool to erase pixels.

- Double-click the Eraser tool to erase the cast member.

To move the view of the Paint window:

- Click and drag the Hand tool to move the visible portion of the image within the Paint window.

- Shift-drag to move straight horizontally or vertically.

Press the Spacebar to temporarily activate this tool while using other paint tools.

To zoom in or out on an area:

Click the Magnifying Glass tool and click in the Paint window to zoom in. Shift-click to zoom out.

See **Zooming in and out in the Paint window**.

To select a color in a cast member:

Click the Eyedropper tool and do one of the following:

- Click a color to select it as the foreground color.
- Shift-click a color to select it as the background color.
- Alt-click (Windows) or Option-click (Macintosh) to select the destination color for a gradient.

Press D to temporarily activate the Eyedropper while using other paint tools.

To fill all adjacent pixels of the same color with the foreground color:

- Click the Bucket tool and click the area you want to fill.

- To open the Gradient Settings dialog box, double-click the Bucket tool.

To enter bitmap text:

- Click the Text tool and then click in the Paint window and begin typing.
-
- Choose character formatting with the Modify > Font command.

Bitmap text is an image. Before you click outside the text box, you can edit text you've typed by using the Backspace key (Windows) or Delete key (Macintosh). Once you have clicked outside the text box, you cannot edit or reformat bitmap text.

To draw a 1-pixel line in the current foreground:

Click the Pencil tool and drag in the Paint window. To constrain the line to horizontal or vertical, Shift-Click and drag.

If the foreground color is the same as the color underneath the pointer, the Pencil tool draws with the background color.

To spray variable dots of the foreground color:

- Click the Airbrush tool and drag in the Paint window.

- Click the Airbrush tool and hold down the mouse button to choose a new brush type from the pop-up menu. Choose Settings to change the selected brush.

See **Using the Airbrush tool**.

To brush strokes of the foreground color:

- Click the Brush tool and drag in the Paint window. To constrain the stroke to horizontal or vertical, Shift-click and drag.
-

To choose a new brush type:

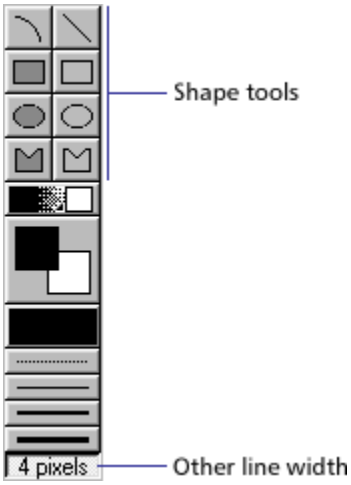
Click the Brush tool and hold down the mouse button to choose a new brush type from the pop-up menu. Choose Settings to change the selected brush.

See **Using the Brush tool**.

To paint shapes or lines:

Click and drag the shape tools. To constrain lines to horizontal or vertical, ovals to circles, and

rectangles to squares, Shift-click and drag.



The filled tools create shapes filled with the foreground color and the current pattern. The thickness of lines is determined by the line width selector.

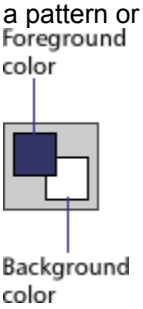
To choose a foreground and destination color for color-shifting inks:

Click the color box on the left to choose a foreground color; click the color box on the right to choose a destination color. 

These colors affect Gradient, Cycle, and Switch inks. Each of these uses a range of colors that shifts between the foreground color and the destination color.
See **Using gradients** and **Using Paint window inks**.

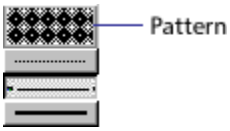
To choose the foreground and background colors

- Use the Foreground Color pop-up menu to choose the primary fill color (used when the pattern is solid and the ink is Normal).
- Use the Background Color pop-up menu to choose the secondary color (the background color in a pattern or text).



To choose a pattern for the foreground color:

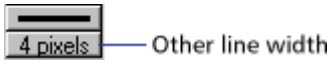
Choose an option from the Patterns pop-up menu:



- To change the pattern palette, choose Pattern Settings at the bottom of the menu.
- To define a tile—a pattern based on a rectangular section of an existing cast member—choose Tile Settings.

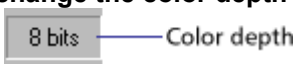
See **Editing patterns** and **Creating a custom tile**.

To choose a line thickness:

1 

- Choose the None, One-, Two-, or Three-Pixel Line button to set the line width.
- Double-click the Other Line Width button to open Paint Window Preferences and assign a width to the line.

To change the color depth of the current cast member:

1 

Double-click the Color Depth button to open the Transform Bitmap dialog box.

The button displays the color depth of the current cast member.

See **Changing size, color depth, and color palette for bitmaps**.

To choose a Paint window ink:

Choose the type of ink from the Ink pop-up menu at the bottom left of the window.

See **Using Paint window inks**.

Using the Lasso tool

You use the Lasso tool to select irregular areas or polygons. Once selected, artwork can be dragged, cut, copied, cleared, or modified with the effects on the Paint toolbar. The Lasso tool selects only those pixels of a color different from the color the Lasso tool was on when you first started dragging it. You use the Lasso pop-up menu to change settings.

To select an irregular area with the Lasso tool:

Drag with the Lasso tool to enclose the pixels you want to select.

To select a polygon area with the Lasso tool:

1 Press Alt (Windows) or Option (Macintosh) while clicking the first point.

2 Click the remaining points.

3 Double-click the last point.

To change Lasso tool settings:

1 Hold down the mouse button while the pointer is on the Lasso tool.

2 Choose one of the following options from the Lasso pop-up menu:

- Shrink causes the lasso to tighten around the selected object so that only the object is selected.
- No shrink permits you to select the entire area you drag around. The lasso selects whatever is inside the selected area.
- See Thru Lasso causes your selection to become transparent, as if the Transparent ink effect were applied.

Using the Marquee tool

The Marquee tool selects artwork in the Paint window. Once selected, artwork can be dragged, cut,

copied, cleared, or modified with the commands on the Paint toolbar. Use the Marquee pop-up menu to change settings.

To select with the Marquee tool:

Drag to select a rectangular area.

To select the entire bitmap:

Double-click the Marquee tool.

To stretch or compress art that is selected with the Marquee tool:

Hold down Control (Windows) or Command (Macintosh) while dragging a border of the selected area. Hold down Shift at the same time to maintain proportions.

To move a selection, do any of the following:

- Click and drag the selection.
- Shift-drag to constrain the movement to horizontal or vertical.
- Use the arrow keys to move the selection one pixel at a time.

To make a copy of artwork that is selected with the Marquee tool:

Hold down Alt (Windows) or Option (Macintosh) while dragging the selection.

To change marquee settings:

Click the Marquee tool, hold down the mouse button, and choose from the following options:

- Shrink causes the rectangle to shrink around the selected artwork.
- No Shrink permits you to select everything within the marquee.
- Lasso tightens the marquee around the object in the same way as the Lasso tool and selects the pixels according to the color of the pixel beneath the cross hair when you started your drag.
- See Thru modifies the selection function so that pixels with the same color as the first pixel selected are not included in the selection.

Using the Airbrush tool

The Airbrush tool sprays the currently selected color, ink, and fill pattern. To modify the spray, you choose the ink effects from the Ink pop-up menu in the Paint window. The longer you hold the Airbrush tool in one spot, the more it fills in the area.

If you hold down the mouse button when the pointer is positioned on the Airbrush tool, the Airbrush pop-up menu appears. Each of the five settings in the pop-up menu can be defined so you can have several types of spray available without opening the Airbrush Settings dialog box.

To use the Airbrush tool:

Click the Airbrush tool and drag in the Paint window.

To define airbrush settings:

1. Click the Airbrush tool and hold down the mouse button.
2. Choose the menu item for which you want to define settings.

3. Open the menu again and choose Settings from the Airbrush pop-up menu. Enter values for the options in the Airbrush Settings dialog box.

   You can also double-click the Airbrush tool to open the Airbrush Settings dialog box.

4. Use the Flow Rate slider to control how quickly the airbrush covers an area with paint. To change the flow, drag the Flow Speed slider.

5. Use the Spray Area slider to set the size of the airbrush's spray area.

6. Use the Dot Size slider to set the size of the dots sprayed by the airbrush.

7. Use Dot Options to specify how dots are sprayed from the airbrush:

- Uniform Spray causes drops sprayed by the airbrush to be uniformly sized.
- Random Sizes sprays with randomly sized drops.
- Current Brush sprays with drops shaped like the current airbrush.

Using the Brush tool

You use the Brush tool to brush strokes with the current color, ink, and fill pattern. To choose a different size and brush shape, you use the Brush Settings dialog box. The choices you make in the Brush Settings dialog box are assigned to the menu item in the pop-up menu and remain in effect until you change them. Each of the five settings in the pop-up menu can be defined so you can have several types of spray available without opening the Brush Settings dialog box.

To use the Brush tool:

Click the Brush tool and drag in the Paint window.

To change brush settings:

1. Click the Brush tool and hold down the mouse button.

2. Choose the menu item for which you want to define settings.

3. Open the menu again and choose Settings from the Brush pop-up menu. Enter values for the options in the Brush Settings dialog box.

   You can also double-click the Brush tool to open the Brush Settings dialog box.

4. To choose from the default brush shapes, choose Standard from the pop-up menu and then click a brush shape below.

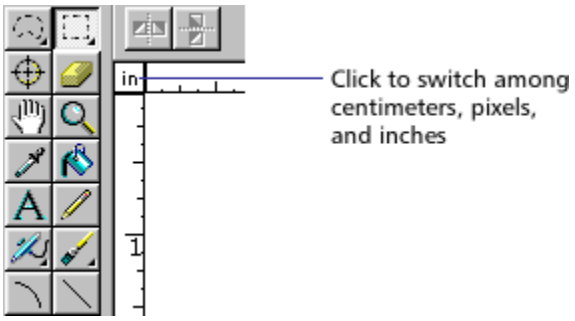
5. To create a new brush shape, choose Custom from the pop-up menu and then choose a brush shape to modify below.

6. Edit the current brush shape by clicking the magnified image of the brush shape. Clicking a blank pixel fills it, and clicking a filled pixel makes it blank. Clicking outside the Brush Shapes dialog box places the pixels on the screen at the point you click. Use the following editing controls to change the brush shape:

- The right and left arrows move the brush shape one pixel to the right or left.
- The up and down arrows move the brush shape up or down one pixel.
- The black and white square reverses the colors of the brush shape (for example, black becomes white and white becomes black).
- Copy copies the brush shape to the Clipboard.
- Paste pastes the brush into the custom set of brush shapes.

Using rulers in the Paint window

The Paint window has vertical and horizontal rulers to help you align and size your artwork.



To hide or show the rulers in the Paint window:

Choose View > Rulers.

To change the location of the zero point, do one of the following:

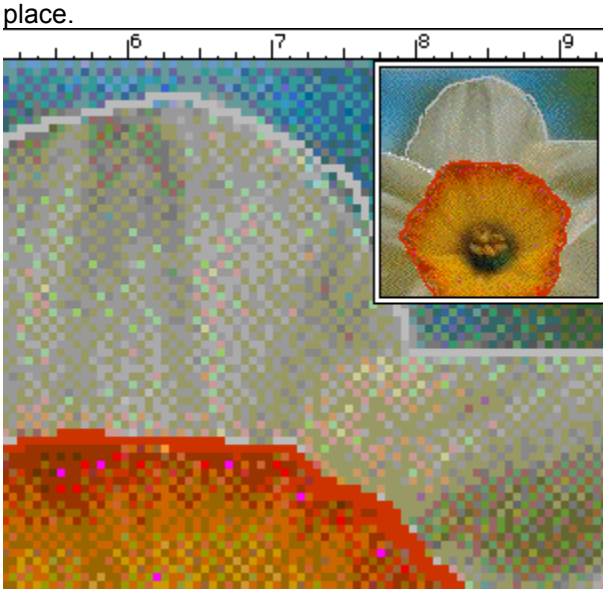
- Drag along the ruler at the top or side of the window.
- Drag into the window to align the zero point with a specific point in the cast member.

Zooming in and out in the Paint window

You can use the Magnify tool or the Zoom commands on the View menu to zoom in or out at four levels of magnification.

To zoom in or out, do one of the following:

- Click the Magnify tool and then click the image. Click again to increase the magnification. Shift-click to zoom out.
- Choose View > Zoom and then choose the level of magnification.
- Press Control + the plus (+) key (Windows) or Command + the plus (+) key (Macintosh) to zoom in, or Control + the minus (-) key (Windows) or Command + the minus (-) key (Macintosh) to zoom out.
- Control-click (Windows) or Command-click (Macintosh) the image to zoom in on a particular place.



To return to normal view, do one of the following:

- Click the normal-sized image in the upper right corner.
- Choose View > Zoom > 100%.

Changing selected areas of a bitmap

Once you have selected part of an image in the Paint window with the Lasso or Marquee tool, you can change the selected area.

To reposition the selected area:

Move the cross hair inside the selected area (the cross hair becomes an arrow pointer). Drag the selected area.

To affect how the selected area behaves when you drag it, use the following key combinations:

- To make a copy of the selected area as you drag, Alt-drag (Windows) or Option-drag (Macintosh) the selection.
- To stretch the selection (Marquee tool only), Control-drag (Windows) or Command-drag (Macintosh) the selection.
- To stretch the selection proportionally (Marquee tool only), Control-Shift- drag (Windows) or Command-Shift-drag (Macintosh) the selection.
- To copy and stretch the selection (Marquee tool only), Control-Alt-drag (Windows) or Command-Option-drag (Macintosh) the selection.
- To constrain the movement of the selection to horizontal or vertical, Shift-drag the selection.
- To move the selection one pixel at a time, use the arrow keys.

Flipping, rotating, and applying effects to bitmaps

The toolbar at the top of the Paint window contains buttons to apply effects to bitmaps. Before using any of these options, you must select part of the bitmap with the Lasso or Marquee tool. Effects that change the shape of the selection work only when the selection is made with the Marquee tool. Effects that change colors within the selection work with both the Marquee and the Lasso tools.

Lingo flips and rotates bitmaps by flipping and rotating bitmap sprites. See **Rotating and skewing sprites** and **Flipping sprites**.

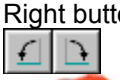
Note: To repeat any of these effects after using them, press Control+Y (Windows) or Command+Y (Macintosh).

To flip, rotate, skew, or apply effects to part of a bitmap:

1. Select part of a bitmap in the Paint window with the Marquee tool.
2. Use any of the following effects:

- To flip the selection, click the Flip Horizontal button to flip right to left, or click the Flip Vertical button to flip top to bottom.

- To rotate the selection 90º counterclockwise or 90º clockwise, click the Rotate Left or Rotate Right buttons.



- To rotate the selection by any amount in either direction, click the Free Rotate button and then drag the rotate handles in any direction. (You can rotate a sprite containing a bitmap instead of the bitmap itself. See **Rotating and skewing sprites**.

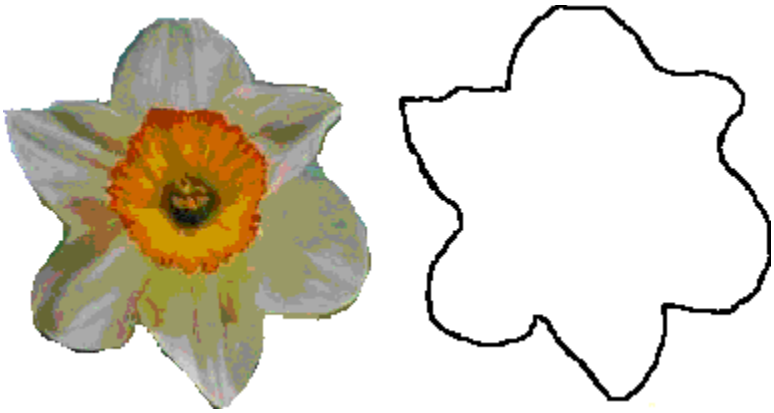

)

- To skew the selection, click the Skew button and drag any of the skew handles.

- To warp the shape of the selected area, click the Warp button and drag any handle in any direction.

- To create a perspective effect, click the Perspective button and drag one or more handles to create the effect you want.

- To create an outline around the edges of the selected artwork, click the Trace Edges button.



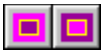
To apply color effects to a selected area:

1. Select an area within a bitmap cast member using either the Marquee or the Lasso tool.
2. Use any of the following effects:

- To soften the edges of the selected artwork, click the Smooth button. This works only with 8-bit cast members.

- To reverse the colors of the selected area, click the Invert Color button.

- To increase or reduce the brightness of the selected area, click the Lighten or Darken Color button. This works on 8-bit (256 color) images only.



- To fill the selected area with the current foreground color and pattern, click the Fill Color button.

- To change all pixels of the foreground color within the selection to the currently selected destination color, click the Switch Colors button.

Using Auto Distort

Use Auto Distort to create animations that show bitmap cast members gradually changing from frame to frame. Auto Distort generates intermediate cast members for any cast member that is free rotated, made into a perspective, slanted, distorted, or skewed.

Cast members created with Auto Distort after the perspective effect was used.

To use Auto Distort:

1. Select the portion of a bitmap cast member you want to change.
2. Use the Free Rotate, Perspective, Skew, Distort, or Stretch button to change the image.
3. Without deselecting the changed image, choose Xtras > Auto Distort.
4. Enter the number of cast members to create.

   Director generates new cast members with an intermediate amount of change applied to each one. The new cast members appear in the first available cast positions.

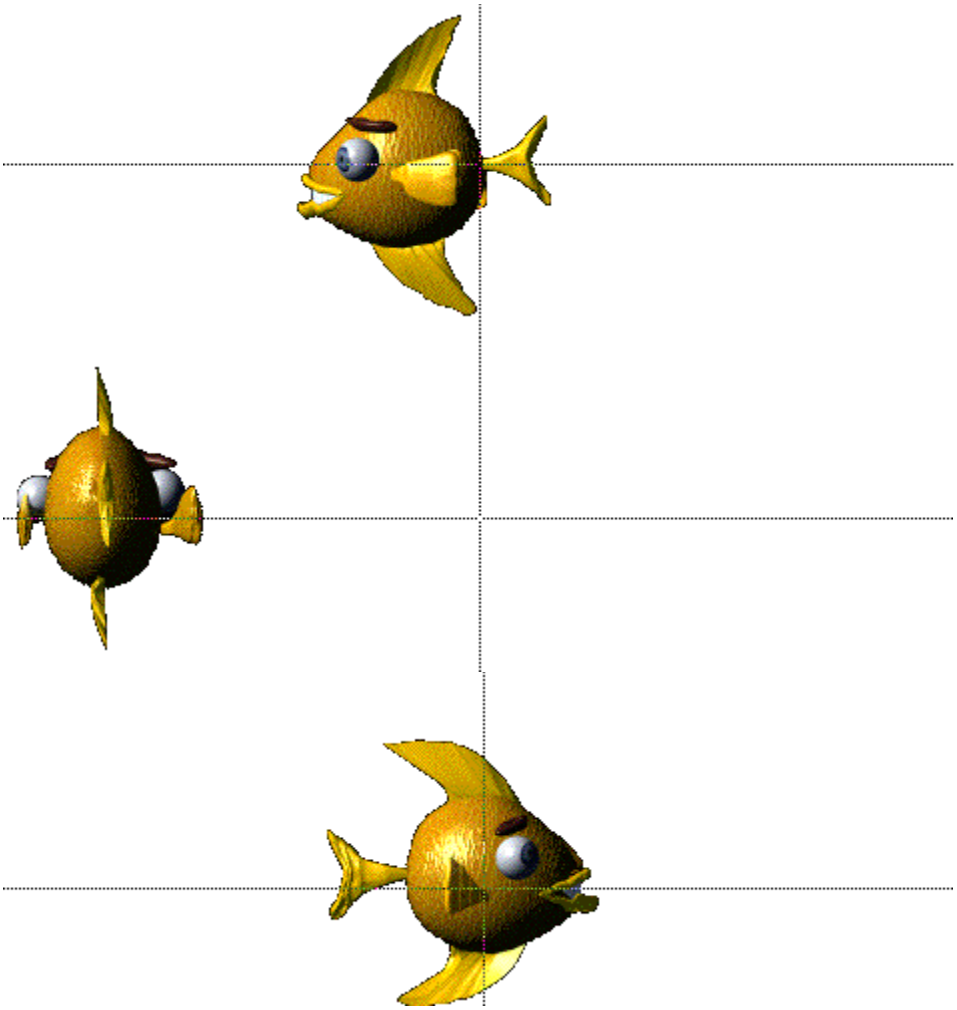
Changing registration points

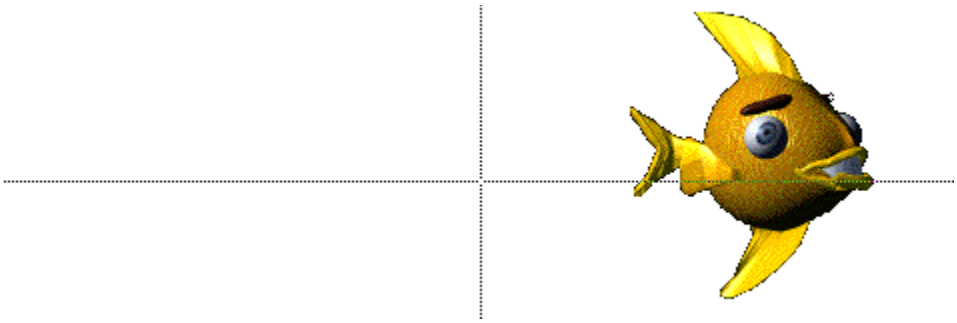
A registration point is a marker that appears on a sprite when you select it with your mouse. (Registration points do not appear on unselected sprites or when a movie is playing.) Registration points provide a fixed reference point within an image, thereby helping you align sprites and control them from Lingo. Registration points are crucial to precisely placing vector shapes, bitmaps, and all cast members that appear on the Stage.

By default, Director assigns a registration point in the center of all bitmaps, but for many types of animation, you may want to move the registration point. To do this, you can use the Registration Point tool.

You can edit a bitmap's registration point in the Paint window or using Lingo.

Moving the registration point is useful for preparing a series of images for animation. When you use Cast to Time or exchange cast members, Director places a new cast member's registration point precisely where the previous one was. By placing the registration point in the different locations, you can make a series of images move around a fixed position without having to manually place the sprites on the Stage. Use onion skinning to set registration points when images are placed in relation to each other. See **Using onion skinning**.





With the registration points set as shown, the series of fish swim in a circle without any tweening or manual placement of sprites.

To set a registration point:

1. Display the cast member you want to change in the Paint window.
2. Click the Registration Point tool.

   The dotted lines in the Paint window intersect at the registration point. The default registration point is the center of the cast member.

   The pointer changes to a cross hair when you move it to the Paint window.
3. Click a location in the Paint window to set the registration point.

   You can also drag the dotted lines around the window to reposition the registration point.

Note: To reset the default registration point at the center of the cast member, double-click the Registration Point tool.

To set a bitmap's registration point with Lingo:

Set the `regPoint` cast member property. Set the `centerRegPoint` property to specify whether Director automatically centers the registration point if the bitmap is edited. See **centerRegPoint** and **regPoint**.

Changing size, color depth, and color palette for bitmaps

You can use Transform Bitmap to change the size, color depth, and palette of selected cast members. Any change you make to a cast member's color depth or palette affects the cast member itself—not just its appearance on the Stage. You can't undo changes to the color depth and palette. If you want to keep a cast member's original bitmap unchanged but temporarily apply a different palette, use the Member tab in the cast member's Property Inspector. To change the size of only the sprite on the Stage, use the Sprite tab in the sprite's Property Inspector.

You can also remap images to new palettes with an image editing program such as Fireworks.

The Transform Bitmap dialog box displays values for the current selection. If more than one cast member is selected, a blank value indicates that cast members in the selection have different values. To maintain a cast member's original value, leave that value blank in the dialog box.

To use Transform Bitmap:

1. Select the bitmap cast members to change.
2. Choose Modify > Transform Bitmap.
3. To change the size of the bitmap, do one of the following:

   If multiple cast members are selected, you can resize all the cast members to the dimensions you enter.

- Enter new measurements (in pixels) in the Width and Height fields.
- Enter a scaling percentage in the Scale box.

  Select Maintain Proportions to keep the width and height of the selected cast member in proportion. If you change the width, the proportional height is automatically entered in the Height field. If you use Transform Bitmap to change several cast members at once, be sure to deselect Maintain Proportions. If you don't, all cast members will be resized to the values in the Width and Height boxes.

4. To change the color depth, choose an option from the Color Depth pop-up menu.

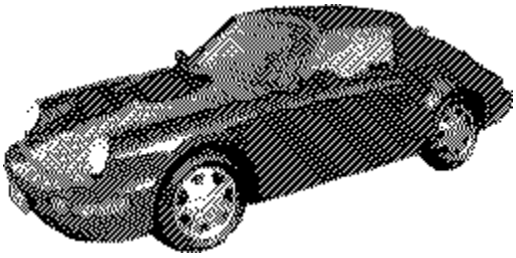
   For more information about the color depth of bitmap cast members, see **Controlling color**.

5. To change the palette, choose a palette from the Palette pop-up menu and choose one of the following remapping options:

- Remapping replaces the original colors in the graphic with the most similar solid colors in the new palette. This is the preferred option in most cases.
- Dithering blends the colors in the new palette to approximate the original colors in the graphic.

256 grays

Remapped to closest colors in black and white



Dithered in black and white

6 Click Transform to execute the changes.

The settings you choose in Transform Bitmap cannot be undone.

Controlling bitmap images with Lingo

Lingo lets you control bitmap images in two different ways. First, you can perform simple operations that affect the content of entire image cast members. These include changing the background and foreground colors, and switching the image that appears in a specific cast member with that of another cast member. Each of these operations manipulates a property of the entire image cast member.

Second, you can use Lingo to perform fine manipulations of the pixels of an image or to create entirely new images. This allows you to be extremely flexible about which images you display. You can create images based on dynamic information, such as user input, or based on any other factors you wish to include. To perform this kind of image operation, Lingo works with image objects. See **Creating image objects**.

To change the image assigned to a bitmap cast member:

- Set the picture cast member property. See **picture (cast member property)**.

To specify the background or foreground of a bitmap sprite:

- Set the backColor or foreColor sprite property. See **backColor** and **foreColor**.

To capture the current graphic contents of the Stage:

- Set a bitmap's `picture` cast member property to the Stage's `picture` property. See **picture (cast member property)**.

  For example, the statement `member("Archive").picture = (the stage).picture` makes the current image of the Stage the image for the bitmap cast member Archive.

Creating image objects

An image object can be either a self-contained set of image data or a reference to the image data of a cast member or of the Stage. If an image object is created by referring to a cast member, the object will contain a reference to the image of the member. The following statement creates an image object containing a reference to the image of the cast member called Boat.

```
myImage = member(“Boat”).image
```

Because the image object `myImage` contains a reference to the cast member Boat, any changes you make to the object will be reflected in the cast member. These changes will also be reflected in any sprites made from that cast member.

You can also create an image object containing a reference to the graphic contents of the Stage:

```
myImage = (the stage).image
```

Any changes to this image object will be reflected on the Stage.

To create an image object that is a self-contained set of image data instead of a reference to a cast member, you must tell Lingo what kind of image you want to create. You do this by providing parameters that describe the size and bit depth of the image you are creating.

This statement creates an image object that contains a 640 x 480 pixel, 16-bit image:

```
myImage = image(640, 480, 16)
```

Editing image objects

Once you have created an image object, its data can be edited with a variety of Lingo commands designed to manipulate the pixels of the image. You can crop images, draw new pixels on them, copy sections of them, and work with mask and alpha channel information.

To draw a line on an image object:

Use the `draw()` command. You must specify both the locations of each end of the line and the line's color. See **draw()**.

This statement draws a line on the previously created 640 x 480 image object `myImage`, running from 20 pixels inside the top left corner to 20 pixels inside the lower right corner, and colors it blue:

```
myImage.draw(20, 20, 620, 460, rgb(0, 0, 255))
```

To draw a rectangle on an image object:

Use the `fill()` command. You provide the same information as for the draw command, but Director draws a rectangle instead of a line. See **fill()**.

This statement draws a red 40 x 40 pixel rectangle near the upper left corner of the image object `myImage`:

```
myImage.fill(rect(20, 20, 60, 60), rgb(255, 0, 0))
```

To determine the color of an individual pixel of an image object or set that pixel's color:

Use getPixel or setPixel. See **getPixel()**and **setPixel()**.

To copy part or all of an image object into a different image object:

Use `copyPixels()`. This command requires you to specify the image you are copying from, the rectangle to copy the pixels to, and the rectangle to copy the pixels from in the source image. See **copyPixels()**.

This statement copies a 40 x 40 rectangle from the upper left area of the image object `myImage` and puts the pixels into a 40 x 40 rectangle at the lower right of the 300 x 300 pixel object called `myNewImage`:

```
myNewImage.copyPixels(myImage, rect(260, 260, 300, 300), rect(0, 0, 40, 40))
```

When using `copyPixels()`, you can specify optional parameters that tell Lingo to modify the pixels you are copying before drawing them into the destination rectangle. You can apply blends and inks, change the foreground or background colors, specify masking operations, and more. You specify these operations by adding a property list at the end of the `copyPixels()` command.

This statement performs the same operation as the previous example and tells Lingo to use the Reverse ink when rendering the pixels into the destination rectangle:

```
myNewImage.copyPixels(myImage, rect(260, 260, 300, 300), rect(0, 0, 40, 40), [#ink: #reverse])
```

To make a new image object from the alpha channel information of a 32-bit image object:

Use `extractAlpha()`. This can be useful for preserving the alpha channel information of a 32-bit image object that you plan to reduce to a lower bit depth. A reduction in bit depth would otherwise delete the alpha information. See **extractAlpha()**.

This statement creates a new image object called `alphaImage` from the alpha channel information of the 32-bit image object called `myImage`:

```
alphaImage = myImage.extractAlpha()
```

There are many more image editing operations available through Lingo. See the categorized section of the *Lingo Dictionary* for a complete list of these commands.

Using gradients

Director can create gradients in the Paint window. You can use gradients with the Brush tool, the Bucket tool, the Text tool, or any of the filled shape tools. Typically, a gradient consists of a foreground color at one side (or the center) of an image and another color, the destination color, at the other side (or outside edge) of the image. Between the foreground and destination colors, Director creates a blend of the two.

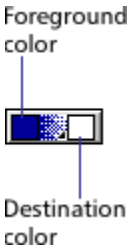
To use a gradient:

1. Choose the Brush tool, the Bucket tool, or one of the filled shape tools.
2. Choose the type of gradient from the Gradient pop-up menu.

Choosing a gradient type automatically sets the current Paint window ink to Gradient. You can also choose Gradient ink from the Ink pop-up menu at the bottom left of the Paint window to create a gradient with all the current settings.
To manually specify a gradient, choose Gradient Setting from the pop-up menu. See **Editing gradients**.

3. Choose a foreground color from Gradient Colors pop-up menu on the left.

   The foreground color is the same color specified for the Paint window.



4. Choose a destination color from the Gradient Colors pop-up menu on the right.

   The destination color is the color of the gradient when it completes the color transition.

5. Use the current tool in the Paint window.

   Director uses the gradient you've defined to fill the image.

6. To stop using a gradient, choose Normal from the Ink pop-up menu. See **Using Paint window inks**.

Editing gradients

You can change gradients before using them, by changing the settings in the Gradient Settings dialog box. In the Gradient Settings dialog box, you set the foreground and background colors as well as the pattern to use with your gradient. There are several pop-up menus that control the style of your gradient fill. Each choice you make is immediately previewed on the left.

To edit gradient settings:

1 Choose Gradient Settings from the Gradient Colors pop-up menu.

Gradient
pop-up menu

2 To determine whether the gradient is created with the pattern you select with the Patterns pop-up menu in the Paint window or with a dithered pattern, choose a Type option:

- Dither produces a smooth transition between colors. If you choose Dither, only dithering options appear in the Method pop-up menu below.
- Pattern uses the current pattern for the color transition. If you choose Pattern, only pattern options appear.

3 To determine how a gradient shifts between colors, choose an option from the Method pop-up menu:

If you choose Dither as the Type option, the following choices are available:

- Best Colors ignores the order of the colors in the palette. Instead, it uses only colors that create a continuous blend from foreground to background colors and blends them with a dithered pattern. Dithering is a technique that creates color from two or more colors of pixels interspersed together.
- Adjacent Colors uses all colors between the foreground and background colors and blends them with a dithered pattern.
- Two Colors uses only the foreground and background colors and blends them with a dithered pattern.
- One Color uses only the foreground color and fades it with a dithered pattern.
- Standard Colors ignores all colors between the foreground and background colors and adds several blended colors with a dithered pattern to create the gradient.
- Multi Colors ignores all the colors between the foreground and background colors and adds several blended colors with a randomized dithered pattern to create a smooth gradient.

If you choose Pattern as the Type option, the following options are available:

- Best Colors ignores the order of the colors in the palette and uses only colors that create a continuous blend of the foreground and background colors.
- Best Transparent ignores the order of the colors in the palette and uses only those colors that create a continuous blend of the foreground and background colors. White pixels in patterns created with this method are transparent.
- Adjacent Colors uses all the colors in the palette between the foreground and background colors for the gradient.
- Adjacent Colors Transparent uses all the colors in the palette between the foreground and background colors for the gradient. White pixels in patterns created with this method are transparent.

4 To determine the way the gradient fills an area in the Paint window, choose an option from the Direction pop-up menu:

- Top to Bottom puts the foreground color at the top and the destination color at the bottom.
- Bottom to Top puts the foreground color at the bottom and the destination color at the top.
- Left to Right puts the foreground color on the left and the destination color on the right.
- Right to Left puts the foreground color on the right and the destination color on the left.
- Directional lets you determine the direction of the gradient. You set the direction of the gradient in the Paint window with the paint tool used to fill the area.
- Shape Burst creates a gradient that starts at the edge of the area and moves toward the center. The foreground color begins at the edge and the destination color appears in the center. This option works only on the Macintosh.
- Sun Burst begins with the foreground color at the edge of the area and moves in concentric circles to the destination color at the center.

5 To control how colors cycle in a gradient, choose a Cycles option:

- Sharp cycles have a banded appearance; smooth cycles go from foreground to destination, and then back to foreground.
- One cycles the gradient once through the range of colors you define.
- Two Sharp cycles the gradient through the range of colors twice, from foreground to destination and from foreground to destination.
- Two Smooth cycles the gradient from foreground to destination, and then from destination to foreground.
- Three Sharp cycles the gradient from foreground to destination three times.
- Three Smooth cycles the gradient from foreground to destination, destination to foreground, and foreground to destination.
- Four Sharp cycles the gradient from foreground to destination four times.
- Four Smooth cycles the gradient from foreground to destination, destination to foreground, foreground to destination, and destination to foreground.

6 To choose how colors are distributed between the foreground and destination colors of the gradient, choose a Spread option:

- Equal provides even spacing of colors between the foreground and destination colors.
- More Foreground increases the amount of the foreground color in the gradient.
- More Middle increases the amount of the middle color in the gradient.
- More Destination increases the amount of the destination color in the gradient.

7 To determine whether the full range of the gradient is created over the paint object, the cast member, or the entire Paint window, choose a Range option:

- Paint Object paints the full gradient as the fill or brush stroke of the object, regardless of the object's location in the Paint window.
- Cast Member paints the full gradient within the size of the cast member.
- Window paints a full gradient only if the object is the length or width of the entire window; otherwise, it paints a partial gradient corresponding to the object's location in the window.

8 To choose a foreground, background, or destination color for the gradient, use the appropriate color picker.

The foreground color is the starting color of the gradient; the destination color is the ending color. Background color has no effect unless you are using a pattern.

9 To choose a pattern, use the Patterns pop-up menu.

Using patterns

You can choose among three sets of patterns included with Director or create your own custom patterns. The patterns you change or edit in the Paint window do not affect the patterns available for shapes.

To use a pattern:

1. Choose the Brush tool, the Bucket tool, or one of the filled shape tools.
2. Choose the type of pattern from the Patterns pop-up menu.

   To manually specify a pattern, choose Pattern Settings from the pop-up menu. See **Editing patterns**.

Editing patterns

You can change patterns before using them, by changing the settings in the Pattern Settings dialog box. Each change you make is immediately previewed.

To choose a new set of patterns or create a custom pattern:

1. Choose Pattern Settings from the bottom of the Patterns pop-up menu.
2. Choose an option from the pop-up menu at the top of the Pattern Settings dialog box:

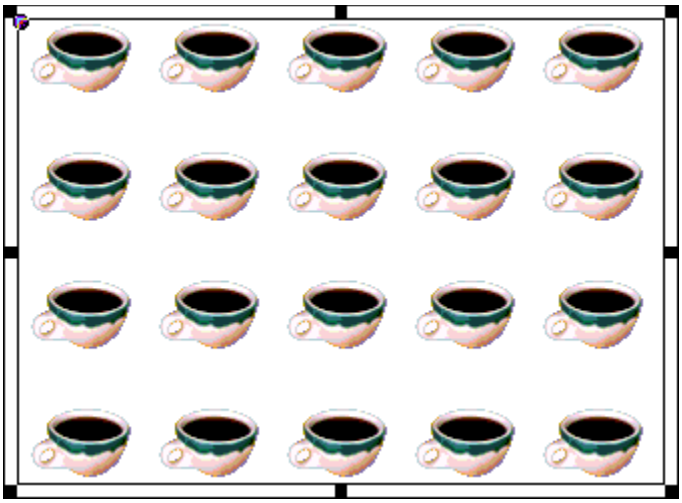
- To choose one of the standard, noneditable sets of patterns, choose QuickDraw, Grays, or Standard.
- To edit a pattern, choose Custom. Custom is an editable copy of the Standard palette set.

3. Select the pattern to edit or use the Copy and Paste buttons to move an existing pattern to one of the empty tile positions.
4. Use any of the following methods to edit the pattern:

- Click the magnified image of the pattern. Click a blank pixel to fill it and click a filled pixel to make it blank.
- Click the right, left, up, and down arrows to move the pattern one pixel in any direction.
- Click the Black and White square to reverse the colors of the pattern (for example, black becomes white, and white becomes black).

Creating a custom tile

Custom tiles provide an effective way of filling a large area with interesting content without using a lot of memory or increasing the downloading time. They are especially useful for large movies on the Web. A custom tile uses the same amount of memory no matter what size area it fills.



To create a custom tile:

1. Create a bitmap cast member to use as a tile and display it in the Paint window.
2. Click the pattern box in the Paint window and choose Tile Setting from the bottom of the Patterns pop-up menu.
3. Click an existing tile position to edit.

   The existing tiles appear next to the Edit label. You have to replace one of the built-in tiles to create a new one. To restore the built-in tile for any tile position, select it and click Built-in.
4. Click Cast Member.

   The cast member appears in the box at the lower left. The box at the right shows how the image appears when it is tiled. The dotted rectangle inside the cast member image shows the area of the tile.

   To choose a different cast member for the tile, use the arrow buttons to the right of the Cast Member button to move through the movie's cast members.
5. Drag the dotted rectangle to the area of the cast member you want tiled.
6. Use the Width and Height controls to specify the size of the tile.

The new tile appears in the tile position you selected. You can use it in the Paint window or from the Tool palette to fill shapes.

Using Paint window inks

You can use Paint window inks to create color effects for bitmap cast members. Paint window inks are different from sprite inks, which affect entire sprites and do not change cast members.

Choose an ink effect from the Ink pop-up menu at the bottom of the Paint window.

The result of the ink you choose depends on whether you are working in color or black and white. Also, some inks work better when painting with patterns, and others work better when painting with solid colors.

Ink	B&W	Color	Works with
Normal	X	X	Solids and patterns
Transparent	X	X	Patterns
Reverse	X		Solids and patterns
Ghost	X	X	Solids (B&W) and patterns (color)
Gradient	X	X	Brush, Bucket, shape tools
Reveal	X	X	Brush, shape tools
Cycle		X	Solids and patterns
Switch		X	Brush
Blend		X	Solids and patterns
Darkest		X	Patterns
Lightest		X	Patterns
Darken		X	Brush
Lighten		X	Brush
Smooth		X	Brush
Smear		X	Brush
Smudge		X	Brush
Spread	X	X	Brush
Clipboard	X	X	Brush

Normal is the default ink. It is opaque and maintains the color of the current foreground color and pattern.

Transparent ink makes the background color of patterns transparent so artwork drawn previously in the current cast member can be seen through the pattern.

Reverse ink makes overlapping colors reverse. Any pixel in the foreground art that was originally white becomes transparent. Any pixel that was black reverses the color of the background art.

Ghost ink in black and white creates an image than can be seen only when drawn over a black background. In color, Ghost draws with the current background color.

Gradient lets you paint with the gradient fill selected in the Gradient Settings dialog box. See **Using**

gradients. A gradient fill is one that progresses from one color, called the foreground color, to another color, called the destination color. You can paint with Gradient ink using the Brush tool, the Bucket tool, and the shape tools.

Reveal works indirectly with the art in the previous cast position. Imagine the previous cast member's artwork covered with a white area. Reveal erases the white area to show the artwork in the previous window. Reveal can be used to create specific shapes from shades created with the Airbrush tool. Since it is impossible to mask certain shapes for the airbrush, spray an area with the airbrush first; then in the next cast member, paint the shapes you need with a Reveal ink. As you paint your object, you will expose the airbrush pattern in the previous window.

Cycle is a color ink. As you draw with Cycle ink, the colors change as the ink progresses through the palette. The beginning and ending points of the color cycle are determined by the foreground and destination colors. If you want to cycle through the whole palette, choose white as the foreground color and black as the destination color. This ink works only when your computer is set to 256 colors.

Switch changes any pixel that is the current foreground color to the current gradient destination color as you paint over that color.

Blend creates a translucent color ink. You can see the background object, but its color is blended with the foreground object's color. Choose the percentage of blend in the Paint Window Preferences dialog box.

Darkest is a useful ink for coloring black-and-white artwork. For example, if you paint yellow over black and white, black will remain black since it is darker than yellow, and white will become yellow because yellow is darker than white.

Lightest is a useful ink for coloring black-and-white artwork. For example, if you paint yellow over black and white, black objects become yellow when painted with the Lightest ink effect, and white remains white because it is lighter than yellow.

Darken makes colors darker. The more times you click with the Brush tool, the darker the area becomes. The colors of the foreground, background, and destination inks have no effect on Darken. Darken creates an effect that is the same as reducing a color's brightness with the controls in the Color Palettes window. You can change the rate of this ink effect in the Paint Preferences dialog box.

Lighten makes colors lighter. The more times you click with the Brush tool, the lighter the area becomes. The color of the foreground, background, and destination inks have no effect on Lighten. Lighten creates an effect that is the same as increasing a color's brightness with the controls in the Color Palettes window. You can change the lightness of this ink effect in the Paint Preferences dialog box.

Smooth blurs existing artwork when it is painted with the Brush tool. It is not directional like Smear and Smudge. The color of the foreground, background, and destination inks have no effect on Smooth. Use it to smooth out jagged edges.

Smear works with the Brush tool and functions like mixing paint. Any area you drag across with a Smear ink spreads in the direction of the brush, fading as it gets farther from the source. The color of the foreground, background, and destination inks have no effect on Smear.

Smudge is a color ink for the Brush tool that is similar to Smear. It also functions like mixing paint. The colors fade faster as they are spread. The color of the foreground, background, and destination inks have no effect on Smudge.

Spread works with the Brush tool in color. Whatever is under the Brush tool when you start to drag is picked up as the ink for the brush. Copies of what is beneath the brush are pushed across the window as you draw.

Clipboard uses the current contents of the Clipboard as a pattern to paint with. The clipboard contents must originate in Director.

Using bitmap filters

Bitmap filters are plug-in image editors that apply effects to bitmap images. You can install Photoshop-compatible filters to change images within Director.

Original image

Filtered image

To install a filter, place it in the Xtras folder in the Director application folder. See **Installing Xtras**.

You can apply a filter to a selected portion of a bitmap image, to an entire cast member, or to several cast members at once.

To apply a filter:

1. Open the cast member in the Paint window or select the cast member in the Cast window.

   You can apply a filter to several cast members at once by selecting them all in the Cast window. To apply a filter to a selected portion of a cast member, use the Marquee or Lasso tool in the Paint window to select the part you want to change.

2. Choose Xtras > Filter Bitmap.

3. In the Filter Bitmap dialog box, choose a category on the left and a filter on the right.

   To view all the filters at once, choose All from the Categories list.

4. Click Filter.

   Many filters require you to enter special settings. When you choose one of these filters, a dialog box or other type of control appears after you click Filter. When you finish choosing filter settings and proceed, the filter changes the cast member.

   Some filters have no changeable settings. When you choose one of these filters, the cast member changes with no further steps.

Using filters to create animated effects

You can use Auto Filter to create dramatic animated effects with bitmap filters. Auto Filter applies a filter incrementally to a series of cast members. You can use it either to change a range of selected cast members or to generate a series of new filtered cast members based on a single image. When you define a beginning and ending setting for the filter, Auto Filter applies an intermediate filter value to each cast member.

You can tween a bitmap filter with Auto Filter.

Note: Most filters do not support auto filtering. The Auto Filter dialog box lists only those filters that do.

To use Auto Filter:

1. Select a bitmap cast member or a range of cast members and then choose Xtras > Auto Filter.

   If you want to change only a portion of a bitmap cast member, use the Marquee or Lasso tool in the Paint window to select the part you want to change.

2. In the Auto Filter dialog box, select a filter.

3. Click Set Starting Values and use the filter controls to enter filter settings for the first cast member in the sequence.

   When you finish working with the filter controls, the Auto Filter dialog box reappears.

4. Click Set Ending Values and use the filter controls to enter filter settings for the last cast member in the sequence.

5. Enter the number of new cast members you want to create. The box is not available if you have selected a range of cast members.

6. Click Filter to begin the filtering.

   A message appears to show the progress. Some filters are very complex and require extra time for computing.

   Auto Filter generates new cast members and places them in empty cast positions following the cast member you selected. If you selected a range of cast members, no new cast members appear, but the cast members in the range you selected are changed incrementally.

Using onion skinning

Onion skinning derives its name from a technique used by conventional animators, who drew on very thin "onion skin" paper so that they could see through it to one or more of the previous images in the animation.

With onion skinning in Director, you can create or edit animated sequences of cast members in the Paint window using other cast members as a reference. Reference images appear dimmed in the background. While working in the Paint window, you can view not only the current cast member that you're painting but also one or more cast members blended into the image.

You can use onion skinning to do the following:

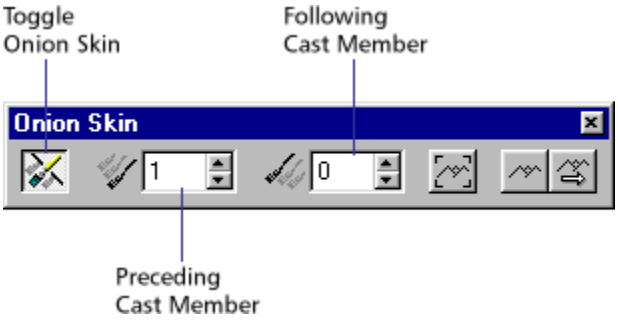
- To trace over an image or create a series of images all in register (aligned) with a particular image.
- To see previous images in the sequence and use those images as a reference while you are drawing new ones.
- To create a series of images based on another parallel animation. The series of images serves as the background while you paint a series of foreground images.

Onion skinning uses registration points to align the current cast member with the previous ones you have chosen. Be careful not to move registration points for cast members after onion skinning. If you do, the cast members may not line up the way you want them to. See **Changing registration points**.

You must have created some cast members in order to use onion skinning.

To activate onion skinning:

1. Open the Paint window and choose View > Onion Skin. The Onion Skin toolbar appears.

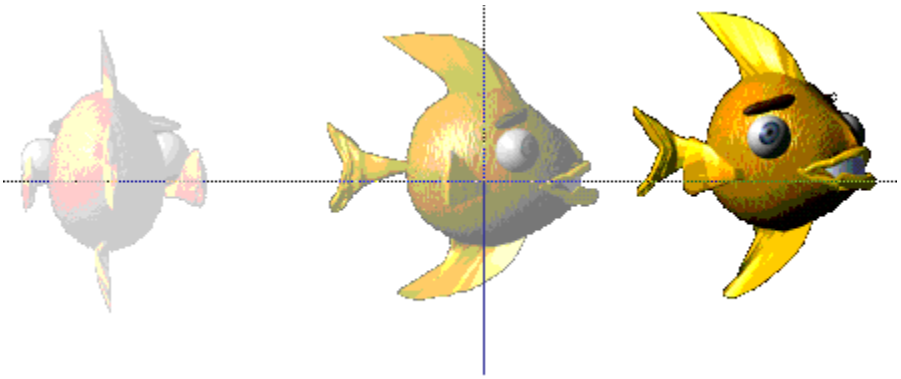


2. Click the Toggle Onion Skinning button at the far left of the toolbar to enable onion skinning.

To define the number of preceding or following cast members to display:

1. Open the Paint window and choose View > Onion Skin. The Onion Skin toolbar appears.
2. If necessary, click the Toggle Onion Skinning button on the Onion Skin toolbar to activate onion skinning.
3. Specify the number of preceding or following cast members you want to display.

- To specify the number of preceding cast members to display, enter a number in the Preceding Cast Members box.
- To specify the number of following cast members to display, enter a number in the Following Cast Members box.



Two preceding cast members shown with onion skinning and registration points

The specified number of cast members appear as dimmed images behind the current cast member. The order is determined by the position in the cast.

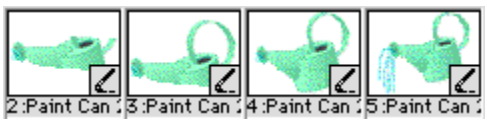
To create a new cast member by tracing over a single cast member as a background image:

1. Open the Paint window and choose View > Onion Skin. The Onion Skin toolbar appears.
2. In the Paint window, open the cast member that you want to use as the reference image or background.

3. If necessary, click the Toggle Onion Skinning button on the Onion Skin toolbar to activate onion skinning.
4. To set the background image, click the Set Background button on the Onion Skin toolbar. 
5. To create a new cast member, click the New Cast Member button in the Paint window. 
6. Click the Show Background button on the Onion Skin toolbar. 
   The original cast member appears as a dimmed image in the Paint window. You can paint on top of the original cast member's image.
7. Paint the new cast member using the background image as a reference.

To use a series of images as a background while painting a series of foreground images:

1. In the Cast window, arrange in consecutive order the series of cast members you want to use as your background.



Cast members in both the foreground and the background series must be adjacent to each other in the

cast.

2 Open the Paint window and choose View > Onion Skin. The Onion Skin toolbar appears.

The Onion Skin toolbar appears.

3 If necessary, click the Toggle Onion Skinning button on the Onion Skin toolbar to activate onion skinning.

Make sure all values in the Onion Skin toolbar are set to 0.

4 Open the cast member you want to use as the first background cast member in the reference series. Click the Set Background button. 

5 Select the position in the cast where you want the first cast member in the foreground series to appear. Click the New Cast Member button in the Paint window to create a new cast member. 

The first cast member in the foreground series can be located anywhere in any cast.

6 Click the Show Background button to reveal a dimmed version of the background image. 

7 Click the Track Background button on the Onion Skin toolbar. 

8 Paint the new cast member using the background image as a reference.

9 When you have finished drawing the cast member, click the New Cast Member button again to create the next cast member.

When Track Background is enabled, Director advances to the next background cast member in the series. Its image appears in the background in the Paint window.

10 Repeat step 8 until you have completed drawing all the cast members in the series.

About using Paste as Pict

You use the Paste as Pict option to paste a PICT image into the cast and have it remain in PICT format.

If you paste a PICT image into the cast using the Paste command in the Edit menu, Director converts it to a bitmap. If you want the artwork to remain in PICT format, use Paste as PICT when you paste it into the cast.

You may want to use Paste as PICT for several reasons. PICT cast members may occupy less memory. Some PICT cast members, such as compound images consisting of lines, shapes, and text, will stretch and scale more smoothly than bitmap cast members. PICT cast members also look better when printed on a laser printer.

However, PICT cast members animate more slowly than bitmap cast members, and they don't support ink effects. When you use color cycling or palette transitions, PICT cast members may yield unexpected results.

Setting bitmap cast member properties

To view important information about cast members, change a cast member's name, choose alpha settings, or turn on highlighting and dithering, you use bitmap cast member properties.

To view or change bitmap cast member properties:

1. Select a bitmap cast member, and click the Member tab of the Property Inspector using the Graphical view.

   The Member tab displays the following:

- Updateable fields to view or change the cast member's name, a Comments field to enter text that appears in the Comments column of the Cast List window, and an Unload field that lets you determine how to remove a cast member from memory.
- View-only fields that indicate when the cast member was created and modified, and the name of the person who modified the cast member.

2. If the bitmap cast member is linked to an external file, the full pathname for the file appears in the Filename field. To choose a different file to link to the cast member, either type a new file name into the field, or click the Browse button and select the path to the new file name.

3. Click the Bitmap tab using the Graphical view.

4. To invert the current cast member when the user clicks it, select Highlight When Clicked.

   Use this option to create buttons. Even if Highlight When Clicked is selected, the cast member will not do anything unless it is controlled by a behavior or Lingo script.

5. To make Director approximate an original color in the bitmap if there is a palette problem, select Dither. When a color is not available because of a palette conflict, Dither displays a pattern of pixels of similar colors. If this option is off, Director uses the color in the current palette closest to the original.

6. If the imported bitmap has a white canvas that you want to remove, select Trim White Space. If you want to retain the white canvas, deselect Trim White Space.

7. To make Director use the alpha channel (transparency) data in the cast member, choose Use Embedded Alpha.

   This option is on by default for all imported cast members with alpha channel data.

8. To determine how a transparent area receives a mouse click, use the Alpha Click Threshold slider to specify a value.

   Any area with a greater degree of opacity than the specified threshold will be able to receive a mouse click.

9. To assign a different palette to an 8-bit cast member while maintaining the cast member's original palette references, choose a new palette from the Palette pop-up menu.

Setting PICT cast member properties

You use PICT cast member properties to change the names of PICT cast members and set their properties.

To view or change PICT cast member properties:

1. Select a PICT cast member and open the Property Inspector in Graphical view.
2. To view or edit the cast member name, use the Name field on the Member tab.
3. To specify how Director removes the cast member from memory if memory is low, choose an option from the Unload pop-up menu. See **Controlling cast member unloading**.

Setting Paint window preferences

You can use Paint window preferences to modify the settings of a number of tools and drawing methods in the Paint window.

To change Paint window preferences:

1. Choose File > Preferences > Paint.
2. To make tools remember the last color or ink used, choose the following options:

- Remember Color remembers the last color used with a tool, which remains, selected for the next time you use the Brush or Airbrush tools.
- Remember Ink remembers the last ink used with a tool, which remains selected for the next time you use any tool.

3. To control the way colors cycle when you draw with Cycle ink, choose one of the following options:

- Repeat Sequence causes colors to cycle from foreground to destination color and then repeat from foreground to destination.
- Reverse Sequence causes colors to cycle from foreground to destination color and then from destination to foreground.

4. To set a line width thicker than the widths available in the Paint window, use the Other Line Width slider to enter a value.

   The width you set will be the width that appears when you draw a line after selecting Other Line Width.

5. To set the opacity of a color when using the Blend ink effect in the Paint window, use the Blend slider to enter a value.

   You can vary the blend value between 0 and 100 percent.

6. To set the rate at which artwork changes when you use the Darken or Lighten effects in the Paint window, use the Lighten or Darken slider to enter a value.
7. To determine how colors are used when using Smooth, Lighten, Darken, or Cycle effects, choose an Interpolate By option:

- Color Value ignores the order of the colors in the palette and produces a continuous blend of the foreground and destination colors.
- Palette Location uses all the colors in the palette between the foreground and destination colors.

Setting button cast member properties

Use button cast member properties to change the name and button type of button cast members.

To view or change button cast member properties:

1. Select a button cast member and click the Member tab of the Property Inspector using the Graphical view.
2. To view or edit the cast member name, use the Name field.
3. To specify how Director removes the cast member from memory if memory is low, choose an option from the Unload pop-up menu. See **Controlling cast member unloading**.
4. To change the type of button, click the Button tab and select Push Button, Check Box, or Radio Button from the Type pop-up menu.

Using shapes

Shape cast members are the same non–anti-aliased shapes that were available in older versions of Director. Similar to vector shapes, they are very memory efficient.

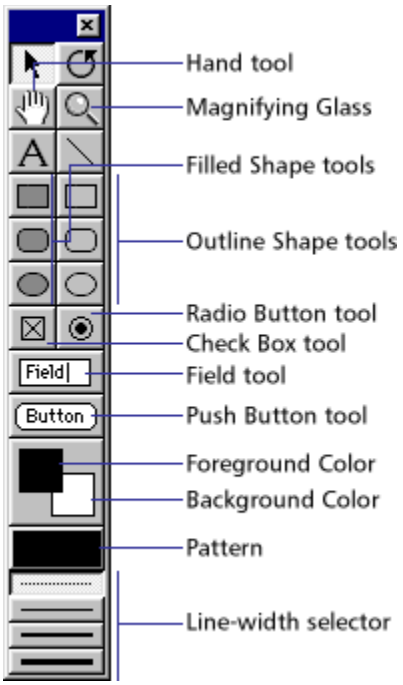
Shapes are images you can create directly on the Stage with the Line, Rectangle, Rounded Rectangle, and Ellipse tools on the Tool palette. You can fill shapes with a color, pattern, or custom tile. Shapes require even less memory than vector shapes, but Director does not anti-alias shapes, so they don't appear as smooth on the Stage as vector shapes. Use shapes for creating simple graphics and backgrounds when you want to keep your movie as small as possible. Shapes are especially useful for filling an area with a custom tile to create an interesting background that downloads quickly from the Internet. See **Creating a custom tile**.

The Radio Button, Check Box, and Button tools in the Tool palette work create simple buttons. These buttons do not do anything unless you attach a behavior or Lingo script to them.

To create a shape:

1. Select a frame in the Score where you want to draw a shape.
2. Choose color, line thickness, and pattern settings with the controls in the Tool palette. (To open the Tool palette, choose Window > Tool Palette.)
3. Click a tool and then drag on the Stage to draw the shape.

   The new shape appears on the Stage and in the Cast window.



Use the Field button to create field cast members directly on the Stage. Use the Push Button tool to create push button cast members directly on the Stage.

To specify a shape's type with Lingo:

Set the `shapeType` cast member property. See **shapeType**.

Setting shape cast member properties

Use cast member properties to view and change settings for selected shape cast members. You can change the type of shape and choose a new fill color or pattern.

To view or change shape cast member properties:

1. Select a shape cast member and open the Property Inspector in Graphical view.
2. Use the Name field on the Member tab to view or edit the cast member name.
3. To specify how Director removes the cast member from memory if memory is low, choose an option from the Unload pop-up menu. See **Controlling cast member unloading**.
4. To change the type of shape, click the Shape tab and choose an option from the Shape pop-up menu.
5. To fill the shape with the current color and pattern, select Filled.

To specify a shape's fill with Lingo:

Set the `filled` and `pattern` shape cast member properties. See **filled** and **pattern**.

To specify the line size for a shape with Lingo:

Set the `lineSize` cast member or sprite property. See **lineSize**.

Compressing bitmaps

If you plan to distribute your movie over the Internet, you can compress your bitmap images to ensure faster downloading. Director lets you compress images at the movie level and for individual cast members. Bitmap compression set at the cast member level overrides compression settings at the movie level.

In addition to Director standard compression, you can use JPEG compression and specify a range of image quality. If you have Fireworks installed, you can use the Optimize in Fireworks button to launch Fireworks, then dynamically apply compression settings while viewing how your image will look at those settings. When you determine the most suitable compression level, Director will remember the settings you established in Fireworks. For more information, see **Optimizing Bitmaps in Fireworks** in the Director Support Center Web site.

To compress a bitmap at the cast member level:

1. Select bitmap cast members or sprites and click the Bitmap tab of the Property Inspector.

   If you've selected multiple cast members or sprites, the Property Inspector displays the compression setting if it is the same for each selected object.

2. Click the Compression pop-up window.

- To compress selected bitmaps using the same settings as those established for movie-level compression, select Movie Setting. For more information on setting bitmaps at the movie level, see the information on the Compression tab under **Changing Publish settings**.
- To use the standard Director compression, select Standard.
- To use JPEG compression, select JPEG and move the slider bar to the desired level of compression. Note that the higher the number you specify, the less your bitmap will be compressed (e.g., 100 indicates no compression).

Movie Setting is usually the default compression setting, except under certain conditions when the compression feature is disabled, or when Director controls image compression choices.

For example, when the image is a JPEG, the compression setting defaults to JPEG compression. You cannot select another compression option.

Similarly, the Compression setting defaults to Standard compression, and you cannot change this, when the cast member is any of the following:

- An 8-bit cast member created in the Paint window
- A GIF imported as a bitmap with no alpha channel information
- An 8-bit PNG
- A linked cast member or a cast member created with Lingo

If you open a Director 7 movie in Director 8, bitmap cast members are assigned Movie Setting as the default, and compression settings at the movie level, set in the Publish Settings dialog box, default to the Standard compression setting. This ensures the movie will continue to play as it did in Director 7.

To compress bitmaps at the movie level:

1. Choose File > Publish Settings.

   The Publish Settings dialog box appears.

2. On the Compression tab, make a selection from the Image Compression pop-up menu and click OK.

- To use the standard Director compression, select Standard.

- To use JPEG compression, select JPEG and move the slider bar to the desired level of compression. Note that the higher the number you specify, the less your bitmap will be compressed (e.g., 100 indicates no compression).

Note: Director saves your publish settings when you save your movie.

Text: Overview

Director creates text that is editable, anti-aliased, and compact—for fast downloading in any font on any platform. Combine these features with any of Director's animation capabilities, such as rotation, and you can create text effects not possible in any other application.

You can embed fonts in a movie to ensure that text appears in a specific font when a movie is delivered, regardless of which fonts are available on the user's computer.

Because Director renders text in the display font and anti-aliases it as the movie plays, text in Director is very compact and downloads quickly from the Internet. Most of the high-quality text you see in Web browsers is actually a GIF or JPEG graphic, and takes longer to download than Director text.

Director provides many ways to add text to a movie. You can either create new text cast members within Director or import text from an outside source such as a document stored on the Internet. You can import plain text, RTF, or HTML documents. Once text is part of your movie, you can format the text in a variety of ways using Director's formatting tools. Director offers standard professional formatting functions, including alignment, tabs, kerning, spacing, subscripts, superscripts, color, and so on. You can also create hyperlinks for any text.

Text in Director is editable when you are working on your movie and, optionally, while a movie plays.

You can also use Lingo to control text. For example, you can use Lingo to edit the text in existing cast members, specify text formatting such as font and size, and interpret strings that users enter.

To create the smallest possible text cast members, use field text. Field text is standard text controlled by your system software, just like the text you see in dialog boxes and menu bars. Director does not anti-alias field text or support paragraph formatting and tabs for fields. As with regular text, Lingo can control field text and specify whether field text is editable while a movie plays.

Whereas regular text is best suited for large type that you want to look as good as possible, field text is an excellent choice for large blocks of smaller text in standard fonts (such as Times or Helvetica) that do not need to be anti-aliased.

Embedding fonts in movies

Before creating text or field cast members, it's good practice to embed the fonts you want to use in the movie. Embedding fonts makes Director store all font information in the movie file so that a font will display properly even if it is not installed in a user's system. Because embedded fonts are available only to the movie itself, there are no legal obstacles to distributing fonts in Director movies.

Embedded fonts appear in a movie as cast members and work on Windows and Macintosh computers. Director compresses embedded fonts so they usually only add 14 to 25K to a file.

For the best display at smaller sizes, include bitmap versions of a font when you embed a font. For small font sizes, usually from about 7 to 12 points, bitmap fonts often look better than anti-aliased outline fonts. (See **About anti-aliased text**.) Adding a set of bitmap characters does, however, make the font cast member larger. Examine the text display quality of your movie to find out if this option is worthwhile.

To speed up movie downloading, you can keep file size small by specifying a subset of characters to be included. You can also specify which point sizes to include as bitmaps and which characters to include in the font package. If you do not embed fonts in a movie, Director substitutes available system fonts.

If you create embedded fonts using the original font name followed by an asterisk (for example, Arial* for the Arial font), Director uses the embedded font for all the text in the movie that uses the original font. This saves you the trouble of manually reapplying the font to all the text in existing movies.

Once you embed a font in a movie file, the font appears on all of the movie's font menus, and you can use it as you would any other font.

To embed a font in a movie:

1. Choose Insert > Media Element > Font.
2. From the Original Font pop-up menu, choose a font that is currently installed on your system.

   You cannot embed a font that is not installed on your system.

   In the New Font Name box, the name of the font is followed by an asterisk. This is the name that will appear on all font menus in Director. In most cases, you should not change the name of a font.
3. To include bitmap versions of the font in specified sizes, click the Sizes button for Bitmaps and enter the point sizes you want to include, separated by spaces or commas. For example, you might enter **9, 10, 14**.
4. To include bitmap versions of bold or italic characters with the font, select Bold or Italic.

   This option provides better-looking bold and italic fonts if you are including a bitmap version of the font, but it increases the file size.
5. To specify the characters included in the font, choose an option for Characters.

- Entire Set includes every character (symbols, punctuation, numbers, and so on) with the font.
- Partial Set lets you choose exactly which characters are included. To choose a group of characters, select Punctuation, Numbers, Roman Characters, or Other. If you choose Other, enter the characters to be included in the box on the right. In some double-byte languages, other groups of characters may appear.

To embed a font in a movie with Lingo:

Use the `recordFont` command. See `recordFont` in the *Lingo Dictionary*.

Creating text cast members

You can create text within Director or import text from external files.

Creating text in Director

Director provides two ways to create text cast members: directly on the Stage or in the Text window.

To create text cast members directly on the Stage:

1. Click the Text tool in the Tool palette. 
2. Drag the pointer on the Stage to create a text cast member.

   You cannot adjust the height of the text object at this point—the height adjusts automatically when you add text.

   When you release the mouse button, a text insertion point appears in the area you just defined.
3. Enter text.

   The new text cast member appears in the first available position in the current cast; the sprite is placed in the first open cell in the current frame in the Score.

To create text cast members in the Text window:

1. Choose Insert > Media Element > Text.

   If the Text window is already open, click the New Cast Member button to create a new text cast member. 
2. Enter text in the Text window.

   Text you enter appears in the first available cast position, but it is not automatically placed on the Stage.
3. To change the width of the cast member, drag the bar at the right edge of the cast member.

Importing text

You can import text from any application that saves text in rich text format (RTF), in plain text (ASCII), or from HTML documents. Use the standard importing procedure with File > Import to import any RTF, ACII, or HTML document. To import an HTML document from the Internet, use the Internet button in the Import dialog box and enter a URL.

Note: Text and RTF files are always imported and stored inside the movie file even if you select Link to External File.

When importing text from an HTML document, Director recognizes most standard tags and parameters, including tables, and approximates the formatting. Director does not recognize embedded objects other than tables, and it does not support nested tables. It also does not recognize `APPLET`, `FORM`, `FRAME`, `INPUT`, or `IMAGE` tags.

Director ignores any tags it does not recognize. Be sure to test the importing of any HTML file that is updated frequently to make sure you're satisfied with the formatting.

When importing text from an RTF file, Director recognizes most standard RTF formatting, but it does not import pictures embedded in the file.

The amount of text in a cast member is limited only by the memory available in the playback system.

Importing text with Lingo

Lingo can import text in several ways.

- To import text from a URL, use the `getNetText()` function. See **getNetText()**.
- To import text from an external file from a URL or the local computer, select or create a text cast member and set its `fileName` property to the name of the external file that contains the text. See **fileName (cast member property)**.
- To import text from a file on disk, use the `getPref()` function. If no `setPref` command has already written such a file, the `getPref()` function returns `VOID`. See **getPref()**.

Editing and formatting text

Director offers a number of ways to edit and format text. You can edit text directly on the Stage and format it with the floating Text Inspector, or use the Text window to work in a more traditional text editing environment. Many of the same formatting controls are in the Font and Paragraph dialog boxes as well as in the Text window and the Text Inspector. Choose the most convenient option for your work style.

Selecting and editing text on the Stage

For basic text editing, it's fastest to edit text directly on the Stage.

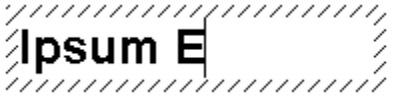
To edit text on the Stage:

1. Click a text cast member on the Stage to select it as a sprite.

   The text sprite appears as a normal sprite with double borders.

2. Click twice to edit the text.

   An insertion point appears in the text, and you can begin editing.



3. Use the Text Inspector to reformat the text.

   You can also use the Modify > Font and Modify > Paragraph commands to reformat selected text.

   When you make a change, Director updates all sprites that display the text cast member.

Note: If you're changing the background color of text, you have two options. To change the background color of the cast member, double-click the text sprite on the Stage and assign a value from the Color box on the Tool palette. You can also tint the sprite's background, which blends the background color of the cast member with the background color of the sprite. To apply this effect, select the sprite and choose a background color on the Sprite tab of the Property Inspector.

To edit text on the Stage during playback:

1. Select a text sprite and click Editable in the Sprite tab of the Property Inspector. See **Displaying and editing sprite properties in the Property Inspector**.
2. Begin playing back the movie.
3. On the Stage, double-click to edit the text.

Formatting characters

Once you have created text cast members for your movie, you can format them in a variety of ways: you can set the font, style, size, line spacing, and color. The following procedure uses the Font dialog box, but many of the same options are available in the Text Inspector and the Text window.

To format characters:

1. Double-click inside a text sprite.
2. Drag to select the text you want to format.
3. Choose Modify > Font to open the Font dialog box.
4. Choose from the following options in the Font dialog box:

- To specify the font, choose a font from the pop-up menu. Be sure to use embedded fonts for movies you intend to distribute (see **Embedding fonts in movies**).
- To use bold, italic, underline, superscript, subscript, or strikeout for text, click the appropriate box.
- To change the point size of text, increase or decrease the size with the Size option.
- To change the distance between lines of text, increase or decrease the spacing with the Spacing option.
- To specify kerning between selected characters, use the Kerning option to specify the number of points. This setting supplements the standard kerning applied to the entire cast member in the Text Cast Member Properties dialog box. See **About kerning**.
- To change the text color, click the color box and choose a color from the Color menu.

Formatting paragraphs

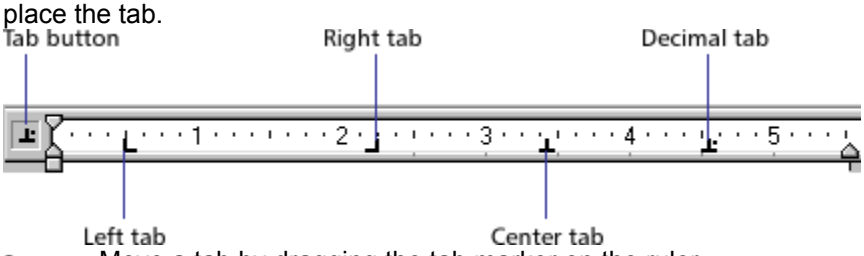
You can specify the alignment, indentation, tabs, and spacing for each paragraph in a text cast member. The following procedure explains how to format paragraphs while you work in the Text window, but many of the same formatting options are available in the Text Inspector and the Paragraph dialog box.

To make formatting changes to a paragraph:

1. Double-click the text cast member in the Score to open the Text window.
2. If the ruler is not visible, choose View > Ruler.

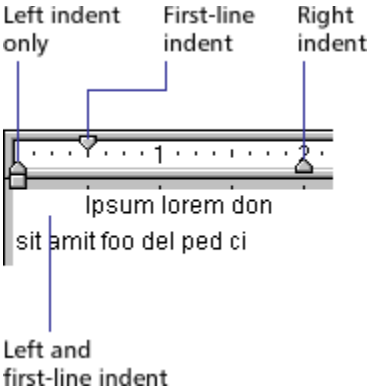
   To change the unit of measure on the text ruler, choose File > Preferences > General and select Inches, Centimeters, or Pixels from the Text Units pop- up menu.
3. Place the insertion point in the paragraph you want to change, or select multiple paragraphs.
4. To define tabs, use any of the following options:

- Set tabs by clicking the tab button until the type of tab you want appears. Then click the ruler to place the tab.

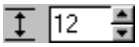


- Move a tab by dragging the tab marker on the ruler.
- Remove a tab by dragging the tab marker up or down off the ruler.

5. To set margins, drag the indent markers on the ruler.



6. To set line spacing, change the setting with the Line Spacing control.



Director adjusts line spacing to match the size of the text you are using.
If you change the line spacing setting, Director stops making automatic adjustments. To resume automatic adjustments of spacing, enter 0 in the Line Spacing box.

7. To set paragraph alignment, click one of the alignment buttons.

8 To change the kerning of selected characters, change the value of the Kerning option.

9 Set spacing before and after paragraphs by choosing Modify > Paragraph and using the Spacing Before and After options.

Formatting entire cast members

Director can apply formatting changes to entire cast members. This process is much faster than manually opening each cast member and applying changes. Any change you apply to a cast member affects all the text within the cast member.

To format text cast members:

1. In a Cast window or on the Stage, select the cast members you want to change.

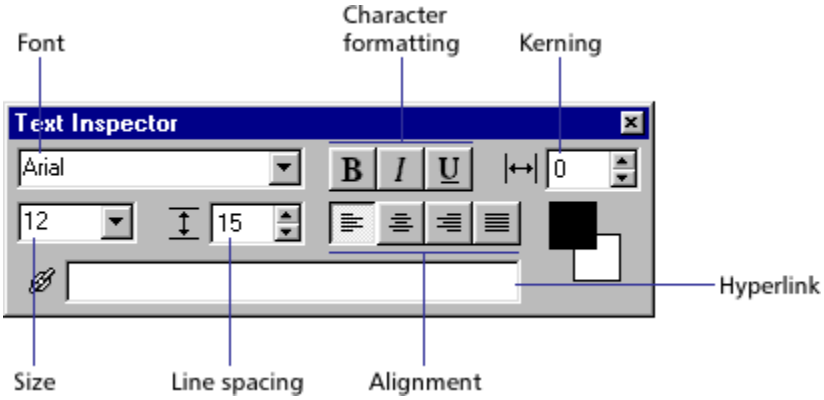
   You can select as many cast members as you want to change.

2. Use the Text Inspector, Modify > Font, or Modify > Paragraph to make formatting changes.

   The change affects all text in the selected cast members.

Formatting with the Text Inspector

The Text Inspector provides many of the most useful formatting controls in a compact floating window for use on the Stage or with entire cast members in the Cast window.



Most of the formatting controls also appear at the top of the Text window and in the Font and Paragraph dialog boxes.

To display the Text Inspector:

Choose Window > Inspector > Text, or press Control+T (Windows) or Command+T (Macintosh).

About anti-aliased text

Anti-aliased text is text that uses color variations to make its jagged angles and curves look smoother. Director activates anti-aliasing by default. You can change this setting in the Text tab of the Property Inspector (see **Setting text or field cast member properties**.) Anti-aliasing functions the same way for embedded fonts and for system fonts that have not been embedded (see **Embedding fonts in movies**).

Using anti-aliased text dramatically improves the quality of large text on the Stage, but it can blur or distort smaller text. Experiment with the size settings to get the best results for the font you are using.

Director can anti-alias all outline (TrueType, PostScript, and embedded) fonts, but not bitmap fonts. When you select a font that cannot be anti-aliased, the message "This font cannot be anti-aliased" appears in the Font dialog box below the font list. (Display the Font dialog box by selecting text or a text sprite and then choosing Modify > Font.)

About kerning

Kerning is a specialized form of spacing between certain pairs of characters that look best when they overlap slightly, such as A and W (AW). Kerning dramatically improves the appearance of large text for headlines, but it often does not improve the appearance of text at small font sizes.

If the Kerning option is on in the Text tab of the Property Inspector, Director kerns all the characters in the cast member according to standard kerning tables (see **Setting text or field cast member properties**). The setting you enter in the Text window or Font dialog box (see **Formatting characters**) supplements the standard kerning.

Finding and replacing text

Use the Find > Text command to quickly search for and replace text in the Text, Field, or Script window. All searches start at the insertion point and search forward.

To search and replace text:

1. Choose Window > Text, Window > Field, or Window > Script to open the window in which you want to search.
2. Place the insertion point at the position where you want the search to begin.
3. Choose Edit > Find > Text.
4. Enter the text you want to search for in the Find box.
5. Enter the text you want to use in place of the found text in the Replace box.
6. To specify the cast members in which to search, choose a Search option:

- Cast Member *Name* limits the search to the current cast member.
- Cast *Cast Name* limits the search to cast members in the current cast.
- All Casts extends the search to all cast members in all casts.

7. To set additional search options, select Wrap-Around, Whole Words Only, or Case Sensitive.

- Wrap-Around specifies whether or not Director returns to the beginning of text once it reaches the end. If you select this option but not All Casts, Director continues searching from the top of the current text after it reaches the bottom of the window. If you select both options, Director searches all cast members of the same type (either text, field, or script, depending on where you initiated the search), beginning with the currently selected cast member and returning to the first cast member of that type if necessary.
- Whole Words Only searches only for occurrences of the specified whole word.
- Case Sensitive searches only for text with the same capitalization as the text in the Find box.

Creating a hyperlink

In the Text Inspector, you can turn any selected range of text into a hyperlink that links to a URL or initiates other actions. Director automatically adds standard hyperlink formatting to the selected text so that it initially appears with blue underlining. You can turn off this formatting in the Text tab of the Property Inspector. See **Setting text or field cast member properties**.

The following procedure describes how to add a hyperlink to selected text. To make a hyperlink actually do something, you need to write an `on hyperlinkClicked` event handler. See **on hyperlinkClicked**.

You can enter any string in the hyperlink box; it does not have to be a URL. The string cannot contain a double quotation mark or the Lingo continuation character.

To define a hyperlink:

1. Select the text you want to define as a hyperlink.
2. Choose Window > Inspector > Text to open the Text Inspector.
3. In the Hyperlink box, enter the URL to which you want to link, or enter any message you want to send to the `on hyperlinkClicked` handler. Then press Enter (Windows) or Return (Macintosh).

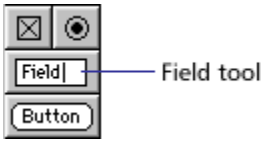
Working with fields

Working with field cast members is similar to working with text. Just as with text cast members, you edit fields on the Stage or in a window and apply formatting with the Text Inspector. Not all text formatting options are available for fields: you cannot apply spacing, tabs, or indents to individual paragraphs within fields. Alignment settings apply to every paragraph in the field.

To create a field cast member:

1 Do one of the following:

- Choose Insert > Control > Field.
- Click the Field tool in the Tool palette and then drag on the Stage to define the area of the field.



The field is created and an insertion point is placed at the beginning of the field.

2 Enter the text for the field. When you are finished, click outside the field to exit the field.

To specify field settings:

Choose Window > Field, or double-click a field cast member in the Cast window.

- If necessary, use the Previous Cast Member and Next Cast Member buttons to navigate to the field you want to edit. See **Setting text or field cast member properties**.

Using editable text

Editable text lets users enter text on a Web page, customize a game, and so on. When text is editable, editing the text changes the text cast member and all the text in the sprites where the cast member appears.

You can make text editable and let users tab between editable sprites from the Property Inspector (see **Setting text or field cast member properties**) or from Lingo.

You can make a text sprite editable in only a certain range of frames in the Score.

To make a text sprite editable in a range of frames:

1. Select a range of frames within a sprite.

   You can select an entire sprite, or Shift-Alt-click (Windows) or Shift-Option-click (Macintosh) to select frames within a sprite.

2. Click the Text or Field tab of the Property Inspector using the Graphical view.

3. Click Editable.

To control whether text is editable with Lingo:

Set the editable property. See **editable**.

To have Lingo specify whether pressing Tab opens the next sprite for editing:

Set the `autotab` property. See **autoTab**.

Converting text to a bitmap

Use Convert to Bitmap to change a text or field cast member to a bitmap. The converted graphic can then be edited in the Paint window. Once you convert a cast member to a bitmap graphic, you cannot undo the change.

This command works only with text and field cast members. You can't convert a shape to a bitmap.

To convert text to a bitmap:

1. In the Cast window, select the cast members to convert.
2. Choose Modify > Convert to Bitmap.

   Director converts the cast members to bitmaps.

Mapping fonts between platforms for field cast members

Director uses a file named Fontmap.txt to map fonts in fields between the Windows and Macintosh platforms. When you create a new movie, Director looks for Fontmap.txt in the same folder as the Director application.

The version of Fontmap.txt included with Director assigns fonts as shown in the following table. These settings provide the best equivalents of common system fonts on both platforms.

Windows font	**Macintosh font**
Arial	Helvetica
Courier	Courier
Courier New	Courier
MS Serif	New York
MS Sans Serif	Geneva
Symbol	Symbol
System	Chicago
Terminal	Monaco
Times New Roman	Times (Because Times New Roman is larger than Times, Fontmap.txt assigns a smaller point size.)

Fontmap.txt also determines the scaling of fonts and how special characters such as bullets and symbols are translated between platforms. Again, the default settings are correct for nearly all applications, but you can edit the settings if necessary.

Setting text or field cast member properties

Use the Property Inspector to view and change settings for selected text cast members. In addition to standard Name and Unload properties, you can specify whether text is editable while the movie plays, improve performance with pre- rendering, and control anti-aliasing and kerning.

To view or change text or field cast member properties:

1. Select a text cast member.
2. To display the Property Inspector, choose Modify > Cast Member > Properties or choose Window > Properties > Inspector.
3. If necessary, click the Member tab using the Graphical view.

   The following noneditable settings are displayed:

- The cast member size in kilobytes
- The cast member creation and edit dates
- The name of the last person who modified the cast member

4. To view or edit the cast member name, use the Name field.
5. To add comments about the cast member, use the Comments field.
6. To specify how Director removes the cast member from memory if memory is low, choose an option from the Unload pop-up menu.
7. To change the text of the cast member, click Edit.
8. Click the Text or Field tab of the Property Inspector using the Graphical view.
9. To determine how Director places text within the boundaries of the cast member, choose a Framing option:

- Adjust to Fit expands the text box vertically when text that is entered extends beyond the current size of the box.
- Scrolling attaches a scroll bar to the right side of the text box. This is useful when there is a large amount of text. Note that the scroll bar will be drawn direct to Stage; this means that even if another cast member is in front of a cast member containing a scroll bar, the scroll bar will appear frontmost.
- Fixed retains the original size of the text box. If you enter text that extends beyond the limits of the box, the text is stored but not displayed. You can set up scrolling with Lingo (see **Controlling scrolling text with Lingo**).
- Limit to Field Size (available only for field cast members) displays only the amount of text that fits within the field's bounding rectangle.

10. To set editing and display options, choose from the following options:

- Editable makes the cast member editable while the movie plays (see **Using editable text**).
- Wrap increases the vertical size of the text box or field on the Stage so that all text is visible.
- Tab to Next Editable Item advances the text insertion point to the next editable sprite on the Stage when the user presses Tab.
- Direct to Stage (text cast members only) makes text display more quickly by rendering it directly to the Stage without composing it with other sprites. This prevents other sprites from appearing over the text and limits the ink options to Copy.
- Use Hypertext Styles (text cast members only) makes hypertext links appear as they do in a Web browser, initially using blue underlining, and then red once the link has been visited. (See **Creating a hyperlink**.)

11 To make text of a text cast member appear on the Stage more quickly, choose a Pre-Rendering option:

- Pre-Render makes Director render text when the current member is loaded instead of when the member first appears on the Stage.
- None provides no pre-rendering.
- Copy Ink optimizes the pre-rendering for Copy Ink. (This option renders text more quickly than Other Ink.)
- Other Ink pre-renders the text for all other ink types.

12 If you choose a Pre-render option, you can make text appear on the Stage even more quickly by selecting Save Bitmap. See **Using the Save Bitmap feature for pre-rendered text**.

13 To control how Director anti-aliases text for a text cast member, choose an Anti-Alias option:

- All Text anti-aliases all the text in the text block.
- Larger Than anti-aliases only text larger than the point size entered in the Points field.
- None turns off anti-aliasing for the current cast member.

Anti-aliasing dramatically improves the appearance of large text, but it can blur or distort smaller text. Experiment with the size setting to get the best results for the font you are using. (See **About anti-aliased text**.)

14 To control how Director kerns text, choose a Kerning option.

Kerning often does not improve the appearance of text at small point sizes. See **About kerning**.

- All Text kerns all the text in the cast member according to the standard kerning table.
- Larger Than kerns only text larger than the point size entered in the Points field.
- None turns off kerning for the current cast member.

Using the Save Bitmap feature for pre-rendered text

The Save Bitmap feature works in tandem with pre-render options to display a buffer image of your text while your user waits for the actual text to load. This feature is useful when you're working with a large amount of anti-aliased text. (The Save Bitmap feature is different from the Convert to Bitmap menu command, which converts a text cast member into a bitmap image.)

You can also use the Save Bitmap feature with pre-render options if you're using special text characters for an audience not equipped to display them. For example, using Save Bitmap enables a non-Japanese system to display a text sprite that contains Japanese characters. Note, however, that the Save Bitmap option adds to the file size. The feature works with static text, but not with editable or scrolling text.

To use the Save Bitmap feature for pre-rendered text:

1. Select the text sprite.
2. On the Text tab of the Property Inspector, select from the Pre-Render pop-up menu:
   - If the text sprite's ink is Copy Ink, select Copy Ink.
   - If the text sprite's ink is any type of ink other than Copy Ink, select Other Ink.

   For this procedure to work, you must make the correct selection from the Pre-Render pop-up menu. You can determine the Sprite's ink on the Sprite tab of the Property Inspector.
3. Select Save Bitmap.

Formatting chunks of text with Lingo

The Director interface lets you format a variety of text characteristics, such as the font, size, style, and line spacing. Using Lingo, you can format text dynamically as the movie plays. You can also use Lingo to rapidly format text during authoring.

Formatting text with Lingo

Lingo can format text in an entire cast member or any specific chunk of text using the following properties.

- To select or identify a chunk of text in a field cast member, use the `selStart` and `selEnd` cast member properties. These properties identify the first and last characters of a text selection. See **selStart** and **selEnd**.
- To refer to a selected chunk of text, use the `selection` cast member property. See **selection (text cast member property)**.
- To specify the font for a text cast member, field cast member, or chunk expression, set the `font` cast member property. See **font**.
- To specify the character size for a text cast member, field cast member, or chunk expression, set the `fontSize` property. See **fontSize**.
- To specify the line spacing for a field cast member, set the `lineHeight` property. See **lineHeight (cast member property)**.
- To specify the style for a text cast member, field cast member, or chunk expression, set the `fontStyle` property. See **fontStyle**.
- To specify the drop shadow size for the characters in a field cast member, set the `boxDropShadow` property. See **boxDropShadow**.
- To specify additional spacing applied to a chunk expression in a text cast member, set the `charSpacing` property. See **charSpacing**.
- To specify the foreground color for a field cast member, set the foreColor property. See **color()**.

Applying paragraph formats with Lingo

Lingo can control paragraph formatting such as alignment and indenting for a chunk expression.

- To set text alignment for a text or field cast member, set the `alignment` property. See **alignment**.
- To set line spacing in points for a text cast member, set the `fixedLineSpace` property. See **fixedLineSpace**.
- To add pixels below paragraphs in a text cast member, set the `bottomSpacing` property. See **bottomSpacing**.
- To add pixels above paragraphs in a text cast member, set the `topSpacing` property. See **topSpacing**.
- To specify line spacing in a field cast member, set the `lineHeight` property. See **lineHeight (cast member property)**.
- To add pixels to the first indent in a chunk expression in a text cast member, set the `firstIndent` property. See **firstIndent**.
- To set the left indent (in pixels) of a chunk expression in a text cast member, set the `leftIndent` property. See **leftIndent**.
- To set the right indent (in pixels) of a chunk expression in a text cast member, set the `rightIndent` property. See **rightIndent**.
- To specify or obtain a list of tabs that are in a chunk expression in a text cast member, set or test the `tabs` property. See **tabs**.

Formatting text or field cast members with Lingo

In addition to formatting text in any chunk expression, Lingo can specify anti- aliasing and kerning for an entire text cast member and control the appearance of the text's bounding rectangle.

Setting anti-aliasing and kerning with Lingo

Use Lingo to specify anti-aliasing and kerning for a text cast member.

- To specify whether Director anti-aliases text in a text cast member, set the `antiAlias` cast member property. See **antiAlias**.
- To specify the size at which anti-aliasing in a text cast member takes effect, set the `antiAliasThreshold` cast member property. See **antiAliasThreshold**.
- To specify automatic kerning for a text cast member, set the `kerning` cast member property. See **kerning**.
- To specify the size at which automatic kerning for a text cast member takes effect, set the `kerningThreshold` cast member property. See **kerningThreshold**.

Formatting text boxes with Lingo

Lingo can specify the type of box that surrounds a text or field cast member. For field cast members, Lingo can also specify box characteristics such as borders, margins, drop shadows, and height.

- To specify the type of box around a text or field, set the `boxType` cast member property. See **boxType**.
- To specify the size of the border around a field, set the `border` field cast member property. See **border**.
- To specify the size of the margin inside a field's box, set the `margin` field cast member property. See **margin**.
- To specify the size of the drop shadow for a field's box, set the `boxDropShadow` field cast member property. See **boxDropShadow**.
- To specify the height of a field's box on the Stage, set the `pageHeight` field cast member property. See **pageHeight**.

Setting text autotabbing and wrapping with Lingo

Lingo can set text autotabbing and wrapping.

- To specify autotabbing for text or field cast members, set the `autoTab` cast member property. See **autoTab**.
- To specify whether lines wrap in a field cast member, set the `wordWrap` cast member property. See **wordWrap**.

Controlling scrolling text with Lingo

Lingo can scroll text and determine the location of specific text within the text box for text and field cast members. For example, this statement sets the `scrollTop` value for the text cast member called Discussion to 0, which makes its first line appear at the top of its scrolling field:

```
(member "Discussion").scrollTop = 0
```

This procedure can be useful for making a scrolling field automatically scroll back to the top.

- To scroll up or down by a specific number of pages in a text or field cast member, use the `scrollByPage` command. See **scrollByPage**.
- To scroll up or down by a specific number of lines in a text or field cast member, use the `scrollByLine` command. See **scrollByLine**.
- To determine the number of lines that appear in a field cast member on the Stage, set the `lineCount` cast member property. (This property doesn't apply to text cast members.) See **lineCount**.
- To determine a line's distance from the top edge of a text or field cast member, use the `linePosToLocV()` function. See **linePosToLocV()**.
- To determine the number of the line that appears at a specific vertical position in a text or field cast member, use the `locVToLinePos()` function. (This measures the distance from the top of the cast member, not what appears on the Stage.) See **locVToLinePos()**.
- To determine the point in a text or field cast member that is closest to a specific character, use the `charPosToLoc()` function. See **charPosToLoc()**.
- To determine the character that is closest to a specific point in a text or field cast member, use the `locToCharPos()` function. See **locToCharPos()**.
- To check or set the distance from the top of the line that is currently visible to the top of the box for a scrolling field or text cast member, test or set the `scrollTop` cast member property. See **scrollTop**.

Checking for specific text with Lingo

The Lingo operators `contains` and equals (=) are useful for checking strings. The `contains` operator compares two strings to see whether one string contains the other. The equals operator can determine whether a string is exactly the same as the contents of a field cast member. Use these operators to check whether a specified string is in a field cast member. See **contains**.

You can also use Lingo to evaluate strings returned by the `text` property of a text or field cast member. See **text**.

Modifying strings with Lingo

As time passes or other conditions change, you may want to update and change text. For example, you may want to frequently update a text sprite that displays the user's name or a description of a musical selection that the user is currently streaming from a Web site.

- To set the entire content of a text or field cast member, set the `text` cast member property to a new chunk of text. The chunk can be a string or another text cast member. See **text**.
- To combine character strings, use the `&` and `&&` operators. The `&` operator attaches the second string to the end of the first string. The `&&` operator includes a space between two strings when they are combined. See **& (concatenator)** and **&& (concatenator)**.
- To insert a string of characters into another string, use the `put...after`, `put...into`, or `put...before` command. The `put...before` command places the string at the beginning of another string. The `put...into` command replaces a specified chunk expression with another chunk expression. The `put...after` command places the string at the end of another string. See **put...after**, **put...before**, and **put...into**.
- To delete a chunk expression from a string of text, use the `delete` command. See **delete**.

Sound, video, and synchronization: Overview

You can give your movie added appeal by including a soundtrack, a voice- over, ambient noises, or other sounds. Adding digital video to your movie creates even more interest. Digital video not only offers high-quality real-time image animation and sound but also supports new types of media such as QuickTime VR.

With Director, you have control over when sounds start and stop, how long they last, their quality and volume, and a number of other effects. Using Shockwave Audio, you can compress sounds for easier distribution and stream them from an Internet source.

Director supports QuickTime video for Windows and Macintosh, and Video for Windows (AVI). QuickTime is a multimedia format in its own right. It offers sophisticated sound features and can include graphics in many formats, including basic navigation of QuickTime VR2 files. For a list of supported QuickTime formats, see Apple Computer's Web site at www.apple.com. To use QuickTime, you must also obtain QuickTime 3 or later from Apple.

Director's media synchronization features let you synchronize events in a movie to precise cue points embedded in sound and digital video.

Sound and video make significant demands on a computer's processing power, so you may need to manage them carefully to make sure they do not adversely affect your movie's performance.

Lingo gives Director more flexibility when playing sound and digital video and can help overcome performance concerns. You can use it to play sound and digital video in ways not possible with the Score alone. Using Lingo, you can do the following:

- Turn sound on and off in response to movie events.
- Control sound volume.
- Control the pan of a sound relative to the pan of a QuickTime VR movie.
- Preload sound into memory, queue multiple sounds, and define precise loops.
- Precisely synchronize sound, digital video, and animation.
- Turn digital video on and off on demand and control individual video tracks.
- Control QuickTime VR.

Note: You can export movies or portions of movies as QuickTime or AVI videos. See **Exporting digital video and frame-by-frame bitmaps**.

Importing internal and linked sounds

Director handles sounds as either internal or linked. You can determine whether a sound is internal or linked when you import it. Each type of sound has advantages for different situations.

Director stores all the sound data for an internal sound cast member in a movie or cast file and loads the sound completely into RAM before playing it. After an internal sound is loaded, it plays very quickly. This makes internal sound best for short sounds, such as beeps or clicks, that recur frequently in your movie. For the same reason, making a large sound file an internal sound is not a good choice, since the sound may use too much memory.

Director does not store sound data in a linked sound cast member. Instead, it keeps a reference to a sound file's location and imports the sound data each time the sound begins playing. Because the sound is never entirely loaded into RAM, the movie uses memory more efficiently.

Also, Director streams many sounds, which means it begins playing the sound while the rest of the sound continues to load from its source, whether on disk or over the Internet. This can dramatically improve the downloading performance of large sounds. Linked sounds are best for longer sounds such as voice-overs or nonrepeating music.

Director can stream the following sounds:

- QuickTime, Shockwave Audio, and MP3 sounds that are linked via a URL
- QuickTime, Shockwave Audio, MP3, AIFF, and WAV sounds that are linked to a local file

Director imports AIFF and WAV sounds (both compressed and uncompressed), AU, Shockwave Audio, MP3, and Macintosh System 7 sounds. For best results, use sounds that have 8- or 16-bit depth and a sampling rate of 44.1, 22.050, or 11.025 kHz.

To import a sound:

1. Choose File > Import.
2. Choose sound files to import.
3. To determine whether the imported sounds will be internal or linked sounds, choose a Media option:
   - Standard Import makes all the selected sounds internal sound cast members.
   - Link to External File makes all the selected sounds linked.
4. Click Import.

Note: If you're authoring on a Macintosh computer that has an audio input or microphone attached, you can record sounds directly into a movie's cast by choosing Insert > Media Element > Sound. The Sound command opens the Macintosh sound recording dialog box. Director for Windows has no equivalent feature.

Setting sound cast member properties

You can use sound cast member properties to make a sound loop, change its name, change the external sound file it's linked to (if it's a linked sound), and set its unload priority.

To set sound cast member properties:

1. Select a sound cast member.
2. Click the Sound tab of the Property Inspector.

   There are several noneditable options in the Sound tab of the Property Inspector:

- The duration of the sound
- The sample rate, sample size, and channels

3. To make the sound play continuously, click Loop. See **Looping a sound**.
4. To play the sound, click the Play button.
5. Click the Member tab in the Property Inspector.

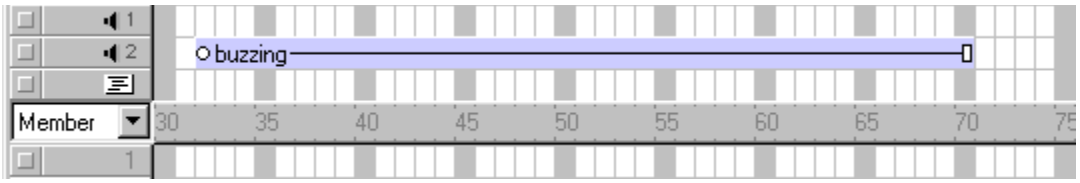
   The following noneditable settings are displayed:

- The cast member size in kilobytes
- The cast member creation and edit dates
- The name of the last person who modified the cast member

6. Use the Name field to view or edit the cast member name.
7. To change the external sound file to which the cast member is linked (if it is a linked sound), enter a new path and file in the Filename field. You can also use the Browse button to select a new file.
8. To specify how Director removes the cast member from memory if memory is low, choose an option from the Unload pop-up menu. See **Controlling cast member unloading**.

Controlling sound in the Score

You control sounds in the Score in much the same way that you control sprites. You place sounds in one of the two sound channels at the top of the Score and extend the sounds through as many frames as required.



Unless you use a behavior or other Lingo to override the Score's sound channels, sounds play only as long as the playback head is in the frames that contain the sound. After a sound begins playing, it plays at its own speed. Director cannot speed up or slow down sounds. If a sound is not set to loop, it stops playing at the end, even if the sprite specifies a longer duration. See **Looping a sound**.

Note: You can speed up or slow down a sound by converting it to a sound-only QuickTime movie and using the `movieRate` sprite property.

In addition to the two sound channels in the Score, Director can use up to six additional sound channels simultaneously. However, the additional channels are accessible only from Lingo or from behaviors. Available RAM and the computer's speed are the real constraints on the number of sounds Director can use effectively.

To place a sound in the Score:

1. If the sound channels are not visible, click the Hide/Show Effects Channels button at the top right side of the Score.

2. Do any of the following:

- Drag a sound cast member from a Cast window to a frame in one of the sound channels.
- Double-click a frame in the sound channel and then choose a sound from the Frame Properties: Sound dialog box. You can also preview any sound cast member in the movie from this dialog box.
- Drag a sound to the Stage to place it into the first available sound channel in the current frame of the Score.

3. Extend the sound through as many frames as necessary.

   New sounds are assigned the same number of frames as set for sprites in the Sprite Preferences dialog box. You may need to adjust the number of frames to make the sound play completely or change a tempo setting to make the playback head wait for the sound to finish. See **Synchronizing media**.

Note: Sound in the last frame of a movie continues to play (but not loop) until the next movie begins or you exit from the application. This sound can provide a useful transition while Director loads the next movie. You can stop the sound using the `puppetSound` command.

Looping a sound

You may find that you want to play a sound over and over to create a continuous sound effect, such as the sound of a person walking. A looped sound repeats as long as the playback head is in a frame where the sound is set. See **Importing internal and linked sounds**.

To make a sound loop:

1 Select a sound cast member.

2 On the Property Inspector's Sound tab, select the Loop option.

You can also loop sounds with Lingo. See **Playing sounds with Lingo**.

Using sound in Windows

The following issues are specific to managing sound for Windows:

- In Windows, a sound that is already playing in either sound channel overrides the sound in a QuickTime or AVI video or a Flash movie. It also prevents the video sound from playing even after the sound in the sound channel has stopped. Once the sound in a digital video has started, however, it overrides a sound in either sound channel.
- To mix QuickTime audio tracks with internal Director sounds, use the `soundDevice` system property to specify QT3Mix or install the Microsoft DirectSound sound driver software version 5.0 or higher (available from http://www.microsoft.com), and use the `soundDevice` property to specify DirectSound.See **soundDevice** (Note that Windows NT4 does not support DirectSound 5.) Check the **Director Support Center Web site** for the latest developments related to this issue.
- The default number of sounds that Director can mix in Windows is eight. This number can be decreased by modifying the value for MixMaxChannels in the Director.ini file in the Director folder.

Playing sounds with Lingo

Lingo lets you play and control sounds regardless of the settings in the Score. You can use Lingo to play sounds, turn them on and off, and play external sounds that aren't cast members. Using Lingo to play sounds lets you control the exact timing of when sounds start and stop. Lingo also allows you to play only part of a sound cast member or play several sounds in succession without interruption.

Sounds played by Director play at the volume set in the computer's sound level control. You can use Lingo to modify the computer's sound level to suit the needs of your movie, or to modify the volume of the sound channel itself.

You can also use Lingo to control and stream Shockwave Audio. See **Playing Shockwave Audio and MP3 audio with Lingo**.

Playing sound cast members

Once a sound has been imported as a cast member, you can control many aspects of how the sound is played.

To play sound cast members regardless of the settings in the Score:

Use the `queue()` and `play()` functions. The `queue()` function loads the sound into Director's RAM buffer so that it can be played immediately when called for. The `play()` command starts the sound playing. If you omit the `queue()` function, the sound may not play immediately when called for. See **queue()**and **play() (sound)**.

The following statements load the sound called Siren into RAM and start it playing in sound channel 1:

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

To queue more than one sound to play in succession:

Use the `queue()` function to list each sound in the order you want them to play. If you queue them before they are played, Director plays the sounds with no pauses between the sounds. Once the sounds are queued, only one `play()` command is needed.

These statements queue the sound members Explosion and Siren and play them in succession in sound channel 2:

```
sound(2).queue(member("Explosion"))
sound(2).queue(member("Siren"))
sound(2).play()
```

To control how a queued sound plays:

Include optional parameters in a property list within the `queue()` function. See **queue()**.

When `setPlayList()` is used, any previously set queue of sounds will be replaced by the new playlist.

Once sounds are queued, you can still control whether the queue is obeyed. You can choose to interrupt loops with the `breakLoop()` command, or to pause playback with the `pause()` function. The `playNext()` command lets you skip immediately to the next sound in the queue. See the *Lingo Dictionary* for details on these and other sound commands.

Playing external sound files

To play external sound files that aren't cast members:

Use the sound playFile command. See **sound playFile**.

Playing external sound files from disk minimizes the amount of RAM used to play sounds. However, since the computer can read only one item from disk at a time, loading cast members or playing more than one sound from disk can cause unacceptable pauses when you use the `sound playFile` command.

Controlling sound channels

You can use Lingo to make actions in a movie dependent on whether a sound is playing. Lingo lets you determine whether a sound is playing in a particular sound channel and control how a channel plays sound.

- To determine whether a specific channel is playing a sound, use the `isBusy()` function. See **isBusy()**.
- To turn off the current sound in a specific channel, use the `setPlayList()` command with `[ ]` as the new play list. This will delete the entire sound queue and leave the current sound playing. Use the `stop()` command to stop the currently playing sound. See **setPlaylist()**and **stop() (sound)**.
- To fade a specific channel's sound in and out, use the `fadeTo()` function. See **fadeTo()**.
- To control a specific sound channel's volume, specify the `volume` property. See **volume (sound channel)**.
- To control the left-to-right panning of a sound, specify the `pan` property. See **pan (sound property)**.

About Shockwave Audio

Shockwave Audio is a technology that makes sounds smaller and plays them faster from disk or over the Internet.

Shockwave Audio can compress the size of sounds by a ratio of up to 176:1 and is streamable, which means Director doesn't have to load the entire sound into RAM before it begins playing. Director starts to play the beginning of the sound while the rest of the sound is still streaming from its source, whether coming from disk or over the Internet. When used properly, the Shockwave Audio compression and streaming features provide fast playback of high- quality audio, even for users with relatively slow modem connections to the Internet.

Compression quality in Shockwave Audio

Although Shockwave Audio uses advanced compression technology that alters original sounds as little as possible, the more a sound is compressed the more it is changed.

Set the amount of compression by choosing a bit rate setting in any of the Shockwave Audio Xtras. The bit rate is not related to sampling rates you may have used in other audio programs. Try compressing the same sound at several different bit rates to see how the sound changes.

Choose the bit rate appropriate for the intended delivery system (modem, ISDN, CD-ROM, hard disk, and so on), the type of movie, and the nature of the sound itself. Voice-over sound quality, for example, may not need to be as high as that of music. Test the sound on several systems to find the right balance between quality and performance.

The more compressed a sound is, the faster it streams. If you choose to use a high quality and low degree of compression, a slow delivery system may not be able to send the data fast enough, resulting in gaps during playback. Most developers choose 16 Kbps for the best results over the Internet.

The following table suggests some general guidelines for setting the bit rate for different delivery systems. It also provides a rough estimate of perceived quality for different rates of compression. Note that real transmission times may be slower than the times shown in this table, depending on network traffic and server load.

Delivery	**Bit rate**	**Quality**
T1	64 to 128 Kbps	Equal to source material
ISDN or CD-ROM	32 to 56 Kbps	FM stereo to CD
28.8 modem	16 Kbps	FM monaural or good-quality AM
14.4 modem	8 Kbps	Telephone

Note: Any sound compressed at less than 48 Kbps is converted to monaural.

Compressing internal sounds with Shockwave Audio

Shockwave Audio can compress any internal sounds in a movie. Although internal sounds are not streamed, compressing them with Shockwave Audio dramatically decreases the size of the sound data in a movie, shortens the download time from the Internet, and saves disk space.

You can use Shockwave Audio settings to specify compression settings for internal sound cast members. The compression settings you choose apply to all internal sound cast members. You cannot specify different settings for different cast members.

You can choose compression settings at any time, but compression occurs only when the Director movie is compressed with the Create Projector, Save as Shockwave Movie, or Update Movies commands. When creating a projector, Director compresses sounds only if the Compressed option is turned on in the Projector Options dialog box. Compressing sounds can substantially increase the time required to compress a Director movie. See **Creating projectors**.

Note: Shockwave Audio does not compress SWA or MP3 audio sounds.

When you distribute a movie that contains sounds compressed with Shockwave Audio, the SWA Decompression Xtra is already included in the Shockwave player. If you compress sounds in Shockwave format in a projector, you must provide the SWA Decompression Xtra for the projector.

To have Director compress internal sound cast members when you create a projector, save a movie as Shockwave, or update the movie:

1. Choose File > Publish Settings.
2. Select the Compression tab.
3. Select Compression Enabled to turn on compression.
4. Choose a setting from the kBits/second pop-up menu.
5. Select Convert Stereo to Mono if you want to convert a stereo file to monaural.

   At rates lower than 48 Kbps, all sounds are converted to monaural.
6. Click OK.

Streaming linked Shockwave Audio and MP3 audio files

Director streams sounds that have been compressed with Shockwave Audio as well as MP3 audio files, from either a local disk or a URL. Before you can set up a streaming Shockwave Audio cast member, you must create a Shockwave Audio or MP3 file.

To create external Shockwave Audio files, do one of the following:

- In Windows, choose Xtras > Convert WAV to SWA and choose the WAV files to convert.
- On the Macintosh, use the Peak LE 2 software to export Shockwave Audio sounds.

For both methods, the audio settings are similar to those for using Shockwave Audio to compress internal sounds. See **Compressing internal sounds with Shockwave Audio**.

Note: Converting WAV to SWA does not compress IMA compressed sounds.

To stream a linked Shockwave Audio or MP3 sound:

1. Choose Insert > Media Element > Shockwave Audio.

   This creates a cast member that controls the streaming Shockwave Audio.

2. In the SWA Cast Member Properties dialog box that appears, click Browse and choose a Shockwave Audio file on a local disk, or enter a URL in the Link Address box.

   Unless you choose a file in the same folder as the movie, the movie always links to the exact location you specify. Be sure to link to the correct location.

3. Set the remaining cast member properties in the Property Inspector as follows:

- To set the volume of the sound, use the Volume slider in the SWA tab of the Property Inspector.
- To choose the sound channel for the sound, choose a number from the Channel pop-up menu in the SWA tab. To avoid potential conflicts, choose Any. This will cause the sound to play in the highest numbered available sound channel.
- To specify the size of the stream buffer, use the Preload option in the SWA tab. Director attempts to load enough sound data to play for the specified time in seconds. This prevents gaps in sounds played over slow or interruption-prone Internet connections.

4. Drag the Shockwave Audio cast member to a sprite channel (*not* one of the sound channels) to create a sprite. Extend the sprite through all frames in which the sound should play, or use the tempo channel to make the movie wait for the end of the sound. See **Synchronizing media**.

   You cannot place streaming audio cast members in the sound channels. The sound streams from the source location when the movie plays.

Playing Shockwave Audio and MP3 audio with Lingo

Use SWA Lingo to preload and control SWA and MP3 sounds, and to determine how much of the sound has streamed over the Internet.

Lingo that controls other types of sounds can also control streaming SWA and MP3 sounds by controlling the sound channel that the sound plays in.

- To preload part of a streaming sound file into memory, use the `preLoadBuffer member` command. See **preLoadBuffer member**.
- To specify the amount of a streaming cast member to download before playback begins, set the `preLoadTime` cast member property. See **preLoadTime**.
- To determine what percentage of a streaming sound file has actually played, test the `percentPlayed` cast member property. See **percentPlayed**.
- To determine the percent of a streaming file that is already streamed from an Internet server, test the `percentStreamed` cast member property. See **percentStreamed**.
- To specify the sound channel in which a streaming sound plays, set the `soundChannel` property. See **soundChannel**.
- To begin playback of a streaming cast member, use the `play member` command. See **play member**.
- To pause a streaming sound file, use the `pause member` command. See **pause member**.
- To stop a streaming sound file, use the `stop member` command. See **stop member**.
- To determine the state of a streaming sound file, test the `state` cast member property. See **state**.
- To determine whether an error occurred when streaming a sound file, use the `getError()` function. See **getError()**.
- To obtain a string describing an error that occurred when streaming a sound file, use the `getErrorString()` function. See **getErrorString()**.
- To determine the length of a streaming sound file, use the `duration` cast member command. See **duration**.
- To determine the bit rate of a streaming sound cast member, test the `bitRate` cast member property. See **bitRate**.
- To determine the original bit depth of a streaming sound, test the `bitsPerSample` property. See **bitsPerSample**.
- To determine the sample rate of the original sound used for a streaming cast member, test the `sampleRate` cast member property. See **sampleRate**.
- To determine the number of channels in a streaming sound, test the `numChannels` streaming cast member property. See **numChannels**.
- To specify a streaming sound's volume, specify the `volume` streaming cast member property. See **volume (cast member property)**.
- To specify a streaming sound file's URL, set the `URL` cast member property. See **URL**.
- To obtain or set the copyright text in a streaming sound file, test or set the `copyrightInfo` cast member property. See **copyrightInfo**.

Importing digital video

When you import QuickTime or AVI digital video, the cast members you create always remain linked to the original external file, even if you choose the Standard Import option. When you distribute a movie, you must always include all digital video files along with the movie.

QuickTime must be installed on a computer in order to author or play back a movie that contains QuickTime digital video.

Director converts an AVI video to QuickTime when it plays one on a Macintosh.

For security reasons, Shockwave will link to media on a local disk only if it is in a folder named dswmedia. To test movies in a browser locally before uploading them to your Web server, place the movie, linked casts, and linked media in folders within a dswmedia folder, and use relative links to refer to them. In order for them to be accessible from your server, you must use file and folder names that do not have spaces or capital letters, and that have recognized file extensions like .dcr and .gif. For more information, see the **Director Support Center Web site**.

To import a digital video:

1. Choose File > Import.
2. Choose QuickTime or AVI (Windows only) from the Files of Type pop-up menu.
3. Choose digital video files to import.

   Because digital video is always imported as linked, you do not have to select an option in the Media pop-up menu.
4. Click Import.
5. Select QuickTime or AVI as the import format.

   If you choose QuickTime, Director imports the video as a QuickTime Asset Xtra, which provides additional playback options. See **Setting digital video cast member properties**.

Using the Video window

Whether a digital video is a cast member or a sprite on the Stage, you can preview it in the Video window. There is a different version of the window for QuickTime and AVI movies.

To open the Video window, do either of the following:

- Double-click a digital video cast member.
- Choose Window > QuickTime or Window > AVI.

The Video window appears.

If you are working with a QuickTime digital video, a video controller bar appears, and you can start and stop the movie as you want. With AVI, you can click the movie to start and stop it.

Setting digital video cast member properties

Use cast member properties to control the media in a digital video, specify how it is framed and whether it plays direct-to-Stage, and set other important options.

To set digital video cast member properties:

1. Select a digital video cast member in the cast.
2. Click the Member tab of the Property Inspector.

   There are several noneditable options in the Member tab of the Property Inspector:

- The cast member size in kilobytes
- The cast member creation and edit dates
- The name of the last person who modified the cast member

3. Use the Name field to view or edit the cast member name.
4. To change the external file to which the cast member is linked, enter a new path and file in the Filename field. You can also use the Browse button to select a new file.
5. To specify how Director removes the cast member from memory if memory is low, choose an option from the Unload pop-up menu. See **Controlling cast member unloading**.
6. Click the QuickTime or AVI tab to set the remaining properties.
7. To determine how a movie image appears within the sprite bounding rectangle when the movie is rotated, scaled, or offset, set Framing options:

- Crop displays the movie image at its default size. Any portions that extend beyond the sprite's rectangle are not visible. For more information, see **Cropping digital video**.
- Center is available only if Crop is selected. It determines whether transformations occur with the cast member centered within the sprite or with the cast member's upper left corner aligned with the sprite's upper left corner.
- Scale fits the movie inside the bounding rectangle.

8. To determine how the video plays back, set options in the bottom portion of the window:

- Show Video displays the video portion of the digital video. If this option is turned off, the video portion does not play. Deselect this option and select Sound if you want to play only the audio portion of a movie.
- Play Sound plays the sound portion of the digital video.
- Direct to Stage allows QuickTime or AVI drivers installed on the computer to completely control the video playback. For more information, see **Playing digital video direct-to-Stage**.
- (QuickTime only) Show Controller displays a controller bar at the bottom of the video if Direct to Stage is selected.
- Paused stops the digital video when it first appears on the Stage (while playing the Director movie).
- Loop replays the digital video continuously from the beginning to the end and back to the beginning.
- Preload loads the cast member into memory when the movie starts. For more information, see **Preloading digital video**.
- (QuickTime only) Streaming begins playing the video while the rest of the video continues to load from its source.

9. If Direct to Stage is selected, choose a Playback option in the top portion of the window to specify how to synchronize the video to its soundtrack.

- Sync to Sound makes the digital video skip frames (if necessary) to keep up with its soundtrack. The digital video may also take less time to play.
- Play Every Frame (No Sound) makes every frame of the digital video appear but does not play the soundtrack, since the video cannot play the soundtrack asynchronously while the video portion plays frame by frame. Depending on the data rate of the digital video, the sprite may play more smoothly with this option selected, but this is not a certainty. In addition, playing every frame may cause the digital video to take more time to play.

10 If Play Every Frame (No Sound) is selected, choose options from the Rate pop-up menu to set the rate at which a digital video plays:

- Normal plays each frame at its normal rate, and no frames are skipped.
- Maximum plays the movie as fast as possible while still displaying each frame.
- Fixed plays the movie using a specific frame rate. Enter the number of frames per second in the field at the right. Use this option only for digital videos that use the same frame rate for each frame of the movie.

Playing digital video direct-to-Stage

Director can play digital video using a feature called Direct to Stage. Direct to Stage allows QuickTime or AVI drivers installed on the computer to completely control the video playback.

Direct to Stage often provides the best performance from a digital video, but there are two disadvantages to using it:

- The digital video always appears in front of all other sprites on the Stage, no matter which channel contains the sprite.
- Ink effects do not work, so it is difficult to conceal the video's bounding rectangle with Background Transparent ink.

When Direct to Stage is off, Director layers a digital video on the Stage exactly like other sprites, and Background Transparent ink works normally. (Matte ink does not work for digital videos.)

To set Direct to Stage options:

1. Select a digital video cast member.
2. Click the QuickTime or AVI tab in the Property Inspector.
3. Select or deselect Direct to Stage.
4. If Direct to Stage is selected, choose a Playback option:

- Sync to Soundtrack makes the digital video skip frames (if necessary) to keep up with its soundtrack. The digital video may also take less time to play.
- Play Every Frame makes every frame of the digital video appear but does not play the soundtrack, since the video cannot play the soundtrack asynchronously while the video portion plays frame by frame. Depending on the data rate of the digital video, the sprite may play more smoothly with this option selected, but this is not a certainty. In addition, playing every frame may cause the digital video to take more time to play.

5. (QuickTime only) If Direct to Stage is on, select Show Controller to display a controller bar below the movie to allow the user to start, stop, and step through the movie.

Controlling digital video in the Score

Add a digital video cast member to a movie just as you would add any other sprite. Digital videos begin playing when the playback head reaches the frame containing the video sprite. Use the QuickTime or AVI tab of the Property Inspector to make the movie pause or loop. See **Setting digital video cast member properties**.

If there's a white bounding rectangle around the video, use the Background Transparent ink to remove it. Inks don't work if Direct to Stage is turned on (see **Playing digital video direct-to-Stage**). Matte ink does not work for any type of digital video.

To create a digital video sprite:

1. Drag a digital video cast member to any sprite channel in the Score.
2. Extend the sprite through as many frames as necessary.

Playing complete digital videos

A digital video, like a sound, is a time-based cast member. If you place a video in just a single frame of the Score, the playback head moves to the next frame before Director has time to play more than a brief instant of the video.

To make sure that Director plays an entire digital video, do one of the following:

- Create a tempo setting in the tempo channel using the Wait for Cue Point option in the Frame Properties: Tempo dialog box. This option keeps the playback head from moving to the next frame until a cue point in the video has passed or, if there are no cue points, until the end of the video is reached. For more information, see **Synchronizing media**.
- Use Lingo or behaviors to make the playback head stay in a frame until the end of the video or until a certain cue point passes. See **Synchronizing media with Lingo**.
- Extend the video through enough frames to give it time to play all the way through.

QuickTime VR

You can use a QuickTime VR movie in a Director movie by inserting it as you would any other QuickTime cast member. To get the best performance, turn on Direct to Stage (see **Playing digital video direct-to-Stage**).

Playing digital video with Lingo

Lingo can take advantage of the most important and powerful features of digital video. Besides playing digital video linearly, Lingo can pause, stop, and rewind a video. These abilities are useful for jumping to segments within a digital video and for emulating a typical digital video control panel. This last feature is especially useful for AVI digital video, which has no control panel of its own.

Lingo also lets you work with individual tracks in a digital video by determining the tracks' content and position and turning these tracks on and off.

Controlling digital video playback with Lingo

The following are ways you can control digital video with Lingo.

- To turn on looping in a digital video cast member, set the digital video's `loop` cast member property to `TRUE`. See **loop (cast member property)**.
- To determine the current time of a digital video sprite, check the sprite's `currentTime` property. See **currentTime**.
- To pause a digital video sprite, set the sprite's `movieRate` property to 0. See **movieRate**.
- To start a paused digital video sprite, set the sprite's `movieRate` property to 1. See **movieRate**.
- To play a digital video sprite in reverse, set the sprite's `movieRate` property to -1. See **movieRate**.
- To rewind a digital video sprite to the beginning, set the sprite's `movieTime` property to 0. See **movieTime**.
- To control a digital video sprite's playback rate, set the sprite's `movieRate` property to the desired rate. See **movieRate**.
- To mix QuickTime audio tracks with internal Director sounds (necessary only in Windows), use the `soundDevice` system property to specify QT3Mix. See **soundDevice**.

Determining digital video content with Lingo

The following are ways that Lingo can determine a digital video's content.

- To determine the time units a digital video cast member uses, check the video's `timeScale` cast member property. See **timeScale**.
- To determine whether a digital video is QuickTime or AVI, check the digital video's `digitalVideoType` cast member property. See **digitalVideoType**.
- To determine the number of tracks in a digital video sprite or cast member, check the digital video's `trackCount` sprite or cast member property. See **trackCount (cast member property)** and **trackCount (sprite property)**.
- To determine which type of media a digital video track contains, check the digital video's `trackType` sprite or cast member property. See **trackType (cast member property)** and **trackType (sprite property)**.
- To determine the start time of a track in a digital video sprite or cast member, check the digital video's `trackStartTime` sprite or cast member property. See **trackStartTime (cast member property)** and **trackStartTime (sprite property)**.
- To determine the stop time of a track in a digital video sprite or cast member, check the digital video's `trackStopTime` sprite or cast member property. See **trackStopTime (cast member property)** and **trackStopTime (sprite property)**.
- To determine whether a sprite's track is enabled to play, check the digital video's `trackEnabled` sprite property. See **trackEnabled**.
- To obtain the text at the current time from a text track in a digital video sprite, check the digital video's `trackText` sprite property. See **trackText**.
- To determine the time of the track just before the current time in a digital video, check the digital video's `trackPreviousSampleTime` cast member property and `trackPreviousKeyTime` sprite property.
- To determine the time of the next sample after the current time in a digital video, check the digital video's `trackNextSampleTime` cast member property and `trackNextKeyTime` sprite property. See **trackNextSampleTime** and **trackNextKeyTime**.

Turning on and off digital video tracks with Lingo

By turning a digital video's soundtracks on or off, you can play just the animation or control which sounds play.

To control whether individual digital video tracks play:

Use the setTrackEnabled command. See **setTrackEnabled**.

Controlling QuickTime with Lingo

Lingo can control QuickTime in ways that aren't available for AVI. You can use Lingo to control a QuickTime video's appearance and sound volume. For QuickTime VR, you can use Lingo to pan a QuickTime VR digital video and specify what happens when the user clicks or rolls over portions of the video.

You can set the rotation, scale, and translation properties for either a QuickTime cast member or a sprite.

- To determine whether a cast member or sprite is a QuickTime VR digital video, test the `isVRMovie` property. See **isVRMovie**.
- To obtain a floating-point value that identifies which version of QuickTime is installed on the local computer, use the `quickTimeVersion()` function. See **quickTimeVersion()**.
- To control a QuickTime sprite's sound volume, set the `volume` sprite property. See **volume (sprite property)**.
- To set the internal loop points for a QuickTime cast member or sprite, set the `loopBounds` sprite property. See **loopBounds**.

Applying masks for QuickTime

Director provides specific Lingo properties for applying masks to QuickTime digital videos.

- To use a black-and-white cast member as a mask for QuickTime media rendered direct-to-Stage, set the `mask` cast member property. See **mask**.
- To control the way Director interprets a QuickTime video's mask cast member property, set the `invertMask` property. See **invertMask**.

Responding to user interaction

Lingo lets you control how QuickTime VR responds when the user clicks a QuickTime VR sprite. Use Lingo to specify how Director handles image quality, clicks and rollovers on a QuickTime VR sprite, clicks on hotspots, and interactions with QuickTime VR nodes.

- To set the codec quality to use when the user drags on a QuickTime VR sprite, set the `motionQuality` sprite property. See **motionQuality**.
- To specify the codec quality to use when a QuickTime VR panorama image is static, set the `staticQuality` sprite property. See **staticQuality**.
- To enable or disable the specified hotspot for a QuickTime VR sprite, use the `enableHotSpot` command. See **enableHotSpot**.
- To control how Director passes mouse clicks on a QuickTime sprite, set the `mouseLevel` sprite property. See **mouseLevel**.
- To find the approximate bounding rectangle for a specific hotspot in a QuickTime VR sprite, use the `getHotSpotRect()` function. See **getHotSpotRect()**.
- To specify the name of the handler that runs when the pointer enters a QuickTime VR hotspot that is visible on the Stage, set the `hotSpotEnterCallback` QuickTime VR sprite property. See **hotSpotEnterCallback**.
- To find the ID of the hotspot, if any, at a specific point on the Stage, use the `ptToHotSpotID()` function. See **ptToHotSpotID()**.
- To specify the name of the handler that runs when the user clicks a hotspot in a QuickTime VR sprite, set the `triggerCallback` sprite property. See **triggerCallback**ck.
- To determine the name of the handler that runs when the pointer leaves a QuickTime VR hotspot that is visible on the Stage, set the `hotSpotExitCallback` property. See **hotSpotExitCallback**.
- To specify the ID of the current node that a QuickTime VR sprite displays, set the `node` QuickTime VR sprite property. See **node**.
- To specify the name of the handler that runs after the QuickTime VR sprite switches to a new active node on the Stage, set the `nodeEnterCallback` QuickTime VR sprite property. See **nodeEnterCallback**.
- To specify the name of the handler that runs when a QuickTime VR sprite is about to switch to a new active node on the Stage, set the `nodeExitCallback` QuickTime VR sprite property. See **nodeExitCallback**.
- To determine the type of node that is currently on the Stage, test the `nodeType` QuickTime VR sprite property. See **nodeType**.

Rotating and scaling QuickTime video

Lingo can rotate and scale QuickTime videos.

- To control the rotation of a QuickTime sprite, set the `rotation` QuickTime sprite property. See **rotation**.
- To control the scaling of a QuickTime sprite, set the `scale` QuickTime sprite property. See **scale**.

Panning QuickTime VR

Use Lingo to pan a QuickTime VR digital video without the user dragging the image.

- To set the current pan of the QuickTime VR sprite, set the `pan` QuickTime VR sprite property. See **pan (QTVR property)**.
- To nudge a QuickTime VR sprite in a specific direction, use the `nudge` command. See **nudge**.

Displaying QuickTime video

Lingo can control how a movie displays QuickTime videos.

- To specify the type of warping performed on the panorama of a QuickTime VR sprite, set the `warpMode` QuickTime VR sprite property. See **warpMode**.
- To specify a QuickTime VR sprite's current field of view, set the `fieldOfView` QuickTime VR sprite property. See **fieldOfView**.
- To swing a QuickTime VR sprite to a specific pan, tilt, or field of view, set the `swing` function. See **swing()**.

Cropping digital video

Cropping a digital video means trimming the edges off the top or sides of the movie image. Cropping doesn't permanently remove the portions you crop; it just hides them.

To crop a digital video:

1. Select the cast member in the Cast window.
2. Click the QuickTime or AVI tab in the Property Inspector.
3. Select Crop.

   Director retains the movie's original size if you resize the bounding rectangle; however, the edges of the movie will be clipped if you make the bounding rectangle too small.
4. Select Center, if you wish.

   Director centers the movie when you resize the bounding rectangle. If Center is not selected, the movie maintains its original position when you resize its bounding rectangle. Center is available only if Crop is selected.
5. Click OK.
6. Select the video in the Score.
7. Go to the Stage and drag any of the handles that appear on the selection rectangle that surrounds the video image.

Director displays only as much of the movie image as will fit in the area defined by the selection rectangle.

If you would rather scale the movie than resize it, select Scale instead of Crop in the QuickTime tab of the Property Inspector. Director scales the movie if you resize the bounding rectangle.

To use Lingo to move the image of a QuickTime video around within the sprite's bounding rectangle:

Set the digital video's `translation` QuickTime sprite or cast member property. See **translation**.

About using digital video on the Internet

In both stand-alone projectors and movies playing in Web browsers, Director can handle digital video the same way it handles all other media, or it can stream the digital video using QuickTime 4. You can link the digital video to a URL, and the movie begins to download and play the digital video when its sprite first appears on the Stage.

In order for the digital video member to stream, you must set its `streaming di` property to `TRUE`. QuickTime 4 must be installed to enable streaming.

If a streaming QuickTime file contains cue points you wish to use, you must set the text track to be preloaded (use a QuickTime editor such as MoviePlayerPro to do this). If you do not preload the text track, Director will disable the cue points so it can stream the file without entirely downloading it first.

You can also import an RTSP (Real Time Streaming Protocol) stream as a QuickTime cast member. The rtsp:// URL must end with the file name extension *.mov* so that Director knows it should be treated as a QuickTime stream.

When using streaming digital video in a movie distributed on the Internet, keep these points in mind:

- The video begins to play immediately unless the member's `pausedAtStart` property is set to `TRUE` or the `controller` member property is set to `TRUE`.
- Once a digital video begins to download, the download continues until it is finished, even if the sprite no longer appears on the Stage. Use the `percentStreamed` member property to test how much of the media has been downloaded.The feature works with QuickTime videos only. See **percentStreamed**.

Preloading digital video

You can eliminate the delay caused by loading a digital video from disk during a Director movie by loading it at the beginning of the movie. You can preload an entire digital video (or as much of the video as will fit into available memory) using the Property Inspector.

To preload a digital video:

1. Select a digital video cast member in the Cast window.
2. Click the QuickTime or AVI tab in the Property Inspector.
3. Select Preload.

This option uses the `preLoad` or `preLoadMember` command. If there is not enough memory to load the entire movie, Director loads only what will fit into memory. If this option is turned off, Director does not load the movie into memory and instead plays it from disk. This results in slower animation speeds, since each frame must be retrieved from disk before it is played.

Synchronizing media

To pause the playback head until a specified cue point in a sound or digital video is reached, you can use the Wait for Cue Point option in the Tempo dialog box. You can also use this function to wait for the end of the sound or digital video, even if it has no cue points. Cue points can also be used to trigger events that will be interpreted by Lingo. See **Synchronizing media with Lingo**.

For example, you can use cue points to make text appear in time with narration. First, use a program such as Peak LE 2 to place cue points in the sound file that correspond to the times when you want the text to appear on stage. In Director, use the Tempo dialog box to pause the playback head at the frame where the corresponding text appears until the voice-over reaches the proper cue point.

For an animated demonstration, see the **Lingo Cuepoint** movie.

In Windows, use Sound Forge 4.0 or later or Cool Edit 96 or later to define cue points (called markers or regions within these programs). (See the Readme Windows Sound Loop-Cue.txt file in the Director application folder for instructions on doing this.)

On the Macintosh, use Sound Edit 16 2.07 or later, or Peak LE 2 or later, to define cue points in AIFF and Shockwave Audio sounds, and in QuickTime digital videos.

Note: You can insert cue points into QuickTime files only on the Macintosh; however, the cue points can be used on both platforms.

AVI digital video does not support cue points.

To use cue points:

1. Place cue points in a sound file or (on the Macintosh only) in a QuickTime file.

   Use an audio editing program to define cue points in both sounds and digital videos.

2. Import the sound or digital video into Director.

Note: Digital video is always linked, whether you choose the Standard Import option or the Link to External File option in the Import dialog box.

3. Place the sound or digital video in a channel in the Score and extend it through all the frames in which you want it to play.
4. Double-click the frame in the tempo channel where you want the playback head to wait for a cue point.
5. In the Tempo dialog box, choose Wait for Cue Point.
6. Select the sound or digital video from the Channel pop-up menu.
7. Choose the cue point to wait for from the Cue pop-up menu.

   Select the End or Next cue point or any named or numbered cue point in the sound or digital video. Director recognizes the end of a sound regardless of whether you've defined cue points ahead of time.

When the movie plays, the playback head pauses at the frame until the cue point passes.

Synchronizing media with Lingo

By writing Lingo that performs an action when a cue point is reached in a sound or QuickTime file, you can synchronize a movie with sound or digital video.

- To set up Lingo that runs when the movie reaches a cue point in a sound or QuickTime file, put the Lingo in an `on cuePassed` handler. See **on cuePassed**.
- To determine whether a sound or QuickTime file has passed a specific cue point, use the `isPastCuePoint()` function. See **isPastCuePoint()**.
- To find the ordinal number of the last cue point passed in a sound or QuickTime file, use the `mostRecentCuePoint` function. See **mostRecentCuePoint**.
- To obtain a list of names for the cue points in a specific sound or QuickTime file, test the `cuePointNames` property. See **cuePointNames**.
- To obtain a list of times for cue points in a specific sound or QuickTime file, test the `cuePointTimes` property. See **cuePointTimes**.

Using interactive media types: Overview

To add complex media and new capabilities to your movie, you can use Flash movies, other Director movies, PowerPoint presentations, and ActiveX controls. Each of these multimedia formats has interactive capabilities that are preserved by Director.

A Flash movie in a Director movie provides a vector-based, scalable, interactive animation that is optimized for use on the Web.

Director movies within other Director movies simplify complex productions. A linked movie appears within another movie as a single cast member, saving you the trouble of managing extra cast members and Score data. Using discrete movies also helps you manage file size for easier downloading.

A PowerPoint presentation can be a starting point for a Director movie. Director converts the entire presentation into a movie file that works almost exactly like the original presentation. Director converts all the presentation media into cast members, lays them out on the Stage, and generates Score data to control events.

ActiveX controls in Director can manage ActiveX application resources from within a movie. ActiveX controls provide a variety of features, including Web browsing, spreadsheet functions, and database management. ActiveX controls function as normal sprites in a movie. ActiveX controls work only in Director for Windows.

Using Flash Movies

You can incorporate Flash vector-based animation in your Director movies, projectors, and Shockwave movies for the Web simply by importing a Flash movie into Director and using it like any other cast member. Effects that once required multiple versions of a bitmap cast member—such as blending one shape into another—can now be accomplished with a single, small Flash movie.

Director can import Flash 2.0 or later. It supports new features of Flash 4, including editable text.

In Director, you can control nearly every Flash movie property—including playing, rewinding, and stepping forward and backward through any Flash movie, adjusting quality settings, and turning sound on or off—using Lingo commands.

In Flash, you can create cross-platform Windows and Macintosh movies and then play or manipulate them in Director. You can create Flash movies that communicate with your Director movie by sending events that Director scripts can capture and process. You can store entire Flash movies in the Director cast file, or you can link to external Flash movies. Director automatically loads the Flash movie it encounters in the Score into memory from disk, from a network drive, or from anywhere on the Internet.

Flash movies are particularly effective for use in Shockwave movies because, as vector-based media, they are extremely small and therefore load much more quickly than most other media types. Because Flash movies are vector-based, you can scale and rotate them while still maintaining their sharpness. For example, you can create splash screens for your Director Shockwave movies that load with lightning speed and entertain your users while the rest of the Director movie streams into memory, or you can create interactive maps in Flash that users can pan across or zoom in on to reveal details with vector- based precision.

Adding a Flash movie cast member

All Flash cast members added to a Director movie must have been created with Flash 2.0 or later and saved in Flash format (SWF).

The following procedure explains how to create a Flash cast member and set properties for it at the same time. Many of these properties can also be set using the Property Inspector.

To add a Flash movie as a cast member:

1. Choose Insert > Media Element > Flash Movie. (You can also import a cast member by using File > Import or by dragging and dropping.)
2. In the Flash Asset Properties dialog box, select the Flash movie (SWF) file you want to add to your Director cast.

- To add a file from your computer or from a network drive, click Browse, select the file, and click Open.
- To add a file from a location on the Internet, click Internet, type the URL of the file, and click OK.
- Type the pathname or the URL of the file in the Link File box.

3. Set Media options:

- Select Linked to leave the actual media of the Flash movie stored in an external file. When a sprite created with this cast member appears on the stage in a Director movie, Director will automatically load the file into memory by looking in the location specified in the Link File box. Deselect Linked to have Director copy the Flash movie into the cast.
- Select Preload to require Director to load the entire Flash movie into memory before playing the movie's first frame. Deselect Preload to have Director start playing the movie immediately while continuing to stream the cast member into memory. This option is available if you selected Linked.

4. Select Playback options to control how a Flash movie sprite plays in a Director projector, in a Shockwave Director movie, and while you are authoring in Director:

- Image displays images from the Flash movie when it plays. When Image is deselected, the images are invisible.
- Sound enables any sound in the Flash movie to play. When Sound is deselected, the movie plays without sound.
- Paused displays only the first frame of the movie without playing the movie. When Paused is deselected, the movie begins playing immediately when it appears on the Director Stage.
- Loop makes the movie play again from frame 1 once it finishes. When Loop is deselected, the movie plays once and stops.
- Direct to Stage displays the movie when it appears on the stage with the fastest, smoothest playback. Deselect Direct to Stage to have Director draw the entire sprite first in memory with other sprites and apply ink effects before actually displaying it. The disadvantage of Direct to Stage is that the movie always appears on top of other sprites, regardless of the channel in which it appears in the Score, and ink effects don't work.

5. Specify a Scale value by typing the percentage to scale the cast member.
6. Specify a Quality value:

- Select a high-quality setting to have the Flash movie play with anti-aliasing turned on, which slows down performance; choose Auto-High to have Director start playing the movie with anti-aliasing on but turn it off if it can't play the movie at the required frame rate.
- Select a low-quality setting to turn off anti-aliasing but speed up performance; choose Auto-Low to have Director start playing the movie without anti-aliasing but turn on anti-aliasing if it can do so and continue playing the movie at the required frame rate.

7 Select a Scale Mode value to control how the Flash movie's sprites are scaled on Stage:

- Show All maintains the movie's aspect ratio and, if necessary, fills in any gaps along the horizontal or vertical dimension using the movie's background color.
- No Border maintains the movie's aspect ratio by cropping the horizontal or vertical dimension as necessary without leaving a border.
- Exact Fit stretches the movie to fit the sprite exactly, disregarding the aspect ratio.
- Auto-Size adjusts the sprite's bounding rectangle to fit the movie when it is rotated, skewed, or flipped. This option always sets the scale to 100% in the Director score.
- No Scale places the movie on the Stage with no scaling. The movie stays the same size no matter how you resize the sprite, even if it means cropping the movie.

8 Select a Rate value to control the tempo at which Director tries to play the Flash movie:

- Normal plays the Flash movie at the tempo stored in the Flash movie.
- Fixed plays the movie at a rate you specify by typing a value in the entry box.
- Lock-Step plays a frame of the Flash movie for each Director frame.

Note: A Flash movie will not play faster than the frame rate set in the Director movie.

9 When you have finished selecting options, click OK.

Director adds the Flash movie to the cast.

Note: You can also use Lingo extensions to adjust these and other properties of the Flash movie. See **Controlling a Flash movie with Lingo**.

About using a Flash movie in a Director movie

Once you have added a Flash movie to the Director cast, using it in your movie is as simple as dragging it to the stage and positioning it where you want it. You can then use the Flash movie sprite in much the same way that you use other sprites.

When working with a Flash movie on the Stage, keep these points in mind:

- A Flash movie's animation plays only as long as the Flash movie's sprite is actually on the Stage. (Flash movies resemble digital video and sound sprites in this regard.)
- Because a Flash movie uses a vector format, you can stretch the movie's sprite without loss of the movie's clarity.
- You can rotate, skew, scale, or flip a Flash movie just as you would a vector shape or bitmap.
- If the movie is set up to play direct-to-Stage, the movie will always appear on top of other sprites, regardless of the channel in which it is placed, and ink effects will be ignored.
- Only the Copy, Transparent, Background Transparent, and Blend ink effects work with Flash movies.
- Blend and color settings are supported for Flash sprites just as they are for vector shapes.

Controlling a Flash movie with Lingo

Lingo gives you precise control over the way Director streams and displays a Flash movie. You can use Lingo to check and control member streaming, zoom and colorize the Flash asset, and pan the Flash image. To see an animated demonstration of controlling a Flash movie with Lingo, see the **Flash Lingo** movie.

When a movie is playing, Lingo can change the Flash cast member's properties. Some cast member properties, such as the `flashRect` and `frameRate` cast member properties, are valid only after the Flash movie's header has streamed into memory.

Director provides the following Lingo that lets you manage how Director uses a Flash movie.

- To control whether changes to a Flash movie cast member immediately appear in sprites that use the cast member, set the cast member's `broadcastProps` property. See **broadcastProps**.
- To control whether a Flash movie is stored in an external file, set the `linked` property. See **linked**.
- To control which frame of a Flash movie Director uses for the Flash movie's thumbnail image, set the `posterFrame` property. See **posterFrame**.
- To display a list of a Flash movie's current property settings in the Message window, use the `showProps` command. See **showProps()**.

Controlling a Flash movie's appearance with Lingo

Lingo can control how a Flash movie appears on the Stage and which part of the Flash movie appears in its sprite's bounding rectangle. Lingo can also skew, rotate, scale, and flip the Flash movie.

Director supports only the Copy, Transparent, Background Transparent, and Blend inks for Flash sprites.

Flipping, rotating, and skewing Flash sprites

Lingo can flip, rotate, and skew Flash sprites as the movie plays.

- To flip a Flash sprite, set the `flipH` and `flipV` sprite properties. See **flipH** and **flipV**.
- To skew a Flash sprite, set the `skew` sprite property. See **skew**.
- To rotate a Flash sprite, set the `rotation` property. Set the `obeyScoreRotation` property to specify whether a Flash sprite obeys the rotation specified in the Score.

  If `obeyScoreRotation` is set to `TRUE`, Director ignores the cast member's `rotation` property and obeys the Score rotation settings instead.

  See **rotation** and **obeyScoreRotation**.

Colorizing and blending Flash sprites

You can use Lingo to change a sprite's color and blend as the Director movie plays.

To specify the color of a Flash sprite:

Set the color sprite property. See **color (sprite property)**.

To specify the blend for a Flash sprite:

Set the `blend` sprite property. See **blend**.

Scaling Flash movies

You can use Lingo to scale Flash cast members and sprites.

To control the scaling of a Flash movie:

Set the scale and scaleMode properties. See **scale** and **scaleMode**.

To set the scale percentage of a Flash movie within its sprite's bounding rectangle:

Set the viewScale property. See **viewScale**.

Controlling a Flash movie's bounding rectangle and registration points

You can use to Lingo to control a Flash movie's bounding rectangle and to set a Flash movie's registration points.

- To control which part of a Flash movie appears within its sprite's bounding rectangle, set the `viewH`, `viewpoint`, `viewScale`, and `viewV` properties. See **viewH**, **viewPoint**, **viewScale**, and **viewV**.
- To control the default size for all new Flash sprites, set the `defaultRect` property. Use the `defaultRectMode` property to control how the default size is set. See **defaultRect** and **defaultRectMode**.
- To determine the original size of a Flash cast member, test the `flashRect` property. See **flashRect**.
- To specify a Flash movie's registration point around which scaling and rotation occurs, set the `originH`, `originMode`, `originPoint`, and `originV` properties. See **originH**, **originMode**, **originPoint**, and **originV**.
- To recenter a Flash cast member's registration point after resizing the cast member, set the `centerRegPoint` property to `TRUE`. See **centerRegPoint**.

Placing Flash movies on the Stage

Lingo can set whether a Flash movie appears at the front of the Stage and whether specific areas of a Flash movie and the Stage overlap.

- To determine whether a Flash movie plays in front of all other layers on the Stage and whether ink effects work, set the `directToStage` property. See **directToStage**.
- To determine which Stage coordinate coincides with a specified coordinate in a Flash movie, use the `flashToStage()` function. See **flashToStage()**.
- To determine which Flash movie coordinate coincides with a specified coordinate on the Director Stage, use the `stageToFlash` function. See **stageToFlash()**.
- To improve performance for a Director movie that uses a static (not animated) Flash movie, set the `static` property. See **static**.
- To control whether a Flash movie's graphics are visible, set the `imageEnabled` property. See **imageEnabled**.
- To control whether a Flash movie plays sounds, set the `sound` property. See **sound**.
- To control whether Director uses anti-aliasing to render a Flash movie, set the `quality` property. See **quality**.

Streaming Flash movies with Lingo

In addition to the Lingo that lets you stream many of Director's media types, Director offers Lingo that specifically lets you control and monitor streaming Flash movies. For general information about using Lingo to stream media in Director, see **Playing movies over the Internet: Overview**.

- To specify whether a linked movie streams or not, set the `preLoad` property. See **preLoad (cast member property)**.
- To specify how much of a Flash cast member streams into memory at one time, set the `bufferSize` cast member property. See **bufferSize**.
- To check how many bytes of a Flash movie have streamed into memory, test the `bytesStreamed` property. See **bytesStreamed**.
- To determine how much of a Flash movie is currently streamed, test the `percentStreamed` property or check the `streamSize` function. See **percentStreamed** and **streamSize**.
- To set when Director attempts to stream part of a Flash movie, set the `streamMode` property. See **streamMode**.
- To clear an error setting for a streaming Flash movie, use the `clearError` command. See **clearError**.
- To determine whether an error occurred while streaming a Flash movie, use the `getError()` function. See **getError()**.
- To check the current state of a streaming file, test the `state` property. See **state**.
- To attempt to forcibly stream a specified number of bytes of a Flash movie, use the `stream` command. See **stream**.

Controlling Flash movie playback with Lingo

You can use Lingo to control a Flash movie's tempo, to specify which frame plays, and to start, stop, pause, and rewind the Flash movie.

- To control the tempo of a Flash movie, set the `fixedRate` and `playBackMode` properties. See **fixedRate** and **playBackMode**.
- To determine the original frame rate of a Flash movie, test the `frameRate` property. See **frameRate**.
- To determine the number of frames in a Flash movie, test the `frameCount` property. See **frameCount**.
- To determine the frame number associated with a label in a Flash movie, use the `findLabel()` function. See **findLabel()**.
- To play a Flash movie starting from a specified frame, set the `frame` property or use the `goToFrame` command. See **frame (sprite property)** and **goToFrame**.
- To set whether a Flash movie starts playing immediately when the Flash sprite appears on the Stage, set the `pausedAtStart` property. See **pausedAtStart**.
- To check whether a Flash movie is playing or paused, test the `playing` property. See **playing**.
- To rewind a Flash movie to frame 1, use the `rewind sprite` command. See **rewind sprite**.
- To stop a Flash movie at its current frame, use the `stop` command. See **stop (Flash)**.
- To stop a Flash movie at its current frame but let any audio continue to play, use the `hold` command. See **hold**.

Controlling Flash movie interactivity with Lingo

Lingo can control whether a Flash movie remains interactive.

- To control whether the actions in a Flash movie are active, set the `actionsEnabled` property to `TRUE`. See **actionsEnabled**.
- To control whether buttons in a Flash movie are active, set the `buttonsEnabled` property. See **buttonsEnabled**.
- To control when a Flash movie detects mouse clicks or rollovers, set the `clickMode` property. See **clickMode**.
- To control whether clicking a button in a Flash movie sends events to sprite scripts, set the `eventPassMode` property. See **eventPassMode**.
- To determine which part of a Flash movie is directly over a specific point on the Stage, use the `hitTest` function. See **hitTest()**.
- To check whether the mouse pointer is over a button in a Flash movie, test the `mouseOverButton` property. See **mouseOverButton**.

Sending Lingo from Flash movies

A Flash 3 or 4 movie can send Lingo instructions to a Director movie. (Flash 2 movies do not support this feature.) The Lingo determines how the movie responds when the user clicks a button or the Flash movie enters a frame. In Flash, you can send a string to Lingo with a Flash `on getURL` handler. The string can be an event message or a complete Lingo statement.

In Flash, you create a button or frame and then assign it a Get URL action in which you specify the Lingo that the Flash cast member sends. To see an animated demonstration, see the **Flash Events** movie.

To set up a Flash movie to generate an event:

1. In Flash, select a button.
2. Choose Modify > Instance. In the Instance Properties dialog box, click the Actions tab and select Get URL from the Action menu.

   Do not specify anything for the Target Window option. (Director ignores this field.)
3. In the URL field, enter the Lingo that you want Flash to send to the movie.

- To specify a string to pass to an `on getURL` handler in the Director movie, enter the string. In Director, include an `on getURL` handler that receives the string from the Flash movie and reads the string as a parameter.

  For example, in Flash, you can specify this in the Network URL field:

  ```
  Dali
  ```

  In Director, you can write this handler:

  ```
  on getURL me stringFromFlash
    go to frame stringFromFlash
  end
  ```

  When the `on getURL` handler receives the text string, it reads the string and then jumps to the frame labeled Dali in the Director Score.

- To specify an event message, specify the word `event` followed by a colon, the name of a handler you will write in Director, and a parameter (if any) to pass along with the event.

  For example, in Flash, you can specify this in the Network URL field:

  ```
  event: FlashMouseUp “Dali”
  ```

  In Director, you write this handler:

  ```
  on FlashMouseUp me whichFrame
    go to frame whichFrame
  end
  ```

  When the Director script receives the `FlashMouseUp` message and the parameter, the movie jumps to the frame specified by the parameter.

- To specify a Lingo statement, specify the word `lingo`, followed by a colon, followed by the Lingo statement that you want Director to execute.

  For example, in Flash, you can specify this in the Network URL field:

  ```
  lingo: go to frame “Dali”
  ```

When Director receives the `getURL` message from the Flash movie, the movie immediately executes the Lingo statement.

You can place handlers to capture events from Flash movies in a Flash sprite or cast member script or in a frame or movie script. The event follows the normal Director message hierarchy.

Using Lingo to set and test Flash 4 variables

Two new sprite functions have been added to support variables in Flash 4 sprites: `getVariable()` and `setVariable`(). Also, a return value has been added to the `hitTest()` function.

To return a string that contains the current value of a Flash sprite variable, use the following statement:

```
getVariable( sprite X, variableName )
```

To set the current value of a Flash sprite variable to a specified string, use the following statement:

```
setVariable( sprite X, variableName, newValue )
```

Note: Be sure to pass the Flash variable's name as a string in both the `getVariable` and `setVariable` functions. Failure to do so will result in script errors being produced when the functions are executed.

To return the type of object within the Flash sprite that is currently over the specified stage location, use the following statement:

```
hitTest( sprite X, somePointLocation )
```

In previous versions of Director, there were only three possible return values for this function: `#background`, `#normal`, and `#button`. With the new Flash Asset Xtra, there is a fourth return value possible: `#editText`. This value indicates that an editable text field within the Flash sprite is over the specified location. See **getVariable()**, **setVariable()**, and **hitTest()**.

Playback performance tips for Flash movies

Performance of Flash movies can vary greatly, depending on the options in effect and the playback environment. Following are tips for getting optimal playback performance from Flash:

- If adequate for your needs, use the Low quality setting rather than High. Using Low turns off anti-aliasing, which speeds up Flash animation rendering. A handy technique is to switch the quality of the sprite to Low while displaying a fast-moving animation sequence (such as a spinning logo), and then switch the quality back to High on the fly as the animation slows down or comes to a stop. This way, performance can be improved during the part of the sequence where it would be more difficult to perceive the improved quality anyway, without sacrificing quality in the end result.
- Experiment with different system color depths to see what provides the best performance. Some display drivers are still optimized for 8 bits, so performance can be faster when running in this mode. Some graphics, such as gradients, display faster at 16 bits.
- Use Copy ink if possible. Transparency, using Background Transparent ink, requires much more processing time. If your Flash sprite is in the background (no other Director sprites are behind it), use Copy instead of Background Transparent, and author your Flash movie in such a way that its background color is the same as the background color you chose for your Director Stage.
- Use Direct to Stage if possible. Layering and transparency are not supported in this mode; however, if you just want to play a Flash movie within a box with the best performance possible, this may be the way to go.
- Make sure that the Director movie tempo is set high enough. Unless you're using Direct to Stage, your Flash movie will not play faster than the Director movie frame rate, regardless of the `frameRate` or `fixedRate` setting. For smoothest playback, set the Director frame rate to at least 30 frames per second (fps).
- Use Lock-Step or Fixed playback mode to adjust the Flash movie frame rate. Lock-Step gives the best performance, because playback of the Flash movie is synchronized to the Director movie frame for frame.
- Set the `static` property of the sprite to `TRUE` if your sprite contains no animation (such as a static block of text) and doesn't overlap other moving Director sprites. This keeps Director from redrawing the sprite every frame unless it moves or changes size.
- When modifying Flash properties using Lingo, set the properties for the sprite rather than for the cast member. Setting the properties for the cast member modifies values at the cast member level and broadcasts the change to all sprites on the Stage. This overhead can affect performance. If you have only a single sprite for the cast member, modify the sprite property directly.
- Limit the amount of Lingo that executes while the Flash movie plays. Avoid tight repeat loops between frames. The usual Director performance optimizations apply when using Flash movies.

Using Director movies within Director movies

You can import a Director movie into another movie as an internal or linked cast member, with the Import command. As with other media types, you can link to an external movie file or import the file so that it becomes internal media. The way you choose to import a movie affects its properties.

- For linked movies, cast member scripts and behaviors (sprite scripts) work as before; select Enable Scripts in the Linked Movie tab of the Property Inspector. Frame and movie scripts do not work. As with other types of linked media, the external movie file must be present on the system when the host movie plays.
- For movies imported as internal media, the movie appears as a film loop.

For both types of imported movies, the host movie controls the tempo settings, palette settings, and transitions. Settings for these functions in the imported movie are ignored.

Once it is imported, the movie appears as a cast member in the Cast window. The cast members of a movie imported as internal media also appear in the Cast window. You can animate the cast member just as you would any graphic cast member, film loop, or digital video.

To import a Director movie:

1. Choose File > Import.
2. From the Files of Type pop-up menu, choose Director Movie.
3. Select a Director movie.
4. To determine whether the movie is imported into the current movie file or linked externally, choose a Media option.

- Standard Import imports all the movie's cast members into the current cast and creates a film loop that contains the Score data.
- Link to External File creates a cast member that references the external movie file. A linked movie appears as a single cast member

5. Click Import.

To place a Director movie cast member in the current movie:

1. Do one of the following:

- For an internal movie, drag the film loop cast member to the Stage or Score.
- For a linked external movie, drag the movie cast member to the Stage or Score.

2. Extend the sprite through all the frames in which you want it to appear.
3. To change any of the movie's properties, use the Movie tab of the Property Inspector.

See **Setting linked Director movie properties**.

Setting linked Director movie properties

To determine whether a linked Director movie is cropped or scaled to fit within a sprite's bounding rectangle, you use the Property Inspector. You can also use the Property Inspector to enable the movie's scripts, mute its sounds, and specify whether it loops.

To set linked movie properties:

1. Select a linked movie cast member.
2. To display the Property Inspector, choose Modify > Cast Member > Properties or choose Window > Inspectors > Properties.
3. If necessary, display the Member tab in Graphical mode.

   The following noneditable settings are displayed:

- The cast member size in kilobytes
- The cast member creation and edit dates
- The name of the last person who modified the cast member

4. To view or edit the cast member name, use the Name field.
5. To add comments about the cast member, use the Comments field.
6. To specify how Director removes the cast member from memory if memory is low, choose one of the following options from the Unload pop- up menu:

- 3—Normal sets the selected cast members to be removed from memory after any priority 2 cast members have been removed.
- 2—Next sets the selected cast members to be among the first removed from memory.
- 1—Last sets the selected cast members to be the last removed from memory.
- 0—Never sets the selected cast members to be retained in memory; these cast members are never unloaded.

7. Click the Linked Movie tab in Graphical mode.
8. To determine how the linked movie appears within the sprite bounding rectangle, choose a Framing option:

- Crop displays the movie image at its default size. Any portions that extend beyond the sprite's rectangle are not visible.
- Center is available only if Crop is selected. It keeps the movie centered within the sprite's bounding rectangle if you resize the sprite.
- Scale resizes the movie to fit inside the bounding rectangle.

9. To determine how the linked movie plays back, set the following options

- Play Sound enables the sound portion of the linked movie. Turn this option off to mute sounds.
- Loop replays the linked movie continuously from the beginning to the end and back to the beginning.
- Enable Scripts makes all the scripts in the linked movie work the same way they do when the movie plays by itself.

Using PowerPoint presentations

You can import Microsoft PowerPoint 4 presentations into Director and then play your presentations as they are, or you can use them as a starting point for creating rich multimedia projects in Director. You can save your Director project as a stand-alone projector or as a movie for the Web. For example, you can animate your presentation by adding bars or lines one at a time, or you can synchronize music, sounds, or video to action in your movie—such as a self- running presentation with voice-over narration—or advance a presentation to the next slide after the narration on the current slide finishes.

Importing your PowerPoint presentations into Director imports the artwork, text, and transitions as individual cast members into your Director cast, converts each PowerPoint slide into a Score section, and assembles the Score for you—complete with build effects and tempo settings that pause the action when necessary. You can use Director to add sophisticated interactivity, sounds, and animations.

Lingo can add more interactivity, such as letting users decide the order in which to view slides in a kiosk, track the slides visited, and list the slides still to be viewed.

Importing a presentation into Director creates a copy of the presentation file. Because the presentation you play in Director is a copy, not the original PowerPoint file, you can enhance your presentation in Director without affecting the PowerPoint original. In most cases, the Director movie will look and act almost exactly like the PowerPoint presentation. For a list of differences, see **Comparison of PowerPoint and Director features**.

Importing a PowerPoint presentation into Director

Director can import presentations from PowerPoint 4 or later. If necessary, open the presentation in PowerPoint and use the File > Save As command to save the presentation in the PowerPoint 4 format.

Presentations that contain OLE objects or large or numerous bitmaps may require additional memory. If you run out of memory while importing a presentation, try saving the presentation as two or more files. Import those files into Director movie files and then recombine the Director movie files by cutting and pasting.

To import a PowerPoint presentation:

1. Create a new movie by choosing File > New > Movie.

   You cannot import a presentation into an existing Director movie.

2. Choose Xtras > Import PowerPoint File and choose a PowerPoint presentation.

   Director opens the PowerPoint Import Options dialog box.

3. Set up the way you want to import the presentation.

- For Slide Spacing, enter the number of blank frames that you want between each slide in the Score.
- For Minimum Slide Duration, enter the minimum number of frames a slide will take up in the Score.
- For Item Spacing, if the presentation contains build effects (other than fly effects), enter the number of frames used for each item before the next item appears on the Stage.
- For Fly Transition Item Spacing, specify the number of frames to the next keyframe, where the item is in full view. Fly transitions are imported as a Score animation. In the first frame in which the item appears, the item is positioned offstage. In general, the larger the number of frames, the smoother the fly effect will be, and the longer it will take to appear on screen.

   For more information on how Director assembles the Score, see **Using PowerPoint presentations**.

4. Click Import.

   When the presentation has been imported, save your movie.

Comparison of PowerPoint and Director features

In cases where there is no exact match for a PowerPoint feature in Director, Director uses the closest matching feature. The following tables summarize how Director handles nonmatching PowerPoint features as well as other issues that occur when importing presentations.

PowerPoint text	Director equivalent
Text Tool objects on the Slide Master	Text Tool objects on the Slide Master aren't placed in the Score. Cast members are created, but you must manually add them to the Score.
Bold or italic text	Depending on the font used, text wrap and character spacing may differ in Director.
Embossed text font effect and shadow font effect	Text appears as regular text. As a workaround, in PowerPoint you can use the Format > Shadow command to add a shadow effect to the entire text block.
Rotated text	Text is not rotated.
Text that uses unavailable fonts	Text uses the nearest matching font. Text spacing and wrapping may change as a result.
Text using non-anti-aliased fonts and italic style	Cast members are inverted (reading left to right).
Text with text wrap turned off	Text wrapping is turned on.
Colored text in Microsoft Word tables within a presentation	Text appears as black text.
Multiple line spacing settings or line spacing in text objects that use multiple fonts	One line spacing setting is used per object.

PowerPoint lines and fills	Director equivalent
Dotted and dashed lines	Solid lines are used.
Medium dashed lines	Lines are not imported. Change the line weight in PowerPoint before importing.
Lines with arrowheads	Lines don't have arrowheads.
Lines with blunt ends	Line ends are tapered or rounded, depending on the line type.
Pattern fills	The object appears as a solid object.
Arcs with fills that end flush with their fills	Arcs with fills extend beyond their fills.
Semitransparent AutoShape fills	No fill is used. It is recommended that you set the blend of the Director sprite to 50%.

From Title shading	From Center radial gradient is used.

PowerPoint bulleted lists	Director equivalent
Backgrounds for bulleted text blocks using build effects	The background appears on the Stage in the same frame in which the first bulleted item appears. If necessary, extend the background sprite's duration to an earlier frame in the Score.
Bullet builds	Bullets always build one bullet item at a time, regardless of their level.
Bulleted items with the Random Effects build effect	A single transition, selected at random, is assigned to each bullet in the text block.
Dimming of previous bulleted items	Dimming is achieved by setting the cast member to a 50% blend. The dimming color is not used.
Text with a text anchor set to top centered, middle centered, or bottom centered	The vertical anchor setting is preserved. All items in a bulleted list must have consistent settings for the centered anchor setting to be preserved.

PowerPoint transitions	Director equivalent
CheckerBoard Across transition	CheckerBoard Down transition
Fade Through Black transition	No transition
Cut and Cut Through transitions	No transition

PowerPoint objects and bitmaps	Director equivalent
Bitmaps saved in a Macintosh presentation and imported into Director for Windows, or bitmaps saved in a Windows presentation and imported into Director for Macintosh	Cross-platform bitmaps are not imported. Before importing, open the presentation on the same platform you will use for Director and save the presentation. For example, before importing into Director for Windows, first open the Macintosh presentation file in PowerPoint for Windows and save the file.
Color depth of graphics	Director creates bitmaps for all graphics in the current monitor depth when importing. For a different color depth, change the monitor setting before importing or use the Modify > Transform Bitmap command to modify the bitmap cast members.
Bitmaps with white extending beyond colored borders	Director strips white space from the outer edges of bitmaps. This may lead to differences in alignment of objects on the Stage, or objects may be stretched to fill the space.

EPS format graphics	There may be some distortion of graphics in this format when it is re-created in Director.
8-bit graphics	On occasion, some colors may not import correctly in 8-bit displays. If this occurs, set the display to a higher color depth before you import.
OLE objects (such as a Microsoft Excel chart or Microsoft Word table)	These objects are imported as bitmaps.
Solid-color shadows on clip art and charts	Solid-color shadows are reproduced as duplicates of the object and are offset by the number of pixels specified for the shadow.
Objects using shading that are created in PowerPoint on the Macintosh and imported into Director on Windows	In rare cases, Director may re-create the shaded objects in different colors.
Page numbers	Page numbers inserted on the SlideMaster using PowerPoint 4 are not imported. However, page numbers inserted into the master slide footer using PowerPoint 95 or PowerPoint 97 are imported properly after they are saved in PowerPoint 4 format.
Hidden slides	If the last slide of your PowerPoint presentation is hidden, Director will display the slide and play it back. Either remove the slide from the Score or move the end marker and script one frame to the left.
Notes pages	Objects on the Notes pages aren't placed in the Score. Bitmap cast members are created for objects such as pictures, clip art, and tables. However, text is ignored.
Rotated shapes	Changing and saving shapes several times may obscure rotation data in Director. Shapes may be imported with rotation off by 180º. Try copying the shapes to a new PPT file and saving them.
Files edited and saved using multiple versions of PowerPoint	If items are not imported as expected, or if bullet builds do not occur, saving the PPT file in multiple PPT formats may have obscured the data to Director. Try copying the slides to a new PPT file and saving.

Using ActiveX controls

In Director for Windows, you can embed ActiveX controls that let you take advantage of the technology and adapt ActiveX controls (formerly known as OLE/OCX controls) to make them function as sprites in Director. You can use ActiveX controls to manage application resources for the hosted ActiveX control—for instance, to manage properties, events, and windows and filing properties. You can also manage resources used by the ActiveX control within the Director movie.

The range of uses for ActiveX in Director is as limitless as the variety of ActiveX controls available. Using the Microsoft Web Browser control (installed with Microsoft Internet Explorer 3.0 or higher), you can browse the Internet from within a multimedia production; using the FarPoint Spreadsheet control, you can create and access spreadsheets; using the InterVista VRML control, you can explore virtual worlds; using MicroHelp's extensive library of Windows widget controls, you can build and simulate complete Windows applications.

Note: Not all ActiveX controls expose their methods and properties in all hosts. Test the controls you want to use to see how they work in Director.

Inserting an ActiveX control

You can place ActiveX controls in a Director movie and have them function as sprites. Note that this procedure is designed only for Director for Windows.

To insert an ActiveX control on the Stage:

1. Make sure that the ActiveX control you want to use in Director is installed on your system.

   Most controls have their own installation utilities provided by the manufacturer of the control.

2. Choose Insert > Control > ActiveX.

3. In the dialog box that appears, select the desired ActiveX control and then click OK. The ActiveX Control Properties dialog box appears.

   (If the desired ActiveX control does not appear in the list, it may not have been installed properly by the system. You can attempt to verify this by viewing the list of ActiveX controls in another application such as Visual Basic.)

   The ActiveX Control Properties dialog box lets you edit each ActiveX control and view information regarding each method the control supports and each event the control can generate.

4. Set the values for each property in the ActiveX control and then click OK. The ActiveX control now appears in the cast.

5. Drag the ActiveX control from the Cast to the Stage.

   Once the ActiveX control appears on the Stage, it can be repositioned and resized just like any other sprite Xtra. When you pause the movie, the ActiveX control stays in authoring mode and does not react to mouse or keyboard events. When you play the movie, the control responds to user input.

Setting ActiveX control properties

An ActiveX control describes its information using properties—named characteristics or values such

as color, text, font, and so on. Properties can include not only visual aspects but also behavioral ones. For example, a button might have a property that indicates whether the button is momentary or push-on/push-off. An ActiveX control's properties define its state—some or all of which properties may persist. Although the control can change its own properties, it is also possible that the container holding the control might change a property, in response to which the control would change its state, user interface, and so on.

When an ActiveX control is inserted into a Director movie, the properties that the control exposes can be viewed and edited by clicking the Properties tab of the Control Properties dialog box for the ActiveX Xtra. Each property exported by the ActiveX control is identified along with the current value of the property. The user edits a property value by simply clicking over the existing value with the mouse. For most properties, such as numeric or string values, the new value can be directly entered into the list using the keyboard.

In Director, all properties that an ActiveX control exports are properties of the corresponding sprite. This is the generic Lingo syntax for setting an ActiveX control property:

```
set the PropertyName of sprite X to Value
```

The generic Lingo syntax for getting an ActiveX control property is as follows:

```
put the PropertyName of sprite X into Value
```

As an example, if the Microsoft Access Calendar control is inserted into a Director movie as the second sprite on the score, the following Lingo code sets the Year property of the Calendar control to a specific year:

```
set the Year of sprite 2 to 1995
```

To get the Year property from the same Calendar control and place it into a Lingo variable named CalendarYear, you can use this Lingo code:

```
put the Year of sprite 2 into CalendarYear
```

Some ActiveX control properties are read-only, and trying to set a property for such a control will cause an error in Director.

Using ActiveX control methods

An ActiveX control describes its functionality using methods. Methods are simply functions implemented in the control that Director can call to perform some action. For example, an edit or other text-oriented control supports methods that let Director retrieve or modify the current text, perhaps performing such operations as copy and paste.

When you insert an ActiveX control in a Director movie, you can view the methods exposed by the control by clicking the Methods tab of the Control Properties dialog box for the ActiveX control. The dialog box displays each method supported by the ActiveX control and a description of the parameters for each method.

In Director, all methods that an ActiveX control supports are functions for the corresponding sprite. The generic Lingo syntax for calling an ActiveX control method is as follows:

```
put MethodName (sprite N, param1, param2, ... ) into RetValue
```

As an example, if the Microsoft Access Calendar control is inserted into a Director movie as the second sprite on the Score, the following Lingo code would increment the year displayed within the Calendar control:

```
NextYear (sprite 2)
```

For the same Calendar control, the following Lingo code would decrement the year displayed by the Calendar control:

```
PrevYear (sprite 2)
```

Parameters passed to the ActiveX control are automatically converted from their Director data types to equivalent ActiveX data types. Likewise, the return value is automatically converted from an ActiveX data type to an equivalent Director data type.

Using ActiveX control events

Each ActiveX control typically generates a variety of events. For example, a button ActiveX control may generate a `click` event when the button is pressed, and a calendar ActiveX control may generate a `dateChanged` event when the date within the calendar is changed. Director converts any event generated by the ActiveX control to a sprite event that it can handle. A list of the control's events appears in the Events tab of the ActiveX Control Properties window.

To respond to an event generated by the ActiveX control, you must write an event handler to capture the event. You can place these event handlers in movie scripts, sprite behaviors, scripts assigned to cast members, or frame behaviors. However, you normally place the handler in the behavior attached to the sprite for the ActiveX control.

As an example, if the Microsoft Access Calendar control is inserted into a Director movie as a sprite on the score, the following Lingo code would capture the `click` event from the Calendar control:

```
on click
  -- Do something interesting here.
  beep 2
end
```

A sprite behavior is a good location for this handler.

Playing movies over the Internet: Overview

A Director movie can use the Internet in various ways: hosting multiuser sessions such as chats and games, streaming movies and sounds, retrieving data from the network, and interacting with a browser. Whether it is distributed on disk or downloaded from the Internet, a movie can use an active network connection to retrieve linked files, send information, open Web pages, and perform many other network activities.

To make a movie appear in a user's browser, you can save it as a Shockwave movie and embed it in an HTML document. The movie can play from a local disk or an Internet server. When the user opens the HTML document stored on an Internet server, the movie begins streaming to the user's system, and usually begins playing after the first frame's content has been downloaded.

You can also distribute a movie over the Internet as a projector—a packaged movie that the user downloads and executes. A projector plays in a stand-alone application, not in a browser. See **About distribution formats**.

While authoring a movie, consider how the movie will be distributed and played on users' systems. If the movie will stream from an Internet source, you may need to modify the movie for the best streaming performance, and to use Director's built-in behaviors to make the movie wait while certain cast members download. Controls and Lingo commands offer methods for sending and retrieving media and other information, interacting with a browser, and monitoring downloading.

About streaming movies

When you distribute a movie on the Internet, streaming provides an immediate and satisfying experience for your users. If you do not specify streaming, your user must wait for the entire movie to download before it begins to play. A streaming movie begins playing as soon as a specified amount of content reaches the user's system. As the movie plays, the remaining content downloads in the background and appears when it is needed. Streaming can dramatically decrease the perceived downloading time.

When Director streams a movie over the Internet, it first downloads the Score data and other nonmedia information such as scripts and the size of each cast member's bounding rectangle. This data is usually quite small compared to the size of the movie's media—usually only a few kilobytes. Before starting the movie, Director then downloads the internal and linked cast members required for the first frame of the movie (or more frames if you have increased the number in the Movie Playback dialog box). After the movie starts, Director continues to download cast members (along with any associated linked media) in the background, in the order the cast members appear in the Score.

If the movie jumps ahead in the Score or uses cast members referenced only by Lingo scripts, the required cast member may not be available when necessary. If cast members are not available, the movie will either ignore them or display a placeholder, depending on how you set the streaming options in the Movie Playback Properties dialog box.

A challenge of authoring for Internet streaming is ensuring all cast members have been downloaded by the time the movie needs them. To avoid missing cast members, make sure that all the cast members required for a particular scene have been downloaded before beginning the scene. You can use Director's behaviors to wait for media in certain frames, or for particular cast members. See **About streaming with the Score and behaviors**. You can also write your own Lingo code to do the same thing. See **Checking whether media elements are loaded with Lingo**.

Director movies stream unless you turn off streaming. In addition to turning streaming off and on, you can specify that the media elements for a certain number of frames must finish downloading before the movie starts playing.

You control streaming movies by arranging sprites in the Score and controlling the movement of the playback head either with Director's behaviors or with Lingo. You can also use Lingo to specify when externally linked files are downloaded.

About network operations

Director allows a network operation to begin even though a previous network operation isn't complete. This capability, often referred to as background loading, allows Director to perform multiple operations while loading files. Because something else is happening while files are loading, the user doesn't perceive the wait.

Note: Loading data from a network is different from loading cast members in Director. Loading from a network loads data to the local disk. Loading cast members in Director means loading cast members into memory.

It's a good idea to author a Shockwave movie so that it performs other tasks while data is loading in the background. Because Internet operations require background loading, Lingo for the Internet behaves differently than Lingo commands that run within one movie. See **Using Lingo with Internet security restrictions**, **Using URLs with Lingo**, **Differences in scripting Lingo for browsers**, **Lingo that is unsupported in browsers**, and **Interacting with browsers**.

Setting movie playback options

To change basic streaming settings for a movie, you use the Movie Playback Properties dialog box. You can turn streaming off and on, specify a number of frames to download before playing the movie, and make Director display placeholders if cast members haven't been downloaded yet. The Movie Playback Properties dialog box also includes options for locking the current tempo and pausing the movie when the window is deactivated.

Turning off streaming makes sense for some types of movies. For example, a game that requires all cast members to be available at once might not be suitable for streaming. Other movies work best if the media for a certain number of frames downloads before the movie begins playing. This option is especially useful for streaming movies that were not originally designed for streaming.

Placeholders are rectangles that appear in place of media elements for cast members that have not yet been downloaded. Placeholders are useful when testing to note missing media.

You can specify streaming options any time before saving a movie as a Shockwave movie.

Note: If you want to test how a movie will stream from a server before you save the movie as a Shockwave movie, first use File > Save and Compact to make sure the data in the movie is properly ordered and that redundant data is removed.

To set movie playback options:

1. Choose Modify > Movie > Playback to define streaming options.
2. To stop the movie from streaming, deselect Play While Downloading Movie.
3. To make the movie wait for all media elements (internal and linked) for a specified range of frames, enter the number of frames in Download __ Frames Before Playing.

   By default, movies download the first frame only. Adjust this setting to the number of frames that is best for your movie.
4. To make the movie display placeholders for media elements that have not been downloaded, select Show Placeholders.

   The placeholders appear as rectangles when the movie plays.
5. To lock the movie to its current tempo settings, select Lock Frame Durations. See **Locking frame durations**.
6. To make the movie pause when its window is deactivated, turn on Pause When Window Inactive.

To set Shockwave playback options, see **Setting Shockwave playback options**.

Setting Shockwave playback options

To view Shockwave movies, your users must have the Shockwave player, which comes preinstalled on many computer systems. The player is also available for free downloading from Macromedia's Web site.

The Shockwave player includes a volume control and a standard context menu that appears when a user right-clicks (Windows) or Control-clicks (Macintosh) a movie. You can select specific playback options to include for your users when you save your movie as a Shockwave movie.

Shockwave movies loop by default. To cause a Shockwave movie to play only once, add the `Hold on Current Frame` behavior to the last frame of the movie.

To set Shockwave playback options:

1. Choose File > Publish Settings.

   The Publish Settings dialog box appears.

2. On the Shockwave tab, select the options you want your users to have:

- Volume Control lets users adjust the volume of the movie's soundtrack.
- Transport Control provides controls for rewinding, stopping, and stepping through the movie in Shockmachine.
- Zooming determines if stretchable Shockwave is allowed. You can disable zooming with Lingo by setting the `allowZooming` property. For more information, see the *Lingo Dictionary.*
- Save Local lets users save movies on their local computers for playback in Shockmachine.
- Display Progress Bar and Display Logo determine if the progress bar and logo appear while the movie loads in the browser.

About creating multiuser applications

The Shockwave Multiuser Server and Xtra let two or more users exchange information over the Internet or smaller networks. Director users rely on multiuser functionality for a wide variety of purposes, including the following:

- Creating a chat movie that allows real-time conversation
- Adding human interaction between an e-commerce site and its customers for technical support and customer service
- Conducting an online meeting with a shared “whiteboard” that each participant can write on
- Running a multiplayer interactive game

Director includes the Multiuser Xtra and a 50-user version of the server application as part of the default installation. You can modify the Multiuser.cfg file to enable 1000 users to connect to the server. For more information, see **Enabling 1000 connections to the Multiuser Server**.

The Library palette contains multiuser behaviors that add commonly used multiuser functionality to a movie, such as connecting to the Multiuser Server.

Using the Multiuser Xtra

The Multiuser Xtra takes messages sent from movies and checks them for errors. The Xtra then prepares the messages for transmission over a network. Movies that use the Multiuser Xtra can exchange information in three ways:

- By sending it to the Shockwave Multiuser Server, which then sends it to the intended movie or movies
- By establishing peer-to-peer connections directly with other movies, bypassing the Shockwave Multiuser Server
- By connecting to a text-based server such as a standard mail server or Internet relay chat server

The Multiuser Xtra extends Lingo by adding new commands and other elements to the Lingo vocabulary. You can use the Xtra with multiuser behaviors, or with other Lingo scripts.

In addition to Director, the Shockwave Player also includes the multiuser Xtra. If you're creating a projector, however, you must add the Multiuser Xtra to the movie's Xtras list. See **About Xtras**.

Using the Shockwave Multiuser Server

The Multiuser Server is a separate application from the Multiuser Xtra. The server typically runs on a separate computer from the computer running your Director movie, but it can also run on the same computer as your Director movie. After determining who should receive a message, the server sends it to the recipient. The recipient's Multiuser Xtra then receives the message from the network.

When using the server, movies can communicate with other instances of the same movie (such as when two people are using two instances of the same movie to play chess together) or with different movies (such as when chess players use a chess movie to chat with other game players running a different movie in a virtual game room).

The messages your movie sends depend on how the movie is designed. Movies can share all the types of data that Lingo supports.

In addition to simply passing information from movie to movie, the Shockwave Multiuser Server also provides functions that make it easy to create a complex multiuser movie in which you can do the following:

- Store and retrieve information such as user names or profiles in databases.
- Create groups to organize users logically, such as by teams.
- Assign attributes to groups, such as a team's score.
- Send messages directly to the server to retrieve information about it and other movies connected to the server.

Installing the Multiuser Server application

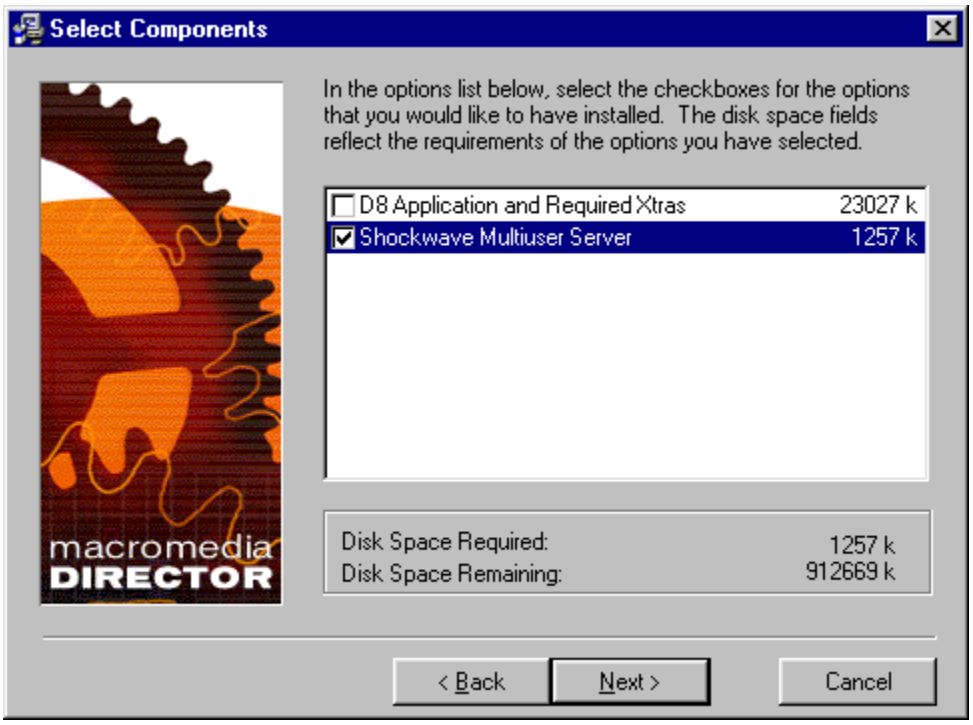
The Multiuser Server is part of the default installation that automatically occurs when you install Director.

You'll find the server application in your Director 8 application folder, in a subfolder named Shockwave Multiuser Server 2.1.

To install the Multiuser Server on a computer other than the one on which you're installing Director:

1. Use your Director 8 CD to run the Installer.
2. After you accept the legal agreement, when the dialog box with installation options appears, select Custom Install from the Install pop-up menu and select Shockwave Multiuser Server.

   The Director application does not need to be installed on the computer where you install the server.
3. To specify where the server application will install, click the Install Location pop-up menu, select or browse to a location, and then click Install.



Note: You can also copy the Shockwave Multiuser Server folder from one location to another.

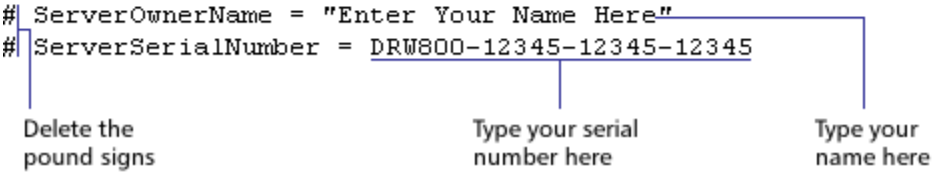
Enabling 1000 connections to the Multiuser Server

The Multiuser Server, by default, allows 50 users to connect. You can enable 1000 simultaneous connections to the server by modifying the multiuser.cfg file.

To enable 1000 connections to the server:

1. In the Multiuser Server 2.1 subfolder of your Director 8 application folder, locate the Multiuser.cfg file and open it as a text file.
2. In the Director users licensing information section, find the line that asks for the ServerOwnerName. Delete the pound sign at the beginning of the line and replace the Enter Your Name Here text with your name.
3. On the next line, delete the pound sign and replace the serial number with a valid serial number, and then save and close the file.

```
#=================================================================
# Director users licensing information
#
# If you have purchased a copy of Director 8, uncomment the
# following two entries and enter your serial number
# and name below.  This will allow the server to accept
# up to 1000 incoming connections.
#
# For some Windows computers, you may need to reconfigure the
# system to allow more connections.  Read TechNote #14107 on
# http://www.macromedia.com/support/director for more information.
#=================================================================
# ServerOwnerName = "Enter Your Name Here"
# ServerSerialNumber = DRW800-12345-12345-12345
```



Launching the Shockwave Multiuser Server

You launch the Multiuser Server using the subfolder where the server was automatically installed.

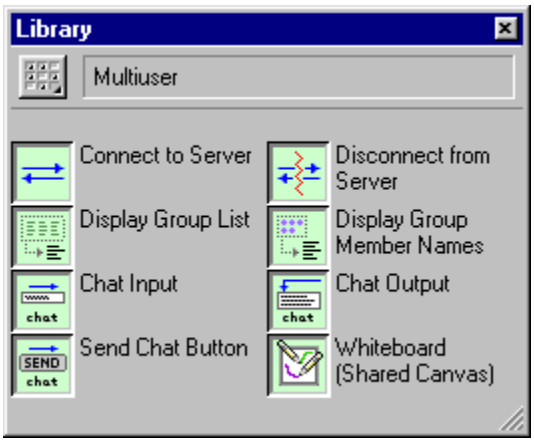
To launch the Multiuser Server:

1. In the Multiuser Server 2.1 subfolder of your Director 8 application folder, double-click the MultiuserServer icon to launch the server.
2. To determine the server's IP address and see additional information about the server, choose Status > Server.
3. Write down the server IP address or copy it to the clipboard.

   You add the IP address to the IP Address field in the `Connect to Server` behavior's Parameters dialog box. See **Connecting users to the Multiuser Server**.

Using multiuser behaviors

The Library palette includes eight Multiuser behaviors that assist you in creating multiuser applications. To access the Multiuser behaviors from the Library List, choose Internet > Multiuser.



Director Multiuser behaviors include the following:

- `Connect to Server` connects your user to the Shockwave Multiuser Server. You drag the `Connect to Server` behavior to the sprite your user will click to establish a server connection. You can attach this behavior to any type of sprite; it's usually attached to a button or bitmap. For additional information, see **Connecting users to the Multiuser Server**.
- `Disconnect From Server` lets your user disconnect from the Shockwave Multiuser Server on MouseUp. In the behavior parameters, you can specify a marker in the Score to go to after your user disconnects. You can attach the `Disconnect from Server` behavior to any type of sprite.
- You attach the `Send Chat Button` behavior to the sprite that your user will click to send information to the Multiuser Server. You can attach the `Send Chat Button` behavior to any type of sprite. In the behavior parameters, you select the sprite that contains the information to send.
- `Whiteboard (Shared Canvas)` is a behavior that works with the `Canvas` behavior and with other paintbox behaviors. You can attach the `Whiteboard (Shared Canvas)` behavior only to a bitmap sprite.
- You attach the `Chat Output` behavior to the field or text sprite that displays text that other chat members send. Behavior parameters include the ability to add a welcome message and the ability to specify the total number of text lines that will appear in the text output area.
- `Chat Input` is the behavior you attach to the text or field sprite in which your user enters information.
- You can attach the `Display Group Member Names` behavior to a sprite to display a list of group members. You can attach this behavior to a field or text sprite.
- By attaching the `Display Group List` behavior to a sprite, you're including functionality that displays a list of groups, and the members within the group. You can attach this behavior to a field or text sprite.

For more information on using behaviors, see **Behaviors: Overview**.

Connecting users to the Multiuser Server

To let your users connect to the Multiuser Server, you can use the multiuser behavior included in the Library palette. You can also write Lingo scripts that configure the Multiuser Xtra and establish communication with the server. For more information about writing Lingo for multiuser applications, see **Using the Shockwave Multiuser Sever**on the Director Support Center

Before using the `Connect to Server` behavior, you should have markers in the Score that your users will go to under different circumstances. For example, you may want the playback head to loop at a Waiting marker while a connection is attempted, then advance to a Mainchat marker when the connection is established.

Your Director movie should also contain the cast members your multiuser movie requires, such as the fields and graphics for the Connect and Disconnect buttons, and the cast members that will contain the input and output information.

To use a behavior that connects users to the server:

1. If the Library palette is not open, choose Window > Library Palette.
2. In the Library palette, click the Library List menu and choose Internet > Multiuser.
3. Drag the `Connect to Server` behavior to the sprite on the Stage that your user will click to establish a server connection.
4. In the Parameters dialog box, use the three Which Member pop-up menus to select cast members that will contain the user name, contain the user password, and display errors.
5. Use the three Which Marker pop-up menus to select the markers in the Score that your users will go to while connecting, when connected, and if the connection fails.
6. In the Server Address field, enter the server IP address. See **Launching the Shockwave Multiuser Server**.

Note: You can also connect to the server that Macromedia hosts, which allows 16 simultaneous connections, by typing trialserver.macromedia.com in the Server Address field.

7. Verify that the Server Port Number field contains 1626 as the default.
8. Either accept the default name for your movie in the Movie ID String field, or enter a new name.
9. To specify encryption during your user's login, select Encryption and enter the encryption key string in the next field. (You must also enter the encryption key string in the Multiuser Server config file.)
10. When you finish setting the parameters, click OK.

About streaming with the Score and behaviors

The easiest way to create a movie that streams well is to arrange the Score properly and use behaviors to control the playback head. Director downloads cast members in the order in which they appear in the Score. Try to arrange the Score so that events don't make the playback head jump far ahead in the Score, where cast members have not yet been downloaded. For example, if you place a menu in the first frame of a movie and a user chooses an option that sends the playback head to frame 400, the cast members for frame 400 probably won't be available right away.

To avoid this problem, begin a movie with a simple introductory scene containing a few small cast members, preferably vector shapes. You can use a streaming behavior from the Library palette to make the introduction loop until the cast members required for the next scene have been downloaded in the background.

Several behaviors included with Director control the playback head or a progress bar while media elements are downloading. These behaviors make it easy to allow enough time for downloading to catch up with action in the Score.

Looping behaviors

Looping behaviors make the playback head return (loop) to a frame or stay on the current frame until specified media elements have been downloaded, and then continue to the next frame. Attach a looping behavior to a frame in the script channel, not to a sprite.

Loop until Next Frame is Available loops the playback head to a specified frame until all the media elements required for the next frame have been downloaded.

Loop until Member is Available loops the playback head to a specified frame until a certain cast member has been downloaded.

Loop until Media at Marker is Available loops the playback head to a specified frame until all the media elements for the frame at the specified marker have been downloaded.

Loop until Media in Frame is Available loops the playback head to a specified frame until all the media elements required for a certain frame have been downloaded.

Jumping behaviors

Jumping behaviors make the playback head skip to a specified frame or marker once certain media elements have been downloaded. Attach a jumping behavior to a frame in the script channel, not to a sprite.

Jump When Member is Available moves the playback head to the specified frame once a certain cast member has been downloaded.

Jump When Media in Frame is Available moves the playback head to the specified frame once the media elements for a particular frame have been downloaded.

Jump When Media in Marker is Available moves the playback head to the specified frame once the media elements for the frame at a particular marker have been downloaded.

Checking whether media elements are loaded with Lingo

Director has several options that let an initial portion of a movie start playing as soon as the required data and cast members are available. You can use Lingo to check whether media elements have been downloaded from a network by testing the following:

- Whether a specific cast member is loaded before the movie proceeds
- Whether the cast members used in a specific frame are loaded before the frame plays

Checking whether a cast member or sprite is loaded

To determine whether a specified cast member is available locally, you use the `mediaReady` cast member or sprite property. You can check for a specific cast member or the cast member assigned to a specific sprite. When `mediaReady` returns `TRUE`, the cast member is available. See **mediaReady**.

This property always returns `TRUE` for local files. It is useful only for movies that stream from a remote server. Since playback can begin before the entire movie has been downloaded, you must make sure that the needed media elements have been downloaded as the movie plays.

Checking whether a frame's contents are loaded

Use the `frameReady()` function to determine whether all the media elements that the specified frame requires are available locally. See **frameReady()**.

Downloading files from the Internet with Lingo

Lingo uses the Internet's resources by obtaining files from the Internet. The data is copied to the local disk or cache. After data is available on the local computer, use Lingo to retrieve the data for the movie. See **Retrieving network operation results with Lingo**.

For a movie or projector playing outside a browser, background loading isn't required. However, preloading is a good idea because it improves playback performance.

All network Lingo operations that obtain data from the network begin downloading the data and return a network ID. The data isn't immediately available.

An unlimited number of network Lingo operations can take place at once. When multiple network Lingo operations run simultaneously, rely on the network ID that the function returns to distinguish which operation is complete. Be aware that running more than four operations at once usually adversely affects performance.

When using network Lingo, the current handler must finish before an operation's result is returned. For best results, place Lingo that initiates a network operation and Lingo that uses the operation's result in different handlers. An `on exitFrame` handler is a good location for checking whether an operation is complete.

To execute a network Lingo operation:

1. Start the operation.

   For example, this statement initiates a text downloading operation and assigns the network ID returned by the `getNetText()` operation to the variable `theNetID`:

   ```
   set theNetID = getNetText("http://www.thenews.com")
   ```

2. Make sure the operation finishes.

   To check an operation's status regularly until the function indicates that the operation is complete, use the `netDone()` function. See **netDone()**.

   For example, this statement loops in the current frame until the download operation is complete:

   ```
   if not netDone(theNetID) then go to the frame
   ```

3. Check whether the operation was successful by using the `netError()` function. See **netError()**.

4. Obtain the results if the operation is complete.

To cancel a network operation in progress:

Use the `netAbort` command to cancel a network operation without waiting for a result. This frees up capacity for Internet access, which allows other network operations to finish faster. See **netAbort**.

To retrieve a file as text:

1. Use the `getNetText() function` or the `postNetText` command to start retrieving text. See **getNetText()**or **postNetText**.

2. Use `netTextResult` to return the text you retrieved with `getNetText` or `postNetText`. See `netTextResult`.

To retrieve and play a new Shockwave movie from the network:

Use the `gotoNetMovie` command. See **gotoNetMovie**.

The current movie continues to run until the new movie is ready to play. After the new movie is ready, the player quits the current movie without warning and plays the new movie in the same display area as the calling movie.

To open a URL in the user's browser:

Use the `gotoNetPage` command. This command works whether the URL refers to a Shockwave Director movie, HTML, or another MIME type. See **gotoNetPage**.

You can specify that this command replace a page's content or open a new page. If the browser isn't open, the command launches the browser. If the `gotoNetPage` command replaces the page that the movie is playing in, the movie keeps playing until the browser replaces the page.

The `gotoNetPage` command is similar to Director's `open` command. It doesn't return a value.

To preload a file from the server into the cache:

Use the `preloadNetThing()` function. See **preloadNetThing()**.

The `preloadNetThing()` function initiates downloading a linked movie asset into the cache, where it is available for later use. Director can later preload the asset into memory without a download delay.

The current movie continues playing while preloading takes place.

To test whether getNetText(), preloadNetThing, or gotoNetMovie operations are complete:

Use the netDone() function. See **netDone()**.

To post information to a server and retrieve a response:

Use the postNetText command. See **postNetText**.

Retrieving network operation results with Lingo

Lingo can retrieve network operation results, such as a text result, a unique identifier for a network operation, a file's MIME type, and the date an HTTP item was last modified.

A returned network ID is for the network operation whose results are being retrieved.

To retrieve the text result of a network operation:

Use the `netTextResult()` function. See **netTextResult()**.

To retrieve the "date last modified" string from the HTTP header for a specific item:

Use the `netLastModDate()` function. See **netLastModDate()**.

To obtain the MIME type of the HTTP item:

Use the `netMIME()` function. See **netMIME()**.

To conserve memory, Director discards the results of the `netTextResult()`, `netDone()`, `netError()`, `netMIME()`, and `netLastModDate()` functions within a short time after the operation completes successfully.

- For `netTextResult()`, Director discards the results when the next operation starts.
- For the other functions, Director retains the results through seven subsequent network operations.

To determine the state of a network operation that retrieves data:

Use the getStreamStatus() function or the on streamStatus event handler. See **getStreamStatus()**or **on streamStatus**.

Using Lingo with Internet security restrictions

Because of security issues for movies that play back in browsers, the following Lingo features are unsupported for Shockwave movies playing in a browser. Many of these restrictions are imposed by the Internet environment. For details about security concerns when playing a movie on the Internet, see **Director and Internet Security**on the Director Support Center Web site.

In general, the following Lingo features are unsupported because of Internet security concerns:

- Setting `colorDepth` for the user's monitor
- Saving a movie by using the `saveMovie` command
- Printing by using the `printFrom` command
- Opening an application by using the `open` command
- Stopping an application or the user's computer by using the `quit`, `restart`, or `shutDown` command
- Opening a local file that isn't in the dswmedia folder or a subfolder of the dswmedia folder
- Pasting content from the Clipboard by using the `pasteClipBoardInto` command
- Searching for files on a user's system with getNthFileNameInFolder(), searchCurrentFolder, or searchPath

Using URLs with Lingo

In addition to the Lingo explicitly intended for use with network operations, some Lingo elements can use URLs as references to external files.

These Lingo elements can use URLs as file references in all circumstances:

- moviePath
- pathName
- unloadMovie

The following Lingo supports URLs as references to external files. If you use this Lingo in projectors or during authoring, you can avoid pauses while the file is being downloaded by first using `preloadNetThing` to download the file. After the file has been downloaded, you can use these Lingo elements with the file's URL without a delay.

When the following Lingo is used in a browser, however, you must first download the file by using the `preloadNetThing` command. If you do not, the Lingo fails.

- Using a `go to movie` statement
- Using an `importFileInto` command
- Using a `preLoadMovie` command
- Using a `play movie` command
- Using an `open window` command

These Lingo elements can use URLs to SWA sound files as file references:

- streamName
- `URL` cast member property

These Lingo elements can use URLs as file references only during authoring or in projectors:

- getNthFileNameInFolder()
- searchCurrentFolder

These elements don't work in Shockwave movies because Shockwave doesn't support movies in windows (MIAWs):

- open window
- forget window
- close window

Differences in scripting Lingo for browsers

There are some general differences in the way to script Lingo for a movie that plays over the Internet, depending on whether the movie is in a browser.

- For a movie playing in a browser, it is best to use `preloadNetThing` to load media elements into the browser's cache first. If the media elements aren't preloaded using `preloadNetThing`, linked media elements may not be present when they are needed.
- Avoid using long repeat loops in browsers; such repeat loops can make the computer appear unresponsive. As an alternative, you can split long operations into sections and execute them over a series of frames or check for user actions in an `on exitFrame` handler.
- Do not use a `repeat while` loop to check whether a network operation is complete.

Lingo that is unsupported in browsers

The following Lingo features are unsupported for movies that play back in browsers:

- Creating and managing a movie in a window (MIAW)
- Installing and managing custom menus

Interacting with browsers

Lingo lets you write and read a Prefs file within the dswmedia folder and display a string in a browser's status area.

To write to a Prefs file on a local disk:

Use the `setPref` command. See **setPref**.

After the command runs, a folder named Prefs is created inside the Shockwave player folder (in the same location as the Xtras folder). The `setPref` command can write only to that folder. The default folder locations for Windows and Macintosh are as follows:

- Windows—the \Macromed\Shockwave 8 subfolder of the system folder; the system folder is typically c:\winnt\system32 or c:\windows\system
- Macintosh—the System Folder:Extensions:Macromedia:Shockwave 8 folder

The `setPref` command can't write to a file that is on a CD.

For more information, see **Director and Internet Security** on the Director Support Center Web site.

To return the content of a file that was written by a previous setPref command:

Use the `getPref()` function. If no `setPref` command has already written such a file, the `getPref()` function returns `VOID`. See **getPref()**.

To specify text in a browser's status area:

Use the `netStatus()` function. See **netStatus**.

Note: Some browsers do not support this function.

Testing your movie

However you choose to create your movie, test it thoroughly before releasing it to the public. Make sure you test on systems with all common types of Internet connections, especially on slow modems and at busy times of day. Here is a list of things you might want to check before distributing your movie over the Internet; remember, however, that each movie has its own special needs:

- Compare a streaming version of the movie to a nonstreaming version to see if the performance is different. Some smaller movies may work better without streaming playback.
- Verify that all linked media elements appear correctly. To see if the movie correctly handles an error, try forcing the linked media elements to fail.
- Run the movie on all systems your users are likely to have. For the general public, this includes Windows 95, 98, and NT and Mac OS 8.x and 9.
- Run the movie on very slow modem connections and on fast T3 connections; problems can arise from fast as well as slow connections.
- Check for display problems on systems set to 8-, 16-, 24-, and 32-bit color. Also test as many types of monitors and display adapters as you can.
- Check for font mapping problems in your movie. If your movie uses nonstandard fonts, use embedded fonts. See **Embedding fonts in movies**.
- Check for sound problems, particularly if you stream sounds with Shockwave Audio.

About downloading speed

Developers distributing multimedia over the Internet usually limit file size, primarily because most users connect at relatively slow speeds. At 28,800 bps, it takes 30 seconds to 1 minute to download a 60K file. Using streaming playback can help you avoid some of the delays caused when downloading large files.

Movies and streaming Shockwave Audio (SWA) sounds always compete for control of the network. This can cause a noticeable problem on slower connections.

If there is heavy traffic at the Internet access point or on the Internet host, or if there is network congestion, the rate drops even lower—to as low as a few hundred bytes per second. In general, it is a good idea to assume your movies will download at about 2K per second.

The following chart shows theoretical throughput times for modems of different speeds. The speeds 14,400 and 28,800 bps are common for modems; 64 Kbps and 128 Kbps are the throughput of an ISDN line; 1.5 Mbps is the throughput of a standard high-speed Internet connection (T1).

Content	**14.4 Kbps**	**28.8 Kbps**	**64 Kbps**	**1.5 Mbps**
Small graphics and animation, 30K	30 sec	10 sec	6 sec	1 sec
Small complete movie, 100 to 200K	100 to 200 sec	50 to 100 sec	20 to 40 sec	1 sec
500K movie	500 sec	120 to 240 sec	90 sec	3 sec
1 MB movie	--	--	180 sec	6 sec

Packaging Movies for Distribution: Overview

When you finish authoring your movie, you have several choices regarding the way to prepare it for distribution. You can distribute a movie in the Shockwave format that plays in a browser or as a stand-alone projector. Stand-alone projectors can include the software necessary to play the movie, or they can use an installed Shockwave player to play the movie independent of a browser. You can also export a movie as a Java applet or as a digital video. For additional information about saving a movie as Java, see **Creating Java Applets with Director** in the Director Support Center Web site.

You can use several Director features to prepare movies for distribution. These features include determining Publish settings and deciding which Xtras to include or download. You can also preview your movie in a browser and batch-process movie files to compact them and protect them from being edited.

Shockwave browser compatibility

Shockwave works with Netscape Navigator as a plug-in, with Microsoft Internet Explorer for Windows 95 and NT as an ActiveX control, and with Microsoft Internet Explorer for Mac OS as a plug-in. Shockwave can play Director movies in the following browsers:

Browser	Version	Platform
Netscape Navigator	3.0 or later	Windows and Macintosh
Microsoft Internet Explorer	3.0 or later	Windows
Microsoft Internet Explorer	4.01 or later	Macintosh

Shockwave also works with browsers that are compatible with the plug-in architecture of Netscape Navigator 3.0, including America Online.

When it first encounters an HTML page that references Shockwave, Internet Explorer for Windows 95, 98, and NT asks the user for permission to download the Shockwave ActiveX control if it is not already installed. If the user approves, it downloads and installs the control.

Previewing a movie in a browser

You can preview a movie in a browser on your local computer to view JPEG- compressed bitmaps, and to check the movie design, Lingo, and any other performance issues related to playing a movie in a browser. Previewing a movie creates temporary Shockwave (DCR) and HTML files that open in a browser.

Note: When you use the Publish command rather than the Preview in Browser command, you can create permanent DCR and HTML files that let you view the movie in a browser.

You may notice that linked media does not work as expected when you preview a movie in a browser. Because of security restrictions, movies playing in browsers cannot read files from a local disk unless they are in the dswmedia folder (also called the support folder), which is a subfolder of the folder containing the Shockwave player.

Therefore, to preview a movie that uses linked media, you need to put the movie and all of its linked media in the dswmedia folder. The movie can open a file in a subfolder of dswmedia provided the relative paths have not changed. If you move the movie and its media to another server, the linked media will continue to work if you preserve the same folder structure. For details about security issues when playing a movie in a browser, see **Director and Internet Security** in the Director Support Center Web site.

To specify the browser to use for previewing:

1. Choose File > Preferences > Network.
2. In the Preferred Browser box, enter the path to the browser application file.

To preview a movie in a browser:

Choose File > Preview in Browser or press F12.

About Xtras

All Xtras a movie requires must be installed on your user's system when the movie runs. When you distribute a movie, you must either include these Xtras or provide the user with the means to download them. Using the Movie Xtras dialog box, you can specify the Xtras to include in a projector and whether Xtras should download for use with Shockwave movies. The Movie Xtras dialog box contains a list of the most commonly used Xtras. Including all these Xtras ensures that your movie will work in most cases but makes the projector much larger. You may want to remove Xtras you know you aren't using.

Each time you create a sprite that requires an Xtra, Director adds the Xtra to the list of required Xtras in the Movie Xtras dialog box. If you remove the sprite, Director does not remove the Xtra from the list, in case you later re- create the sprite. Director cannot detect Xtras required in Lingo code. You must manually add any Xtras required by your Lingo code to the list in the Movie Xtras dialog box. See **Managing Xtras for distributed movies**.

Managing Xtras controls the size and capabilities of the movie you distribute. Many important features in Director, such as text and vector shapes, are controlled by Xtras, as is the ability to import all types of linked media. If you don't use a feature or import a media type that is controlled by an Xtra, you should not distribute the related Xtra with your movie. This is especially true for movies distributed on the Internet.

The Shockwave player includes the Xtras that support the most common features and media types. These include text; vector shapes; Flash; BMP, PICT, JPEG, and GIF file importing; sound management; and Shockwave Audio.

Xtras not included with the Shockwave player must be installed in a user's system before the movie plays. Use the Download If Needed option in the Movie Xtras dialog box to make the movie prompt the user to download the Xtra. Director downloads Xtras from the URL specified in the Xtrainfo.txt file in the Director application folder.

Xtras downloading from projectors requires use of Lingo. See **gotoNetMovie**.

Xtrainfo.txt includes URLs for all Macromedia Xtras included with Director, but you may need to manually edit Xtrainfo.txt to add the URL for third-party Xtras or Macromedia Xtras not included with Director. Xtrainfo.txt includes a description of how to enter this information. Xtra developers may also provide installation programs or other means of modifying Xtrainfo.txt automatically.

If a user chooses to download an Xtra, Director retrieves the Xtra from the URL specified in Xtrainfo.txt using the Verisign download security system. Verisign is a standard means of downloading software from secure sources.

You can also include Xtras in projector files. Select the Include in Projector option in the Movie Xtras dialog box for any Xtra you want to include.

Xtras usually required for the movie to play back correctly include the following:

- Xtras that create cast members (text, Flash, vector shapes, QuickTime, and so on)
- Shockwave Audio Xtras (if the movie uses files in the SWA format)
- Transition Xtras (if the movie uses third-party transitions)
- Import Xtras, if the movie uses nonstandard types of linked external cast members
- Network Xtras required for a movie to access the Internet
- Lingo Xtras (if the movie uses any special Lingo that requires Xtras)

Managing Xtras for distributed movies

You can manage Xtras for your movie using the Modify menu.

To manage Xtras for the current movie:

1. Choose Modify > Movie > Xtras.
2. To add or remove Xtras, do any of the following:

- To add the Xtras required to connect a projector to the Internet, click Add Network.
- To restore the list of default Xtras, click Add Defaults.
- To manually add an Xtra to the list (which you would do, for example, if you've used Lingo code that requires Xtras), click Add and choose from the list of Xtras installed in your system.
- To delete an Xtra from the list, highlight the Xtra and click Remove.

3. To change settings for Xtras in the list, select an Xtra and select either of the following options:

- Include in Projector makes Director include the selected Xtra in any projector that includes the current movie.
- Download If Needed makes the movie prompt the user to download a required Xtra if it is not installed in the user's system. The Xtra is downloaded from the location specified in the Xtrainfo.txt file and permanently installed in the user's system.

4. To get information about a selected Xtra, click Info.

   The information comes from an Internet source. Not all Xtras include information. Third-party Xtras often include some explanations and information about the developers.

Note: Another way to include Xtras with a movie is to create an Xtras folder containing all required Xtras in the same folder as a projector file. This allows you to see which Xtras are included without opening the movie. If you use this method, you cannot include Xtras in the projector file because the movie will fail to initialize.

About distribution formats

Before deciding how to distribute a movie, it helps to understand how Director plays movies. Director movies play either with the Shockwave player or through a projector player. The Shockwave player is a system component that plays movies in Web browsers and also outside browsers as stand-alone applications. A projector player can only play movies independent of a Web browser.

You can distribute movies as Shockwave movies (with the DCR extension), projectors, protected movies (DXRs), or Java applets. You should not distribute source movies (DIRs) unless you want your users to be able to change the movie in the Director authoring environment.

- A Shockwave movie is a compressed version of a movie's data and does not include a player. Shockwave movies are created primarily to distribute over the Internet for playback in a Web browser. Another reason to create a Shockwave movie is to compress it for distribution on a disk when the movie is contained in a projector. In addition to compressing the data, saving a movie in the Shockwave format removes all information necessary to edit the movie.
- A projector is a movie intended for play outside of a Web browser. A projector can include a player (called the Standard player), Xtras, multiple casts, and linked media in a single file. A projector can also include several different movie files. Configured in this way, a projector can be a completely stand-alone application.

  You can use the Shockwave player projector option to make a much smaller projector. A Shockwave projector uses an installed Shockwave player on the user's system to play a movie instead of including the player code in the projector itself. If no Shockwave player is installed on the user's system, the user must download a copy. A Shockwave projector is excellent for distributing movies on the Internet that you don't want to play in a Web browser.

  You can also reduce the file size of a projector by turning on projector options that compress the movie data, the player code, or both. In Windows, compressing the player code reduces the minimum projector size from approximately 2.1 MB to 1.1 MB for a projector, and to about 60K for a Shockwave projector.

  On the Macintosh, compressing the player code reduces the minimum projector size from approximately 2.5 MB to 1.2 MB for a projector, and to approximately 12K for a Shockwave projector.

- Protected movies (DXRs) are uncompressed movies that users can't open for editing. These can be useful when you want to distribute uncompressed movies on a disk, but you don't want users to edit the source file. Protected movies may play faster than Shockwave movies from a disk because they do not need to be decompressed. These movies are preferable if disk space isn't limited. Like Shockwave movies, protected movies do not include the information necessary to edit the movie or the software that plays the movie. They can be played only by a projector, a movie in a window, or the Shockwave player.
- A Java applet created by Director is a movie converted to Java. Java applets do not require the Shockwave player and provide an alternative for playing simple movies at Web sites where plug-ins are not allowed. Not all Director features are available when saving as Java; you have a number of authoring issues to consider when converting a movie to Java. For a complete description of Java authoring issues, see **Save as Java** in the Director Support Center Web site. You cannot include Java applets in a projector or play them as a movie in a window.

Note: To edit a movie packaged for distribution, you must edit the source file (DIR) and create a new movie in one of the distribution formats. Always save your source files.

Using linked media on the Internet

When you distribute a movie on the Internet for playback in a Web browser, the linked media must be at the specified URL when the movie plays. Otherwise, the user will receive an error message when the movie plays.

Distributing movies on a disk

Whenever a movie plays from a disk, it accesses all external linked files the same way that it did in the authoring environment. All linked media—bitmaps, sounds, digital videos, and so on—must be in the same relative location as they were when you created the movie. To make sure you don't forget any linked media when you distribute a movie on a disk, place linked files in the same folder as the projector or in a folder inside the Projector folder.

If your movie includes Xtras, you must include the Xtras in the projector. If a movie distributed on a disk connects to the Internet in any way, be sure to click the Add Network button in the Movie Xtras dialog box.

Distributing movies on a local network

If you plan to place a movie on a local area network (LAN), all files must be set to read-only, and users must have read/write access to their system folders. Otherwise, the requirements are the same as for normal disk-based distribution.

Creating Shockwave movies

You save your work as a Shockwave (DCR) movie to prepare it for playback in a Shockwave-enabled Web browser, or to make disk-based movies smaller. Using a Shockwave movie also prevents your users from editing the movie if they own Director.

If the Shockwave movie you're creating will be distributed on the Internet and requires any Xtras, make sure the Xtras are listed in the Movie Xtras dialog box and that Download If Needed is selected for each required Xtra. See **Managing Xtras for distributed movies**.

Note: Use Update Movies to convert several movies at once to the Shockwave format. For more information, see **Processing movies with Update Movies**.

Using Publish default settings

To create a Shockwave movie, you use the Publish command. The default settings create a DCR file and an HTML file with all of the tags necessary to display your DCR movie.

If you use the default Publish settings, Director will do the following:

- Create a DCR and HTML file in the same directory as your Director (DIR) movie.

Note: Director creates a CCT file for an external cast and, by default, saves the CCT file in the same folder as the DCR file. To specify a different file location, hold Alt (Windows) or Option (Macintosh) when you choose File > Publish. Continue to hold the key for access to dialog boxes that let you specify new paths for both your DCR and CCT files.

- Give both your DCR and HTML files the same name as your DIR file, with the appropriate extensions (e.g., MyMovie.dcr and MyMovie.html).
- Set the DCR movie's width and height to match the dimensions of the DIR movie.
- Configure the DCR movie and HTML file so that if your users resize their browsers, the DCR movie remains the same size as the original DIR movie.
- Use the same background color while loading your DCR movie as your movie Stage color.
- Compress bitmap images and sound using JPEG compression. Note that if you've compressed images for individual cast members, those settings will override compression set at the movie level in Publish settings.

If you change the default Publish settings, Director saves those changes when you save your movie. For more on changing Publish settings, see **Changing Publish settings**.

To create a Shockwave movie:

1. Save your movie.
2. Choose File > Publish.

   Director creates a Shockwave version of your movie, and an HTML file if you selected a template, based on your Publish settings. Your default browser launches with the HTML page you just created.

Note: Your default browser is specified in your Network Preferences dialog box. To change your default browser, choose File > Preferences > Network.

Changing Publish settings

You can change Publish settings by using the Publish Settings dialog box.

To change Publish settings:

Choose File > Publish Settings.

The Publish Settings dialog box appears with some or all of the following tabs, depending on the HTML template you select. The Image tab, for example, appears only if you select the Shockwave with Image HTML template.

- Formats
- General
- Shockwave
- Compression
- Shockwave Save
- Image

To use the Formats tab:

- To select an HTML template, or to create a Shockwave file without an HTML template, you use the HTML Template pop-up menu.

  To create a Shockwave file without an HTML file, select no HTML Template.

  To use `OBJECT` and `EMBED` parameters in the HTML file to display the Shockwave file, use the Shockwave Default template.

  When you select the Detect Shockwave template, JavaScript and VB Script determine if the correct version of the Shockwave plug-in or ActiveX is on your user's computer. If not, a message tells your user to update Shockwave.

  To expand the Shockwave file to fill the entire browser window, select Fill Browser Window.

  To display a Java applet created using the Save as Java functionality, select Java.

  To play a loader movie while the Shockwave file downloads, select Loader Movie.

  To display a game with a progress bar while the Shockwave file loads, select Loader Game.

  To display a progress bar and image while the Shockwave file downloads, select Progress Bar with Image.

  If you select the Shockwave with Image template, the template automatically detects the Shockwave player or Active X control on your user's browser and uses it to display your movie. If Shockwave is not found and the user's browser is Internet Explorer on Windows 95 or NT, the browser automatically installs the Active X control. In all other cases, the image that you specify on the Publish Settings Image tab is displayed. The Image tab is available only when you select the Shockwave with Image template.

  To display a progress bar while the Shockwave file downloads, select Simple Progress Bar.

  To center the Shockwave movie in the browser, select Center Shockwave.

- The HTML File and Shockwave File fields indicate where Director will save your HTML and DCR files, respectively. To specify another path, edit the path in the field, or click the field's Browse button and choose a new path.
- If you select the Java template, the Formats tab includes a Java Class File field. This field points

to the Director movie that you saved as Java.

- To launch your movie in a browser automatically when you execute the Publish command, select Output: View in Browser.

To use the General tab:

- To make the DCR movie match the dimensions of your DIR movie, select Match Movie in the Dimensions field.
- If you use the default Match Movie setting in the Dimensions field, values in the `OBJECT` and `EMBED` tags in the HTML file are set to the exact dimension of your movie. To change the dimensions, select either Pixels or Percent of Browser Window, and type the new dimensions in the Width and Height fields. Your movie will resize to fit the new rectangle only if you have not selected No Stretching in the Stretch Style pop-up menu on the Shockwave tab.
- To change the background color of your HTML file, either click the Page Background color box and select a color, or enter a value in the hexadecimal field.

  The Page Background setting is different from Background Color, which you specify on the Shockwave tab. Background Color lets you determine the color that appears, while the DCR is downloading, in the rectangle where your DCR movie will play. Another background color option, Stage Fill Color, which you set on the Movie tab of the Property Inspector, defines the color of the Stage.

To use the Shockwave tab:

Select Shockwave playback options to enable the following features for your user:

- Volume Control lets users adjust the volume of the movie's soundtrack.
- Transport Control provides controls for rewinding, stopping, and stepping through the movie in Shockmachine.
- Zooming determines if stretchable Shockwave is allowed. You can disable zooming with Lingo by setting the `allowZooming` property. For more information, see the Lingo Dictionary.
- Save Local determines if the movie can be saved to Shockmachine.
- Display Progress Bar and Display Logo determine if the progress bar and logo, respectively, appear while the movie loads in the browser.

For more information, see **Setting Shockwave playback options**.

- To specify how your movie behaves when the width and height values in the HTML file are a different size than the movie, select from the Stretch Style pop-up menu.

  If the HTML height and width values have been set to a percentage on the General tab, the movie will resize as the browser window resizes. The way in which the movie resizes varies according to the stretch style selected.

  No Stretching, the default, means the movie plays at its original size.

  Preserve Proportions keeps the same aspect ratio of your original Director movie no matter what size the user makes the browser. The movie stretches to fill the height and width values specified in the HTML file or those determined by the percentage and size of your browser window.

  Stretch to Fill stretches the movie to fill the height and width values in the HTML file. If the aspect ratio of the movie changes, sprites on the Stage could appear distorted.

  Expand Stage Size lets users resize the Stage, but the sprites on the Stage remain the same size. The setting expands the Stage size to the size of the height and width values in the HTML file.

Note: If, on the General tab, you're using the default Match Movie Dimension setting, the stretch style settings will not affect the way your movie responds to browser resizing.

- You can use the Stretch Position Horizontal Align and Vertical Align selections to determine how your movie will line up within the `OBJECT` or `EMBED` values in the HTML file.

  For more information about using stretch styles, see the procedure under **Setting movie options for browser resizing**.

- To change the background color that displays while your DCR file is downloading, either click the Background Color box and select a color, or enter a value in the hexadecimal field.
- If you use Lingo to call JavaScript in your movie, you must select Movie Uses Browser Scripting. This creates a flag for Netscape to start Java when the movie loads.

To use the Compression tab:

The Compression tab sets bitmap compression for all cast members in a movie. (You can set the compression quality for individual bitmap cast members on the Bitmap tab of the Property Inspector.)

- To apply compression techniques used by Director in versions 4 through 7, select Standard. This setting is suitable for graphics with few colors.
- To use JPEG compression, click JPEG and specify the image quality setting by moving the slider to a value between 0 and 100 percent. The higher the percentage, the less the image is compressed.
- To compress the sound in your movie, select Compression Enabled and select the level of compression from the kBits/second pop-up menu. For more information about sound compression, see **Compressing internal sounds with Shockwave Audio**.
- You can convert stereo sounds to monaural by selecting Convert Stereo to Mono.
- If you entered text in the Comments field of the Property Inspector for your cast members, you can include those comments in your DCR file by selecting Include Cast Member Comments. You can then use Lingo to access the comments in the DCR file.

To use the Shockwave Save tab:

To save your Shockwave file for playback with Macromedia Shockmachine, you use fields on the Shockwave Save tab:

- To display the standard Shockwave context menu when your user right clicks (Windows) or Control-Clicks (Macintosh) your DCR file playing in the browser, select Context Menu.
- To suggest a Shockmachine category, such as "games," enter a description in the Suggested Category field.
- You can enter a title for the user interface in the Shockwave Title field. The title can include spaces.
- The Send URL field lets you specify and override the URL that the Shockmachine detects. You can use frame-based targets, encode form data, or include other information in the URL. If the URL is not specified, the movie uses the URL for the page that contains the movie, if the URL is available.
- You can use the Icon File field to specify the path to a BMP file.
- The Package File field lets you enter a fully qualified or relative URL to a package text file. This text file, in XML, provides a list of URLs with support files to download for a complete save of the current project.
- For content with packages, you can use the Total File Size field to specify the size, in bytes, of all content that needs to download for the movie to save successfully.

For more information about the Shockwave Save fields, and for detailed instructions on developing for Shockmachine, visit **shockwave.com** for the Shock Remote and Shock Machine Development Guide.

To use the Image tab:

If you selected the Shockwave with Image HTML template on the Formats tab, the Image tab appears on the Publish Settings dialog box.

You can specify the image that should appear if the user doesn't have Shockwave or the ActiveX control.

- In the Poster Frame field, enter the frame number from your movie's Score that you want to appear as a JPEG image for users who are unable to view your movie.
- To specify compression for the image, move the Quality slider to the desired compression setting. The higher the percentage, the less the image is compressed.
- To specify that the image download as a progressive JPEG, select Progressive. The JPEG will then display at low resolution and increase in quality as it continues to download. Making a JPEG progressive also reduces its file size.

Setting movie options for browser resizing

If users view your movie in browsers, chances are they will resize their browsers. How your movie behaves when the browser size changes depends on what you select in the Publish Settings dialog box.

To set movie options for browser resizing:

1. Choose File > Publish Settings.
2. On the General tab of the Publish Settings dialog box, select from the Dimensions pop-up menu. Note that when you make a selection, the width and height values default to the movie size.

- To create an HTML file with parameters that match the height and width of the movie, select Match Movie.
- To specify height and width values in the HTML file in pixels, select Pixels.
- You can select Percentage of Browser Window, and specify a percentage in the Width and Height fields. (To make browser resizing affect the size of the DCR movie, you must specify percentages and select either Preserve Proportions, Stretch to Fill, or Expand Stage Size on the Shockwave tab of the Publish Settings dialog box.)

3. On the Shockwave tab, select an option from the Stretch Style pop-up menu.

- To specify that your movie not resize at all, select No Stretching.
- To maintain the same aspect ratio of your original Director movie no matter what size the user makes the browser, select Preserve Proportions. The movie will fit within the width and height parameters, as much as possible, while preserving the movie's aspect ratio. The movie aligns within the window based on the align tags that you specify in step 4.
- To change the size of the movie to fit the size of the browser, select Stretch to Fill. Any browser resizing stretches the movie to fill the width and height parameters. Note, however, that if the aspect ratio of the movie changes, sprites on the Stage could appear distorted. If you select Stretch to Fill, Director ignores the align tags that you specify in step 4.

To let users resize the Stage without resizing the sprites, select Expand Stage Size. The movie is aligned within the browser based on the align tags that you specify in step 4.

4. To specify align tags for your movie, use the Horizontal Align and Vertical Align pop-up menus. You can select left, center, or right horizontal alignment, and top, center, or bottom vertical alignment.

About projectors

To create projectors for any version of Windows, you must use the Windows version of Director; likewise, you can create Macintosh projectors only with the Macintosh version of Director.

Projectors require certain Xtras to use text, use Flash movies, connect to the Internet, and use certain other features. Director includes the most common required Xtras by default. You can include or exclude Xtras for each movie using the Include in Projector option in the Movie Xtras dialog box. You can also add Xtras to a projector manually the same way you select movie files. (See **Creating projectors**.)

In addition to the standard projector, you can create a fast-start projector, which typically launches faster. A fast-start projector doesn't include Xtras inside the projector itself, so there's nothing to unpack.

Creating projectors

When creating a projector, place the starting movie at the top of the list of files in the Create Projector dialog box. If the Play Every Movie option is selected in the Projector Options dialog box, movies play in the order they appear in the list. If this option is off, only the first movie plays. If your movie contains Lingo that switches between movies, the order of the other movies may not be important.

You can include only Director 8 movies in projectors. Use the Update Movies command to convert older movies to the latest version of Director. For more information, see **Processing movies with Update Movies**.

To create a standard projector:

1. Choose File > Create Projector.
2. Double-click the movies and external casts to include in the projector. Click Add All to include all the movies in the open folder.

   Director transfers the name of the movies or casts to the file list.
3. Use the Move Up and Move Down buttons to arrange the movies in the proper order.
4. Click Options.

   Director retains the options settings once you define them; you don't have to set them every time.
5. To control how movies interact with the user's system, select Playback options:

- Play Every Movie specifies that the projector play all movies in the play list. Otherwise, the projector plays only the first movie in the play list (unless other movies are called by Lingo from the first movie). In a projector with Play Every Movie selected, pressing Control + period (Windows) or Command + period (Macintosh) will branch to the next movie, and Control + Q (Windows) or Command + Q (Macintosh) will quit.
- Animate in Background allows the movie to continue playing if a user switches to another application. This is useful if you want the movie to continue running in the background when its window is not active. If this option is not selected, the movie pauses when the user switches to another application and resumes when the user switches back.
- Reset Monitor to Match Movie's Color Depth (Macintosh only) automatically changes the color depth of your monitor to the color depth of each movie in the projector play list. For example, if you are working on a color monitor set to 256 colors and a movie in the play list was created in thousands of colors, the monitor will automatically switch to thousands of colors.

6. To determine how the projector appears on the screen, make an Options selection:

- Full Screen displays the movie in the entire screen, placing the menu bar (if there is one) at the top of the screen and hiding all of the desktop. If there's a menu, it overlays the top of the Stage.
- In a Window displays the movie in a normal window, without taking over the screen. The window cannot be resized.
- Show Title Bar is available only if In a Window is selected. If this option is selected, the window where the movie appears has a title bar. The window can be moved only if it has a title bar.

7. To specify how the Stage size of multiple movies in the projector can be adjusted, choose a Stage Size option:

- Use Movie Settings uses the Stage size of the new movie or matches the size of the current movie.
- Match First Movie repositions and resizes the movie based on the first movie in the projector.
- Center centers the Stage on the screen, which is useful if the Stage size is smaller than the

screen size. Otherwise, the movie plays using its original Stage position. In Windows, projectors are always centered.

8. To compress the projector's movie data in the Shockwave format, choose Compress (Shockwave Format).

   This makes the projector smaller, but it may increase the load time as the movies are decompressed.

9. To determine how the player code is included in the projector, choose an option for Player.

   For more about these options, see **About distribution formats**.

- Standard includes the uncompressed player code in the projector file. This option starts the movie faster than other options but creates the largest projector file.
- Compressed includes a compressed version of the player code in the projector file. This substantially reduces the projector file size, but decompressing the player code adds a few seconds to the startup time of a movie.
- Shockwave makes the projector use the Shockwave player installed in a user's system instead of including the player code in the projector file. If the Shockwave player is not available when the movie runs, the movie prompts the user to download it.

10. (Macintosh only) To make Director use available system memory when its own partition is full, choose Use System Temporary Memory.

11. Once all projector options are set, click OK.

12. Click Create in the Projector dialog box and then enter a name and location for the projector.

    To avoid problems with linked media, create the new projector in its final folder location and do not move it to a different folder.

    Director turns the movies, casts, and included Xtras into a single projector.

To create a fast-start projector:

1. Create a new folder on your computer desktop.

   It does not matter what you name the folder.

2. In Director, choose Modify > Movie: Xtras.

   The Xtras dialog box appears.

3. Select the name of each Xtra and deselect Include in Projector for each, then click OK.

4. Choose File > Save and Compact.

   If you are adding multiple movies to the package, repeat steps 2 through 4 for each of the movies.

5. Choose File > Create Projector.

6. In the Create Projector dialog box, select the movies to include in the projector and click Add.

7. Click Options and do one of the following:

- Select Shockwave (Windows) and click OK.
- Select Standard (Macintosh) and click OK.

8. In the Create Projector dialog box, click Create.

9 In the dialog box that appears, type a name for the projector. If necessary, use the pop-up menu to browse to the desktop folder you created in step 1, and then click Save.

10 Exit Director and return to your computer desktop.

11 Open the folder you created in step 1. Create a subfolder within this folder and name it Xtras.

12 In your Director application folder, copy the Xtras required to play your movie into the Xtras folder you just created.

You must also include external movies, external casts, and linked media with your projector. If the external files are in the folder that contains the projector, the projector can automatically link to the files.

13 (Windows only) In your Director application folder, copy the files dirapi.dll, iml32.dll, proj.dll, and msvcrt.dll into the Xtras folder.

14 Launch your projector to see it open quickly and play your movies.

For more information about creating a fast-start projector, visit the Director Support Center.

Processing movies with Update Movies

Use the Update Movies command on the Xtras menu to do the following:

- Update movies and casts from older version of Director to the latest file format.
- Compress movies for faster downloading from the Internet.
- Remove redundant and fragmented data in movie and cast files. The Save and Compact and Save As commands do this as well.
- Prevent users from opening movie and cast files.
- Batch-process movie and cast files in large projects.

When beginning a project, use Update Movies to convert Director 6 and 7 files to the latest file format.

At the end of a project, use Update Movies to compress all your movies and casts at once.

To update and compress movies and casts:

1. Choose Xtras > Update Movies.

   The Update Movies dialog box appears.

2. Select one of the Action options:

- Update converts movies from Director 5 or later versions to the latest file format. As it updates movies, Director consolidates and removes fragmented data, just as when you use Save As. (To update movies from older versions, you must first convert them to the Director 5 file format.)
- Protect removes all the data required to edit the movie, but it does not compress the movies further. It adds the .dxr extension to movies, and .cxt to casts. Protect also flags the movie so it can't be opened in the authoring environment.
- Convert to Shockwave Movie(s) rewrites movies and casts in the compressed Shockwave file format and adds the .dcr extension to movies and .cct to casts. This options also prevents users from opening the movie or cast and making changes. Once a movie is compressed, there is no way to it to recover an editable file, so be sure to keep the original movie.

3. Select one of the Original Files options:

- Back Up into Folder specifies that the original files go in a selected folder. Click Browse to select the folder for the original files. To avoid overwriting old backups, you should choose a new folder each time you run Update Movies.
- Delete specifies that the newly updated files overwrite the original files. Be very careful when using this option. Once a file is protected or compressed, you cannot open it again in Director.

4. Click OK.

   A dialog box appears from which you select the files to change.

5. Select the movies and casts you want to change and click Add.

- Click Add All to add all the movies in the current folder. The items you select appear in the file list at the bottom of the dialog box. You can update movies in different folders at the same time.
- Select Add All Includes Folders before you click the Add All button to include any movies or casts inside folders appearing in the upper list. This option is useful for updating large projects with several levels of folders.

6. Click Update.

   Director saves new versions of the selected movies with the same names and locations as the original movies. This ensures that all links and references to other files continue to work properly. Director copies the original movies to the folder you specified, re-creating their original folder structure. If you didn't specify a folder for the original movies, Director prompts you to select one.

Director adds the .dcr extension to Shockwave movies and the .cct extension to external casts in the Shockwave format. Protected movies have the .dxr extension, and protected casts have the .cxt extension.

Exporting digital video and frame-by-frame bitmaps

You can export all or part of a movie as a digital video. You can use this digital video in other applications or import it back into Director. Any interactivity in the movie is lost when it is exported as a digital video. You can also export a movie or a part of a movie as a series of bitmaps: BMP in Windows, and PICS, PICT, or Scrapbook on the Macintosh.

You can export QuickTime digital video from either the Windows or the Macintosh version of Director. QuickTime must be installed on the system to export as QuickTime (version 4 or later required for Windows, version 3 or later required for Macintosh). You can export the Video for Windows (AVI) format only using the Windows version of Director. When you export to AVI, all sounds are lost.

When Director exports animation as a video or bitmaps, it takes snapshots of the Stage moment by moment and turns each snapshot into a frame in the video or a bitmap file. Sprites animated solely by Lingo are not exported.

When Director exports video or bitmaps, it always uses the entire Stage.

To export to digital video or bitmaps:

1. Choose File > Export.

   The Export dialog box appears.

2. Select the range of frames you want from the Export options at the top of the dialog box:

- Current Frame exports the current frame on the Stage. This is the default.
- Selected Frames exports the selected frames in the Score.
- All Frames exports all frames.
- Frame Range exports only the range of frames that begin and end with the frame numbers you enter in the Begin and End boxes.

3. If you choose Selected Frames, All Frames, or Frame Range as the Export option, select one of the following options.

   These options do not work with digital video.

- Every Frame exports all frames in the selected range.
- One in Every _ Frames exports only the frames at the interval you specify in the box.
- Frames with Markers exports frames with markers set in the Score window.
- Frames with Artwork Changes in Channel exports frames only when a cast member changes in the channel you specify in the box.

4. From the Format pop-up menu at the bottom of the dialog box, choose Video for Windows (AVI), BMP (Windows), QuickTime Movie, PICT (Macintosh), Scrapbook (Macintosh), or PICS (Macintosh).

   BMP is the standard format for a Windows bitmap series. PICT, Scrapbook, and PICS are all Macintosh bitmap file formats.

5. If you are exporting in PICS format, click Use Frame Differencing to create smaller files.

   This option is dimmed unless you choose PICS from the Format pop-up menu.

6. If you are exporting video, click the Options button.

   The Video for Windows or QuickTime Options dialog box appears.

7. Select the options you want to use and then click OK.

For AVI movies, enter a number of frames per second for Frame Rate.

For information about the QuickTime options, see **Setting QuickTime export options**.

The Export dialog box reappears when you click OK.

8 Click Export.

A dialog box appears, prompting you to save the movie.

9 Name the file and then click Save.

When you click Export, a dialog box appears allowing you to name the file. If you are saving in video, PICS, or Scrapbook format, only one file will be created. If you are saving in BMP or PICT format, Director automatically creates one file for each frame, attaching the corresponding frame number to each file. For example, if the name of the exported file is Myfile, frame 1 will be exported to a file named Myfile0001.

Setting QuickTime export options

You use the QuickTime Options dialog box to specify options for exporting a movie as a QuickTime digital video. This dialog box appears when you click the Options button in the Export dialog box and QuickTime is the specified format.

To set QuickTime export options:

1. Choose File > Export.
2. Choose QuickTime Movie from the Format pop-up menu.
3. Click Options.
4. To set the speed the video will play, choose a Frame Rate option:

- Tempo Settings exports the settings in the tempo channel to the QuickTime movie. This setting lets you create a QuickTime movie at any tempo, even if Director is not capable of playing the movie at that tempo in real time.

  The size of an exported QuickTime movie is influenced by the tempo settings, transitions, and palette transitions in the Director movie. Fast tempos, certain transitions, and palette transitions all increase the size of the QuickTime movie. The tempo settings determine the number of QuickTime frames per second and the number of frames per transition. The faster the tempo, the more frames per second.

  A movie that would work well with Tempo Settings as the Frame Rate option is one in which the tempos have been carefully timed. For instance, some frames could be set to a tempo of 10 frames per second, and their QuickTime frame durations would be exactly one-tenth of a second. Other frames later in the movie could be set to a tempo of 1 frame per second; when the movie is exported, these slower frames would each last precisely 1 second in the QuickTime movie.

- Real Time lets you export a QuickTime movie that matches the performance of the Director movie as it plays on your system. (You should always play the entire movie with Lingo disabled before using this feature.)

  When you export a movie with Real Time selected, each Director frame becomes a QuickTime frame. Each frame in the QuickTime movie will match the duration of the same frame in the Director movie.

  Director will generate as many frames as required to duplicate each transition, up to 30 frames per second. To increase the number of frames created for any transition, reduce the smoothness of the transition.

  This option causes Director to use the actual durations that were stored the last time you played the entire movie, regardless of the actual tempo settings of the movie.

5. To reduce the file size of a QuickTime movie at the expense of quality, choose an option from the Compressor pop-up menu. Different options appear on the Compressor pop-up menu depending on the video hardware and software available in your system. Consult your QuickTime documentation.

- Animation compression is for simple animations.
- Cinepak compresses 16-bit and 24-bit video for playback from CD-ROMs.
- Component Video is usually used when capturing from a live video feed.
- Graphics compression is for exporting single frames of computer graphics.
- None exports with no compression.
- Photo-JPEG compression is good for scanned or digitized continuous-tone still images.

- Video compression is for exporting video clips.

6 To determine the compression quality and resulting file size when using the chosen compressor, use the Quality slider. A higher-quality setting preserves the appearance of the images and motion but increases the size of the file. A lower-quality setting results in poorer image quality but decreases the size of the file.

7 To determine the color depth (the number of colors) of your artwork, choose a setting from the Color Depth pop-up menu. The compression method you choose determines the color depth options available to you in this pop-up menu.

8 To determine the method by which the exported QuickTime movie is resized, choose values for Scale. You can choose a percentage from the Scale pop-up menu, or you can type pixel dimensions in the fields. By entering the number of pixels, you can stretch a movie so that it plays in a rectangle that does not adhere to the original aspect ratio.

9 To choose which soundtracks are exported with your movie, choose Channel 1 or Channel 2. A checked box indicates that the associated sound channel in the score is exported with your QuickTime file.

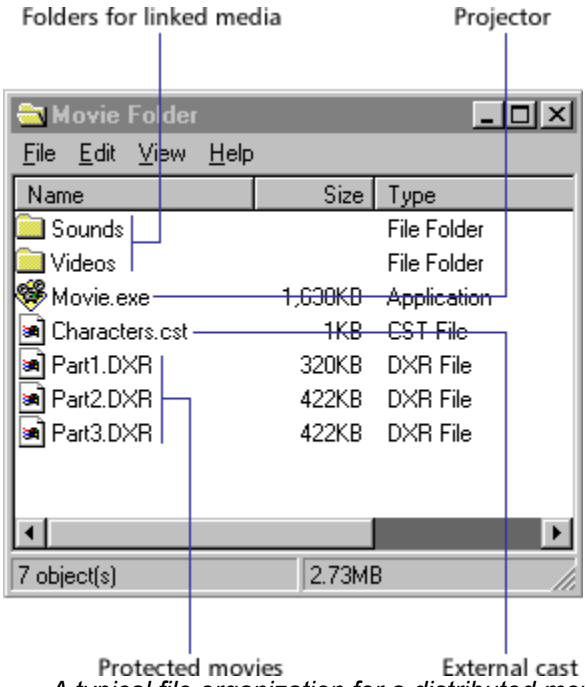
External sounds (sounds you imported as linked cast members) are not exported when you export a digital video. To include sound when you export a digital video movie, you must import the sounds as cast members instead of linking to them.

Looped sounds don't loop in a movie that you have exported as a digital video. To loop a sound in a movie that you plan to export as a digital video, you must trigger the sound by alternating it between the two sound channels.

About organizing movie files

In most cases, you should divide a large production into a series of smaller movies. You can combine as many movies as you want in a projector, but larger files take longer to save and are cumbersome to work with. Also, movies are easier to edit if they are organized in discrete sections.

The best way to organize a large production is to create a small projector file that launches the movie and then branches to Shockwave or protected movies. This saves you the trouble of re-creating the projector every time you change one part of a movie.



A typical file organization for a distributed movie

This approach also makes sense for movies on the Internet, but for different reasons. If the first movie is small, users don't have to wait as long for something to happen. Branching to a series of smaller movies also enables users to avoid downloading time for parts of the movie they do not use.

The size of your movie may be less of an issue if you use streaming Shockwave. For more information, see **Setting movie playback options**.

Lingo Dictionary Overview

This dictionary describes the syntax and use of Lingo elements in Director 8. Nonalphabetical operators are presented first, followed by all other operators in alphabetical order. For information about Lingo used for the Multiuser server Xtra, see **Using the Shockwave Mulituser Server** in the Director Support Center. For information about Lingo used for XML parsing, see **Using the XML Parser** in the Director Support Center.

To use examples in a script, copy the example text and paste it in the Script window.

Lingo by alphabet

Lingo by feature

(symbol)

Syntax `#symbolName`

Description Symbol operator; defines a symbol, a self-contained unit that can be used to represent a condition or flag. The value *`symbolName`* begins with an alphabetical character and may be followed by any number of alphabetical or numerical characters.

A symbol can:

- Assign a value to a variable
- Compare strings, integers, rectangles, and points
- Pass a parameter to a handler or method
- Return a value from a handler or method

A symbol takes up less space than a string and can be manipulated, but unlike a string it does not consist of individual characters. You can convert a symbol to a string for display purposes by using the `string` function.

The following are some important points about symbol syntax:

- Symbols are not case sensitive.
- Symbols can't start with a number.
- Spaces may not be used, but you can use underscore characters to simulate them.
- Symbols use the 128 ASCII characters, and letters with diacritical or accent marks are treated as their base letter.
- Periods may not be used in symbols.

All symbols, global variables, and names of parameters passed to global variables are stored in a common lookup table.

Example This statement sets the state variable to the symbol `#Playing`:

```
state = #Playing
```

See also **ilk()**,**string()**,**symbol()**,**symbolP()**

. (dot operator)

Syntax
```
objectReference.objectProperty
textExpression.objectProperty
object.commandOrFunction()
```

Description
Operator; used to test or set properties of objects, or to issue a command or execute a function of the object. The object may be a cast member, a sprite, a property list, a child object of a parent script, or a behavior.

Example
This statement displays the current member contained by the sprite in channel 10:

```
put sprite(10).member
```

Example
To use the alternate syntax and call a function, you can use the form:

```
myColorObject = color(#rgb, 124, 22, 233)
put myColorObject.ilk()
-- #color
```

- (minus)

Syntax `(Negation): -expression`

Description Math operator; reverses the sign of the value of *expression*.

This is an arithmetic operator with a precedence level of 5.

Syntax `(Subtraction): expression1 - expression2`

Description Math operator; performs an arithmetic subtraction on two numerical expressions, subtracting *expression2* from *expression1*. When both expressions are integers, the difference is an integer. When either or both expressions are floating-point numbers, the difference is a floating-point number.

This is an arithmetic operator with a precedence level of 3.

Example (Negation): This statement reverses the sign of the expression 2 + 3:

```
put -(2 + 3)
```

The result is -5.

Example (Subtraction): This statement subtracts the integer 2 from the integer 5 and displays the result in the Message window:

```
put 5 - 2
```

The result is 3, which is an integer.

Example (Subtraction): This statement subtracts the floating-point number 1.5 from the floating-point number 3.25 and displays the result in the Message window:

```
put 3.25 - 1.5
```

The result is 1.75, which is a floating-point number.

-- (comment)

Syntax

```
-- comment
```

Description

Comment delimiter; indicates the beginning of a script comment. On any line, anything that appears between the comment delimiter (double hyphen) and the end-of-line return character is interpreted as a comment rather than a Lingo statement.

The Director player for Java accepts Lingo that uses this delimiter, but comments do not appear in the final Java code.

Example

This handler uses a double hyphen to make the second, fourth, and sixth lines comments:

```
on resetColors
  -- This handler resets the sprite's colors.
  sprite(1).forecolor = 35
  -- bright red
  sprite(1).backcolor = 36
  -- light blue
end
```

& (concatenator)

Syntax `expression1 & expression2`

Description String operator; performs a string concatenation of two expressions. If either *`expression1`* or *`expression2`* is a number, it is first converted to a string. The resulting expression is a string.

This is a string operator with a precedence level of 2.

Be aware that Lingo allows you to use some commands and functions that take only one argument without parentheses surrounding the argument. When an argument phrase includes an operator, Lingo interprets only the first argument as part of the function, which may confuse Lingo.

For example, the `open window` command allows one argument that specifies which window to open. If you use the `&` operator to define a pathname and file name, Director interprets only the string before the `&` operator as the file name. For example, Lingo interprets the statement `open window the applicationPath & "theMovie"` as `(open window the applicationPath) & ("theMovie")`. Avoid this problem by placing parentheses around the entire phrase that includes an operator, as follows:

```
open window (the applicationPath & "theMovie")
```

The parentheses clear up Lingo's confusion by changing the precedence by which Lingo deals with the operator, causing Lingo to treat the two parts of the argument as one complete argument.

Example This statement concatenates the strings "abra" and "cadabra" and displays the resulting string in the Message window:

```
put "abra" & "cadabra"
```

The result is the string "abracadabra".

Example This statement concatenates the strings "$" and the content of the `price` variable and then assigns the concatenated string to the Price field cast member:

```
member("Price").text = "$" & price
```

&& (concatenator)

Syntax `expression1 && expression2`

Description String operator; concatenates two expressions, inserting a space character between the original string expressions. If either *`expression1`* or *`expression2`* is a number, it is first converted to a string. The resulting expression is a string.

This is a string operator with a precedence level of 2.

Example This statement concatenates the strings “abra” and “cadabra” and inserts a space between the two:

```
put "abra" && "cadabra"
```

The result is the string “abra cadabra”.

Example This statement concatenates the strings “Today is” and today’s date in the long format and inserts a space between the two:

```
put "Today is" && the long date
```

If today’s date is Tuesday, March 15, 1999, the result is the string “Today is Tuesday, March 15, 1999”.

() (parentheses)

Syntax `(expression)`

Description Grouping operator; performs a grouping operation on an expression to control the order of execution of the operators in an expression. This operator overrides the automatic precedence order so that the expression within the parentheses is evaluated first. When parentheses are nested, the contents of the inner parentheses are evaluated before the contents of the outer ones.

This is a grouping operator with a precedence level of 5.

Be aware that Lingo allows you to use some commands and functions that take only one argument without parentheses surrounding the argument. When an argument phrase includes an operator, Lingo interprets only the first argument as part of the function, which may confuse Lingo.

For example, the `open window` command allows one argument that specifies which window to open. If you use the `&` operator to define a pathname and file name, Director interprets only the string before the `&` operator as the file name. For example, Lingo interprets the statement `open window the applicationPath & "theMovie"` as `(open window the applicationPath) & ("theMovie")`. Avoid this problem by placing parentheses around the entire phrase that includes an operator, as follows:

```
open window (the applicationPath & "theMovie")
```

Example These statements use the grouping operator to change the order in which operations occur (the result appears below each statement):

```
put (2 + 3) * (4 + 5)
-- 45
put 2 + (3 * (4 + 5))
-- 29
put 2 + 3 * 4 + 5
-- 19
```

* (multiplication)

Syntax `expression1 * expression2`

Description Math operator; performs an arithmetic multiplication on two numerical expressions. If both expressions are integers, the product is an integer. If either or both expressions are floating-point numbers, the product is a floating-point number.

This is an arithmetic operator with a precedence level of 4.

Example This statement multiplies the integers 2 and 3 and displays the result in the Message window:

```
put 2 * 3
```

The result is 6, which is an integer.

Example This statement multiplies the floating-point numbers 2.0 and 3.1414 and displays the result in the Message window:

```
put 2.0 * 3.1416
```

The result is 6.2832, which is a floating-point number.

+ (addition)

Syntax `expression1 + expression2`

Description Math operator; performs an arithmetic sum on two numerical expressions. If both expressions are integers, the sum is an integer. If either or both expressions are floating-point numbers, the sum is a floating-point number.

This is an arithmetic operator with a precedence level of 4.

Example This statement adds the integers 2 and 3 and then displays the result, 5, an integer, in the Message window:

```
put 2 + 3
```

Example This statement adds the floating-point numbers 2.5 and 3.25 and displays the result, 5.7500, a floating-point number, in the Message window:

```
put 2.5 + 3.25
```

/ (division)

Syntax `expression1 / expression2`

Description Math operator; performs an arithmetic division on two numerical expressions, dividing *`expression1`* by *`expression2`*. If both expressions are integers, the quotient is an integer. If either or both expressions are floating- point numbers, the quotient is a floating-point number.

This is an arithmetic operator with a precedence level of 4.

Example This statement divides the integer 22 by 7 and then displays the result in the Message window:

```
put 22 / 7
```

The result is 3. Because both numbers in the division are integers, Lingo rounds the answer down to the nearest integer.

Example This statement divides the floating-point number 22.0 by 7.0 and then displays the result in the Message window:

```
put 22.0 / 7.0
```

The result is 3.1429, which is a floating-point number.

< (less than)

Syntax `expression1 < expression2`

Description Comparison operator; compares two expressions and determines whether *`expression1`* is less than *`expression2`* (`TRUE`), or whether *`expression1`* is greater than or equal to *`expression2`* (`FALSE`).

This operator can compare strings, integers, floating-point numbers, rects, and points. Be aware that comparisons performed on rects or points are handled as if the terms were lists, with each element of the first list compared to the corresponding element of the second list.

This is a comparison operator with a precedence level of 1.

<= (less than or equal to)

Syntax `expression1 <= expression2`

Description Comparison operator; compares two expressions and determines whether *`expression1`* is less than or equal to *`expression2`* (`TRUE`), or whether *`expression1`* is greater than *`expression2`* (`FALSE`).

This operator can compare strings, integers, floating-point numbers, rects, and points. Be aware that comparisons performed on rects or points are handled as if the terms were lists, with each element of the first list compared to the corresponding element of the second list.

This is a comparison operator with a precedence level of 1.

<> (not equal)

Syntax `expression1 <> expression2`

Description Comparison operator; compares two expressions, symbols, or operators and determines whether *`expression1`* is not equal to *`expression2`* (`TRUE`), or whether *`expression1`* is equal to *`expression2`* (`FALSE`).

This operator can compare strings, integers, floating-point numbers, rects, and points. Be aware that comparisons performed on rects or points are handled as if the terms were lists, with each element of the first list compared to the corresponding element of the second list.

This is a comparison operator with a precedence level of 1.

= (equals)

Syntax `expression1 = expression2`

Description Comparison operator; compares two expressions, symbols, or objects and determines whether *`expression1`* is equal to *`expression2`* (`TRUE`), or whether *`expression1`* is not equal to *`expression2`* (`FALSE`).

This operator can compare strings, integers, floating-point numbers, rects, lists, and points.

Lists are compared based on the number of elements in the list. The list with more elements is considered larger than the than the list with fewer elements.

This is a comparison operator with a precedence level of 1.

> (greater than)

Syntax `expression1 > expression2`

Description Comparison operator; compares two expressions and determines whether *`expression1`* is greater than *`expression2`* (`TRUE`), or whether *`expression1`* is less than or equal to *`expression2`* (`FALSE`).

This operator can compare strings, integers, floating-point numbers, rects, and points. Be aware that comparisons performed on rects or points are handled as if the terms were lists, with each element of the first list compared to the corresponding element of the second list.

This is a comparison operator with a precedence level of 1.

>= (greater than or equal to)

Syntax `expression1 >= expression2`

Description Comparison operator; compares two expressions and determines whether *`expression1`* is greater than or equal to *`expression2`* (`TRUE`), or whether *`expression1`* is less than *`expression2`* (`FALSE`).

This operator can compare strings, integers, floating-point numbers, rects, and points. Be aware that comparisons performed on rectangles or points are handled as if the terms were lists, with each element of the first list compared to the corresponding element of the second list.

This is a comparison operator with a precedence level of 1.

[ ] (bracket access)

Syntax
```
textExpression[chunkNumberBeingAddressed]
textExpression[firstChunk..lastChunk]
```

Description Operator; allows a chunk expression to be addressed by number. Useful for finding the *n*th chunk in the expression. The chunk can be a word, line, character, paragraph, or other Text cast member chunk.

Example This outputs the first word of the third line in the text cast member First Names:

```
put member("First Names").text.line[3].word[1]
```

[ ] (list)

Syntax `[entry1, entry2, entry3, ...]`

Description List operator; specifies that the entries within the brackets are one of four types of lists:

- Unsorted linear lists
- Sorted linear lists
- Unsorted property lists
- Sorted property lists

  Each entry in a linear list is a single value that has no other property associated with it. Each entry in a property list consists of a property and a value. The property appears before the value and is separated from the value by a colon. You cannot store a property in a linear list. When using strings as entries in a list, enclose the string in quotation marks.

  For example, [6, 3, 8] is a linear list. The numbers have no properties associated with them. However, [#gears:6, #balls:3, #ramps:8] is a property list. Each number has a property—in this case, a type of machinery— associated with it. This property list could be useful for tracking the number of each type of machinery currently on the Stage in a mechanical simulation. Properties can appear more than once in a property list.

  Lists can be sorted in alphanumeric order. A sorted linear list is ordered by the values in the list. A sorted property list is ordered by the properties in the list. You sort a list by using the appropriate command for a linear list or property list.

- In linear lists, symbols and strings are case sensitive.
- In property lists, symbols aren't case sensitive, but strings are case sensitive.

  A linear list or property list can contain no values at all. An empty list consists of two square brackets ([ ]). To create or clear a linear list, set the list to `[ ]`. To create or clear a property list, set the list to [:].

  You can modify, test, or read items in a list.

  Lingo treats an instance of a list as a reference to the list. This means each instance is the same piece of data, and changing it will change the original. Use the `duplicate` command to create copies of lists.

  Lists are automatically disposed when they are no longer referred to by any variable. When a list is held within a global variable, it persists from movie to movie.

  You can initialize a list in the `on prepareMovie` handler or write the list as a field cast member, assign the list to a variable, and then handle the list by handling the variable.

  Not all PC keyboards have square brackets. If square brackets aren't available, use the `list` function to create a linear list.

  For a property list, create the list pieces as a string before converting them into a useful list.

```
myListString = numToChar(91) & ":" & numToChar(93)
put myListString
-- "[:]"
myList = myListString.value
put myList
-- [:]
put myList.listP
-- 1
myList[#name] = "Brynn"
put myList
```

```
-- [#name: "Brynn"]
```

Example This statement defines a list by making the `machinery` variable equal to the list:

```
set machinery = [#gears:6, #balls:3, #ramps:8]
```

Example This handler sorts the list aList and then displays the result in the Message window:

```
on sortList aList
  alist.sort()
  put aList
end sortList
```

If the movie issues the statement `sortList machinery`, where `machinery` is the list in the preceding example, the result is [#balls:3, #gears:6, #ramps:8].

Example This statement creates an empty linear list:

```
set x = [ ]
```

Example This statement creates an empty property list:

```
set x = [:]
```

See also **add**, **addAt**, **append**, **count()**,**deleteAt**, **duplicate() (list function)**, **findPos**, **findPosNear**, **getProp()**,**getAt**, **getLast()**,**getPos()**,**ilk()**,**list()**,**max()**,**min**, **setAt**, **setaProp**, **sort**

" (string)

Syntax "

Description String constant; when used before and after a string, quotation marks indicate that the string is a literal—not a variable, numerical value, or Lingo element. Quotation marks must always surround literal names of cast members, casts, windows, and external files.

Example This statement uses quotation marks to indicate that the string “San Francisco” is a literal string, the name of a cast member:

```
put member("San Francisco").loaded
```

See also **QUOTE**

¬ (continuation)

Description This symbol is obsolete. Use the \ character instead. See **\ (continuation)**.

\ (continuation)

Syntax

```
first part of a statement on this line \
second part of the statement \
third part of the statement
```

Description

Continuation symbol; when used as the last character in a line, indicates that the statement continues on the next line. Lingo then interprets the lines as one continuous statement.

Example

This statement uses the \ character to wrap the statement onto two lines:

```
set the memberNum of sprite mySprite \
to member "This is a long cast name."
```

@ (pathname)

Syntax `@pathReference`

Description Pathname operator; defines the path to the current movie's folder and is valid on both Windows and Macintosh computers.

Identify the current movie's folder by using the `@` symbol followed by one of these pathname separators:

- `/` (forward slash)
- `\` (backslash)
- `:` (colon)

When a movie is queried to determine its location, the string returned will include the @ symbol.

Be sure to use only the @ symbol when navigating between Director movies or changing the source of a linked media cast member. The @ symbol does not work when the FileIO Xtra or other functions are used outside those available within Director.

You can build on this pathname to specify folders that are one or more levels above or below the current movie's folder. Keep in mind that the @ portion represents the current movie's location, not necessarily the location of the projector.

- Add an additional pathname separator immediately after the `@` symbol to specify a folder one level up in the hierarchy.
- Add folder and file names (separated by `/`, `\`, or `:`) after the current folder name to specify subfolders and files within folders.

You can use relative pathnames in Lingo to indicate the location of a linked file in a folder different than the movie's folder.

Example These are equivalent expressions that specify the subfolder bigFolder, which is in the current movie's folder:

```
@/bigFolder
@:bigFolder
@\bigFolder
```

Example These are equivalent expressions that specify the file linkedFile, in the subfolder bigFolder, which is in the current movie's folder:

```
@:bigFolder:linkedFile
@\bigFolder\linkedFile
@/bigFolder/linkedFile
```

Example This expression specifies the file linkedFile, which is located one level up from the current movie's folder:

```
@//linkedFile
```

Example This expression specifies the file linkedFile, which is located two levels up from the current movie's folder:

```
@:::linkedFile
```

Example These are equivalent expressions that specify the file linkedFile, which is in the folder otherFolder. The otherFolder folder is in the folder one level up from the current movie's folder.

```
@::otherFolder:linkedFile
@\\otherFolder\linkedFile
@//otherFolder/linkedFile
```

See also **searchPath**, **fileName (cast property)**, **fileName (cast member property)**, **fileName (window property)**

abbr, abbrev, abbreviated

These elements are used by the `date` and `time` functions.

See also **date() (system clock)**

abort

Syntax `abort`

Description Command; tells Lingo to exit the current handler and any handler that called it without executing any of the remaining statements in the handler. This differs from the `exit` keyword, which returns to the handler from which the current handler was called.

The `abort` command does not quit Director.

Example This statement instructs Lingo to exit the handler and any handler that called it when the amount of free memory is less than 50K:

```
if the freeBytes < 50*1024 then abort
```

See also **exit**, **halt**, **quit**

abs()

Syntax `abs (numericExpression)`

Description Math function; calculates the absolute value of a numerical expression. If *`numericExpression`* is an integer, its absolute value is also an integer. If *`numericExpression`* is a floating-point number, its absolute value is also a floating-point number.

The `abs` function has several uses. It can simplify the tracking of mouse and sprite movement by converting coordinate differences (which can be either positive or negative numbers) into distances (which are always positive numbers). The `abs` function is also useful for handling mathematical functions, such as `sqrt` and `log`.

Example This statement determines whether the absolute value of the difference between the current mouse position and the value of the variable `startV` is greater than 30 (since you wouldn't want to use a negative number for distance). If it is, the foreground color of sprite 6 is changed.

```
if (the mouseV - startV).abs > 30 then sprite(6).forecolor = 95
```

actionsEnabled

Syntax

```
the actionsEnabled of sprite whichFlashSprite
the actionsEnabled of member whichFlashMember
sprite whichFlashSprite.actionenabled
member whichFlashMember.actionenabled
```

Description

Cast member property and sprite property; controls whether the actions in a Flash movie are enabled (`TRUE`, default) or disabled (`FALSE`).

This property can be tested and set.

Example

This handler accepts a sprite reference as a parameter, and then toggles the sprite's `actionsEnabled` property on or off.

Dot syntax:

```
on ToggleActions whichSprite
  sprite (whichSprite).actionsEnabled = not sprite
(whichSprite).actionsEnabled
end
```

Verbose syntax:

```
on ToggleActions whichSprite
  set the actionsEnabled of sprite whichSprite = not the
actionsEnabled of sprite whichSprite
end
```

activateApplication

Syntax `on activateApplication`

Description Built-in handler; runs when the projector is brought to the foreground. This handler is useful when a projector runs in a window and the user can send it to the background to work with other applications. When the projector is brought back to the foreground, this handler runs. Any MIAWs running in the projector can also make use of this handler.

During authoring, this handler is called only if Animate in Background is turned on in General Preferences.

On Windows, this handler is not called if the projector is merely minimized and no other application is brought to the foreground.

Example This handler plays a sound each time the user brings the projector back to the foreground:

```
on activateApplication
  sound(1).queue(member("openSound"))
  sound(1).play()
end
```

See also **deactivateApplication**, **on activateWindow**, **on deactivateWindow**

on activateWindow

Syntax

```
on activateWindow
     statement(s)
end
```

Description

System message and event handler; contains statements that run in a movie when the user clicks the inactive window and the window comes to the foreground.

You can use an `on activateWindow` handler in a script that you want executed every time the movie becomes active.

Clicking the main movie (the main Stage) does not generate an `on activateWindow` handler.

Example

This handler plays the sound Hurray when the window that the movie is playing in becomes active:

```
on activateWindow
  puppetSound 2, "Hurray"
end
```

See also

activeWindow, **close window**, **on deactivateWindow**, **frontWindow**, **on moveWindow**, **open**

activeCastLib

Syntax `the activeCastLib`

Description System property; indicates which cast was most recently activated. The `activeCastLib` property's value is the cast's number.

The `activeCastLib` property is useful when working with the `selection castLib` property. Use it to determine which cast the selection refers to.

This property can be tested but not set.

Example These statements assign the selected cast members in the most recently selected cast to the variable `selectedMembers`:

```
castLibOfInterest = the activeCastLib
selectedMembers = castLib(castLibOfInterest).selection
```

An equivalent way to write this is:

```
selectedMembers = castLib(the activeCastLib).selection
```

activeWindow

Syntax `the activeWindow`

Description Movie property; indicates which movie window is currently active. For the main movie, `activeWindow` is the Stage. For a movie in a window, `activeWindow` is the movie in the window.

Example This example places the word Active in the title bar of the clicked window and places the word Inactive in the title bar of all other open windows:

```
on activateWindow
   set clickedWindow = (the windowlist).getPos(the activeWindow)
   set windowCount = (the windowlist).count
   repeat with x = 1 to windowCount
    if x = clickedWindow then
             (the activeWindow).title = "Active"
    else
                   (the windowlist[x]).title = "Inactive"
      end if
   end repeat
end
```

See also **on activateWindow**, **windowList**

actorList

Syntax `the actorList`

Description Movie property; a list of child objects that have been explicitly added to this list. Objects in `actorList` receive a `stepFrame` message each time the playback head enters a frame.

To add an object to `the actorList`, use `add actorList, theObject`. The object's `on stepFrame` handler in its parent or ancestor script will then be called automatically at each frame advance.

To clear objects from `the actorList`, set `actorList` to `[ ]`, which is an empty list.

Director doesn't clear the contents of `actorList` when branching to another movie, which can cause unpredictable behavior in the new movie. To prevent child objects in the current movie from being carried over to the new movie, insert the statement `set the actorList = [ ]` in the `on prepareMovie` handler of the new movie.

Unlike previous versions of Director, `actorList` is now supported in the Director player for Java.

Example This statement adds a child object created from the parent script Moving Ball. All three values are parameters that the script requires.

```
add the actorList, new(script "MovingBall", 1, 200,200)
```

Example This statement displays the contents of `actorList` in the Message window:

```
put the actorList
```

Example This statement clears objects from `actorList`.

```
the actorList = [ ]
```

See also **new()**

add

Syntax `linearList.add(value)`

`add linearList, value`

Description List command; for linear lists only, adds the value specified by *`value`* to the linear list specified by *`linearList`*. For a sorted list, the value is placed in its proper order. For an unsorted list, the value is added to the end of the list.

This command returns an error when used on a property list.

> Note: Don't confuse the `add` command with the + operator used for addition or the & operator used to concatenate strings.

Example These statements add the value 2 to the list named `bids`. The resulting list is [3, 4, 1, 2].

```
bids = [3, 4, 1]
bids.add(2)
```

Example This statement adds 2 to the sorted linear list [1, 4, 5]. The new item remains in alphanumeric order because the list is sorted.

```
bids.add(2)
```

See also **sort**

addAt

Syntax

```
list.AddAt(position, value)
addAt list, position, value
```

Description

List command; for linear lists only, adds a value specified by *value* to a list at the position specified by *position*.

This command returns an error when used with a property list.

Example

This statement adds the value 8 to the fourth position in the list named `bids`, which is [3, 2, 4, 5, 6, 7]:

```
bids = [3, 2, 4, 5, 6, 7]
bids.addAt(4,8)
```

The resulting value of `bids` is [3, 2, 4, 8, 5, 6, 7].

addProp

Syntax

```
list.addProp(property, value)
addProp list, property, value
```

Description

Property list command; for property lists only, adds the property specified by *property* and its value specified by *value* to the property list specified by *list*. For an unsorted list, the value is added to the end of the list. For a sorted list, the value is placed in its proper order.

If the property already exists in the list, Lingo creates a duplicate property. You can avoid duplicate properties by using the `setaProp` command to change the new entry's property.

This command returns an error when used with a linear list.

Example

This statement adds the property named `kayne` and its assigned value 3 to the property list named `bids`, which contains [#gee: 4, #ohasi: 1]. Because the list is sorted, the new entry is placed in alphabetical order:

```
bids.addProp(#kayne, 3)
```

The result is the list [#gee: 4, #kayne: 3, #ohasi: 1].

Example

This statement adds the entry `kayne: 7` to the list named `bids`, which now contains [#gee: 4, #kayne: 3, #ohasi: 1]. Because the list already contains the property `kayne`, Lingo creates a duplicate property:

```
bids.addProp(#kayne, 7)
```

The result is the list [#gee: 4, #kayne: 3, #kayne: 7, #ohasi: 1].

addVertex

Syntax

```
member(memberRef).AddVertex(indexToAddAt, pointToAddVertex \
{,[ controlLocH, controlLocV ], [ controlLocH, controlLocV ]})
addVertex(member memberRef, indexToAddAt, pointToAddVertex \
{,[controlLocH, controlLocV], [controlLocH,controlLocV]})
```

Description

Vector shape command; adds a new vertex to a vector shape cast member in the position specified.

The horizontal and vertical positions are relative to the origin of the vertex shape cast member.

When using the final two optional parameters, you can specify the location of the control handles for the vertex. The control handle location is offset relative to the vertex, so if no location is specified, it will be located at 0 horizontal offset and 0 vertical offset.

Example

This line adds a vertex point in the vector shape Archie between the two existing vertex points, at the position 25 horizontal and 15 vertical:

```
member("Archie").addVertex(2, point(25, 15))
```

See also

vertexList, **moveVertex()**,**addVertex**, **deleteVertex()**,**originMode**

after

See **put...after** command

alert

Syntax `alert` *`message`*

Description Command; causes a system beep and displays an alert dialog box containing the string specified by *`message`* and an OK button. This command is useful for providing error messages of up to 255 characters in your movie.

The message must be a string. If you want to include a number variable in an alert, use the `string` function to convert the variable to a string.

Example The following statement produces an alert stating that there is no CD-ROM drive connected:

```
alert "There is no CD-ROM drive connected."
```

Example This statement produces an alert stating that a file was not found:

```
alert "The file" && QUOTE & filename & QUOTE && "was not found."
```

See also **string()**,**alertHook**

alertHook

Syntax `the alertHook`

Description System property; specifies a parent script that contains the `on alertHook` handler. Use `alertHook` to control the display of alerts about file errors or Lingo script errors. When an error occurs and a parent script is assigned to `alertHook`, Director runs the `on alertHook` handler in the parent script.

Although it is possible to place `on alertHook` handlers in movie scripts, it is strongly recommended that you place an `on alertHook` handler in a behavior or parent script to avoid unintentionally calling the handler from a wide variety of locations and creating confusion about where the error occurred.

Because the `on alertHook` handler runs when an error occurs, avoid using the `on alertHook` handler for Lingo that isn't involved in handling an error. For example, the `on alertHook` handler is a bad location for a `go to movie` statement.

The `on alertHook` handler is passed an instance argument and two string arguments that describe the error. Depending on the Lingo within it, the `on alertHook` handler can ignore the error or report it in another way.

Example The following statement specifies that the parent script Alert is the script that determines whether to display alerts when an error occurs. If an error occurs, Lingo assigns the error and message strings to the field cast member Output and returns the value 1.

```
on prepareMovie
  the alertHook = script "Alert"
end
-- parent script "Alert"
on alertHook me, err, msg
  member("Output").text = err && msg
  return 1
end
```

alignment

Syntax

```
member(whichCastMember).alignment
the alignment of member whichCastMember
```

Description

Cast member property; determines the alignment used to display characters within the specified cast member. This property appears only to field and text cast members containing characters, if only a space.

For field cast members, the value of the property is a string consisting of one of the following: left, center, or right.

For text cast members, the value of the property is a symbol consisting of one of the following: #left, #center, #right, or #full.

The parameter *whichCastMember* can be either a cast name or a cast number.

This property can be tested and set. For text cast members, the property can be set on a per-paragraph basis.

Example

This statement sets the variable named `characterAlign` to the current `alignment` setting for the field cast member Rokujo Speaks:

Dot syntax:

```
characterAlign = member("Rokujo Speaks").alignment
```

Verbose syntax:

```
set characterAlign = the alignment of member "Rokujo Speaks"
```

Example

This repeat loop consecutively sets the alignment of the field cast member Rove to left, center, and then right.

Dot syntax:

```
repeat with i = 1 to 3
  member("Rove").alignment = ("left center right").word[i]
end repeat
```

Verbose syntax:

```
repeat with i = 1 to 3
  set the alignment of member "Rove" to word i of "left center
right"
end repeat
```

See also

text, **font**, **lineHeight (cast member property)**, **fontSize**, **fontStyle**, **& (concatenator)**, **&& (concatenator)**

allowCustomCaching

Syntax `the allowCustomCaching`

Description Movie property; will contain information regarding a private cache in future versions of Director.

This property defaults to `TRUE`, and can be tested and set.

See also **allowGraphicMenu**, **allowSaveLocal**, **allowTransportControl**, **allowVolumeControl**, **allowZooming**

allowGraphicMenu

Syntax `the allowGraphicMenu`

Description Movie property; sets the availability of the graphic controls in the context menu when playing the movie in a Shockwave environment.

Set this property to `FALSE` if you would rather have a text menu displayed than the graphic context menu.

This property defaults to `TRUE`, and can be tested and set.

Example the allowGraphicMenu = 0

See also **allowSaveLocal**, **allowTransportControl**, **allowVolumeControl**, **allowZooming**

allowSaveLocal

Syntax `the allowSaveLocal`

Description Movie property; sets the availability of the Save control in the context menu when playing the movie in a Shockwave environment.

This property is provided to allow for enhancements in future versions of Shockwave.

This property defaults to `TRUE`, and can be tested and set.

See also **allowGraphicMenu**, **allowTransportControl**, **allowVolumeControl**, **allowZooming**

allowTransportControl

Syntax `the allowTransportControl`

Description Movie property; This property is provided to allow for enhancements in future versions of Shockwave.

This property defaults to `TRUE`, and can be tested and set.

See also **allowGraphicMenu**, **allowSaveLocal**, **allowVolumeControl**, **allowZooming**

allowVolumeControl

Syntax `the allowVolumeControl`

Description Movie property; sets the availability of the volume control in the context menu when playing the movie in a Shockwave environment.

When set to `TRUE` one or the other volume control is active, and is disabled when the property is set to `FALSE`.

This property defaults to `TRUE`, and can be tested and set.

See also **allowGraphicMenu**, **allowSaveLocal**, **allowTransportControl**, **allowZooming**

allowZooming

Syntax `the allowZooming`

Description Movie property; determines whether the movie may be stretched or zoomed by the user when playing back in Shockwave and ShockMachine. Defaults to `TRUE`. This property can be tested and set. Set this property to `FALSE` to prevent users from changing the size of the movie in browsers and ShockMachine.

See also **allowGraphicMenu**, **allowSaveLocal**, **allowTransportControl**, **allowVolumeControl**

alphaThreshold

Syntax

```
member(whichMember).alphaThreshold
the alphaThreshold of member whichMember
```

Description

Bitmap cast member property; governs how the bitmap's alpha channel affects hit detection. This property is a value from 0 to 255, that exactly matches alpha values in the alpha channel for a 32-bit bitmap image.

For a given `alphaThreshold` setting, Director detects a mouse click if the pixel value of the alpha map at that point is equal to or greater than the threshold. Setting the `alphaThreshold` to 0 makes all pixels opaque to hit detection regardless of the contents of the alpha channel.

See also **useAlpha**

ancestor

Syntax `property {optionalProperties} ancestor`

Description Object property; allows child objects and behaviors to use handlers that are not contained within the parent script or behavior.

The `ancestor` property is typically used with two or more parent scripts. You can use this property when you want child objects and behaviors to share certain behaviors that are inherited from an ancestor, while differing in other behaviors that are inherited from the parents.

For child objects, the `ancestor` property is usually assigned in the `on new` handler within the parent script. Sending a message to a child object that does not have a defined handler forwards that message to the script defined by the `ancestor` property.

If a behavior has an ancestor, the ancestor receives mouse events such as `mouseDown` and `mouseWithin`.

The `ancestor` property lets you change behaviors and properties for a large group of objects with a single command.

The ancestor script can contain independent property variables that can be obtained by child objects. To refer to property variables within the ancestor script, you must use this syntax:

me.propertyVariable = value

For example, this statement changes the property variable `legCount` within an ancestor script to 4:

```
me.legCount = 4
```

Use the syntax `the variableName of scriptName` to access property variables that are not contained within the current object. This statement allows the variable `myLegCount` within the child object to access the property variable `legCount` within the ancestor script:

```
set myLegCount to the legCount of me
```

Example Each of the following scripts is a cast member. The ancestor script Animal and the parent scripts Dog and Man interact with one another to define objects.

The first script, Dog, sets the property variable `breed` to Mutt, sets `the ancestor of Dog` to the Animal script, and sets the `legCount` variable that is stored in the ancestor script to 4:

```
property breed, ancestor
on new me
  set breed = "Mutt"
  set the ancestor of me to new(script "Animal")
  set the legCount of me to 4
  return me
end
```

The second script, Man, sets the property variable `race` to Caucasian, sets `the ancestor of Man` to the Animal script, and sets the `legCount` variable that is stored in the ancestor script to 2:

```
property race, ancestor
on new me
  set race to "Caucasian"
```

```
  set the ancestor of me to new(script "Animal")
  set the legCount of me to 2
  return me
end
```

See also **new()**;**me**, **property**

and

Syntax `logicalExpression1 and logicalExpression2`

Description Logical operator; determines whether both *`logicalExpression1`* and *`logicalExpression2`* are `TRUE (1)`, or whether either or both expressions are `FALSE` (0).

The precedence level of this logical operator is 4.

Example This statement determines whether both logical expressions are `TRUE` and displays the result in the Message window:

```
put 1 < 2 and 2 < 3
```

The result is 1, which is the numerical equivalent of `TRUE`.

Example The first logical expression in this statement is `TRUE`; and the second logical expression is `FALSE`. Because both logical expressions are not `TRUE`, the logical operator displays the result 0, which is the numerical equivalent of `FALSE`.

```
put 1 < 2 and 2 < 1
-- 0
```

See also **not** and **or**

antiAlias

Syntax `member(whichMember).antiAlias`

sprite(*whichVectorSprite*).antiAlias

Description Cast member property; controls whether a text, Vector shape, or Flash cast member is rendered using anti-aliasing to produce high-quality rendering, but possibly slower playback of the movie. The `antiAlias` property is `TRUE` by default.

For vector shapes, `TRUE` is the equivalent of the `#high` quality setting for a Flash asset, and `FALSE` is the equivalent of `#low`.

The `antiAlias` property may also be used as a sprite property only for Vector shape sprites.

This property can be tested and set.

Example This behavior checks the color depth of the computer on which the movie is playing. If the color depth is set to 8 bits or less (256 colors), the script sets the `antiAlias` of the sprite to `FALSE`.

```
property spriteNum

on beginsprite me
  if the colorDepth <= 8 then
    sprite(spriteNum).antiAlias = FALSE
  end if
end
```

See also **antiAliasThreshold**, **quality**

antiAliasThreshold

Syntax `member(whichTextMember).antiAliasThreshold`

Description Text cast member property; this setting controls the point size at which automatic anti-aliasing takes place in a text cast member. This has an effect only when the `antiAlias` property of the text cast member is set to TRUE.

The setting itself is an integer indicating the font point size at which the anti- alias takes place.

This property defaults to 14 points.

See also **antiAlias**

append

Syntax

```
list.append(value)
append list, value
```

Description

List command; for linear lists only, adds the specified value to the end of a linear list. This differs from the `add` command, which adds a value to a sorted list according to the list's order.

This command returns a script error when used with a property list.

Example

This statement adds the value 2 at the end of the sorted list named bids, which contains [1, 3, 4], even though this placement does not match the list's sorted order:

```
set bids = [1, 3, 4]
bids.append(2)
```

The resulting value of bids is [1, 3, 4, 2].

See also

add, **sort**

applicationPath

Syntax `the applicationPath`

Description System property; determines the path or location of the folder containing the running copy of the Director application during authoring, or the folder containing the projector during run time. The property value is a string.

If you use `the applicationPath` followed by `&` and a path to a subfolder, enclose the entire expression in parentheses so that Lingo parses the expression as one phrase.

The Director player for Java doesn't support this property, nor does Shockwave.

This property can be tested but not set.

Example This statement displays the pathname for the folder that contains the Director application.

```
put the applicationPath
--"Z:\Program Files\Macromedia\Director"
```

Example This statement opens the movie Sunset Boulevard in a window (on a Windows machine):

```
open window (the applicationPath & "\Film Noir\Sunset Boulevard")
```

See also **@ (pathname)**, **moviePath**

appMinimize

Syntax `appMinimize`

Description Command; on Windows, `appMinimize` causes a projector to minimize to the Windows Task Bar. On the Macintosh, `appMinimize` causes a projector to be hidden. Once hidden, the projector may be re-opened from the Macintosh application menu.

This is useful for projectors and MIAW's that play back without a title bar.

See also **windowType**

atan()

Syntax

```
(number).atan
atan (number)
```

Description

Math function; calculates the arctangent, which is the angle whose tangent is a specified number. The result is a value in radians between pi/2 and +pi/2.

Example

This statement displays the arctangent of 1:

```
(1).atan
```

The result, to four decimal places, is 0.7854, or approximately pi/4.

Note that most trigonometric functions use radians, so you may want to convert from degrees to radians.

Example

This handler lets you convert between degrees and radians:

```
on DegreesToRads degreeValue
  return degreeValue * PI/180
end
```

The handler displays the conversion of 30 degrees to radians in the Message window:

```
put DegreesToRads(30)
-- 0.5236
```

See also

cos(), **PI**, and **sin()**

autoMask

Syntax

```
member(whichCursorCastMember).autoMask
```

```
the autoMask of member whichCastMember
```

Description

Cast member property; specifies whether the white pixels in the animated color cursor cast member *whichCursorCastMember* are transparent, allowing the background to show through (`TRUE`, default), or opaque (`FALSE`).

Example

In this script, when the custom animated cursor stored in cast member 5 enters the sprite, the automask is turned on so that the background of the sprite will show through the white pixels. When the cursor leaves the sprite, the automask is turned off.

Using dot syntax, the script is written as:

```
on mouseEnter
  member 5.autoMask = TRUE
end
on mouseLeave
  member 5.autoMask = FALSE
end
```

Using traditional Lingo syntax, the script is written as:

```
on mouseEnter
  set the autoMask of member 5 = TRUE
end
on mouseLeave
  set the autoMask of member 5 = FALSE
end
```

autoTab

Syntax

```
member(whichCastMember).autoTab
```

```
the autoTab of member whichCastMember
```

Description

Cast member property; determines the effect that pressing the Tab key has on the editable field or text cast member specified by *whichCastMember*. The property can be made active (TRUE) or inactive (FALSE). Tabbing order depends on sprite number order, not position on the Stage.

This property is always TRUE in an applet created with the Save as Java feature of Director.

Example

This statement causes the cast member Comments to automatically advance the insertion point to the next editable field or text sprite after the user presses Tab.

Dot syntax:

```
member ("Comments").autotab = TRUE
```

Verbose Lingo syntax:

```
set the autoTab of member "Comments" to TRUE
```

backColor

Syntax
```
member(whichCastMember).backColor = colorNumber
set the backColor of member whichCastMember to colorNumber
sprite(whichSprite).backColor
the backColor of sprite whichSprite
```

Description Cast member and sprite property; sets the background color of the specified cast member or sprite according to the color value assigned.

- For cast members: it affects field or button cast member displays.
- For sprites: setting `the backColor` of a sprite is the same as choosing the background color from the tool palette when the sprite is selected on the Stage. For the value that Lingo sets to last beyond the current sprite, the sprite must be a puppet. The background color applies only to 1-bit bitmap and shape cast members.

The `backColor` value ranges from 0 to 255 for 8-bit color and from 0 to 15 for 4-bit color. The numbers correspond to the index number of the background color in the current palette. (A color's index number appears in the color palette's lower left corner when you click the color.)

You should not apply this property to bitmap cast members deeper than 1-bit, as the results are difficult to predict.

For a movie that plays back as an applet created with the Save as Java feature of Director, specify colors for `backColor` using the decimal equivalent of the 24-bit hexadecimal values used in an HTML document.

For example, the hexadecimal value for pure red, FF0000, is equivalent to 16711680 in decimal numbers. This statement specifies pure red as a cast member's background color:

```
set the backColor of member = 16711680
```

This property can be tested and set.

Example This statement changes the color of the characters in cast member 1 to the color in palette entry 250.

Dot syntax:

```
member(1).backColor = 250
```

Verbose Lingo syntax:

```
set the backColor of member 1 to 250
```

Example The following statement sets the variable `oldColor` to the background color of sprite 5:

```
oldColor = sprite (5).backColor
```

Example The following statement randomly changes the background color of a random sprite between sprites 11 and 13 to color number 36:

```
sprite(10 + random(3)).backColor = 36
```

See also **bgColor**, **color (sprite property)**

backgroundColor

Syntax `member(whichVectorMember).backgroundColor`

`the backgroundColor of member whichVectorMember`

Description Vector shape cast member property; sets the background color of the specified cast member or sprite to the RGB color value assigned.

This property can be both tested and set.

Example member("Archie").backgroundColor= rgb(255,255,255)

See also **bgColor**

BACKSPACE

Syntax `BACKSPACE`

Description Constant; represents the Backspace key. This key is labeled Backspace in Windows and Delete on the Macintosh.

Example This `on keyDown` handler checks whether the Backspace key was pressed and, if it was, calls the handler `clearEntry`:

```
on keyDown
  if the key = BACKSPACE then clearEntry
  stopEvent
end keyDown
```

beep

Syntax `beep {numberOfTimes}`

Description Command; causes the computer's speaker to beep the number of times specified by *numberOfTimes*. If *numberOfTimes* is missing, the beep occurs once.

- In Windows, the beep is the sound assigned in the Sounds Properties dialog box.
- For the Macintosh, the beep is the sound selected from Alert Sounds on the Sound control panel. If the volume on the Sound control panel is set to 0, the menu bar flashes instead.

Example This statement causes two beeps if the Answer field is empty:

```
if field "Answer" = EMPTY then beep 2
```

beepOn

Syntax `the beepOn`

Description Movie property; determines whether the computer automatically beeps when the user clicks on anything except an active sprite (`TRUE`), or not (`FALSE`, default).

Scripts that set `beepOn` should be placed in frame or movie scripts.

This property can be tested and set.

Example This statement sets `beepOn` to `TRUE`:

```
the beepOn = TRUE
```

Example This statement sets `beepOn` to the opposite of its current setting:

```
the beepOn = not the beepOn
```

before

See **put...before**

beginRecording

Syntax `beginRecording`

Description Keyword; starts a Score generation session. Only one update session in a movie can be active at a time.

Every `beginRecording` keyword must be matched by an `endRecording` keyword that ends the Score generation session.

Example When used in the following handler, the `beginRecording` keyword begins a Score generation session that animates the cast member Ball by assigning the cast member to sprite channel 20 and then moving the sprite horizontally and vertically over a series of frames. The number of frames is determined by the argument `numberOfFrames`.

```
on animBall numberOfFrames
  beginRecording
    horizontal = 0
    vertical = 100
    repeat with i = 1 to numberOfFrames
      go to frame i
      sprite(20).member = member "Ball"
      sprite(20).locH = horizontal
      sprite(20).locV = vertical
      sprite(20).type = 1
      sprite(20).foreColor = 255
      horizontal = horizontal + 3
      vertical = vertical + 2
      updateFrame
    end repeat
  endRecording
end
```

See also **endRecording**, **updateFrame**, **scriptNum**, **tweened**

on beginSprite

Syntax

```
on beginSprite
  statement(s)
end
```

Description

System message and event handler; contains statements that run when the playback head moves to a frame that contains a sprite that was not previously encountered. Like `endSprite`, this event is generated only one time, even if the playback head loops on a frame, since the trigger is a sprite not previously encountered by the playback head. The event is generated before `prepareFrame`.

Director creates instances of any behavior scripts attached to the sprite when the `beginSprite` message is sent.

The object reference `me` is passed to this event if it is used in a behavior. The message is sent to behaviors and frame scripts.

If a sprite begins in the first frame that plays in the movie, the `beginSprite` message is sent after the `prepareMovie` message but before the `prepareFrame` and `startMovie` messages.

> Note: Be aware that some sprite properties, such as the `rect` sprite property, may not be accessible in a beginSprite handler. This is because the property needs to be calculated, which is not done until the sprite is drawn.

The `go`, `play`, and `updateStage` commands are disabled in an `on beginSprite` handler.

Example

This handler plays the sound cast member Stevie Wonder when the sprite begins:

```
on beginSprite me
  puppetSound "Stevie Wonder"
end
```

See also

on endSprite, **on prepareFrame**, **scriptInstanceList**

bgColor

Syntax

```
sprite(whichSpriteNumber).bgColor
the bgColor of sprite whichSpriteNumber
the bgColor of the stage
(the stage).bgColor
```

Description

Sprite property and system property; determines the background color of the sprite specified by *whichSprite* or the color of the Stage. Setting the `bgColor` sprite property is equivalent to choosing the background color from the Tools window when the sprite is selected on the Stage. Setting the `bgColor` property for the Stage is equivalent to setting the color in the Movie Properties dialog box.

The sprite property has the equivalent functionality of the `backColor` sprite property, but the color value returned is a color object of whatever type has been set for that sprite.

This property can be tested and set.

Example

This example sets the color of the Stage to an RGB value.

Dot syntax:

```
(the stage).bgColor = rgb(255, 153, 0)
```

Verbose Lingo syntax:

```
set the bgColor of the stage = rgb(255, 153, 0)
```

See also

color(),**backColor**, **backgroundColor**, **stageColor**

bitAnd()

Syntax `bitAnd(integer1, integer2)`

Description Function; converts the two specified integers to 32-bit binary numbers and returns a binary number whose digits are 1's in the positions where both numbers had a 1, and 0's in every other position. The result is the new binary number, which Lingo displays as a base 10 integer.

Integer	Binary number (abbreviated)
6	00110
7	00111
Result	
6	00110

Description

Example This statement compares the binary versions of the integers 6 and 7 and returns the result as an integer:

```
put bitAnd(6, 7)
-- 6
```

See also **bitNot()**,**bitOr()**,**bitXor()**

bitmapSizes

Syntax
```
member(whichFontMember).bitmapSizes
the bitmapSizes of member whichFontMember
```

Description Font cast member property; returns a list of the bitmap point sizes that were included when the font cast member was created.

Example This statement displays the bitmap point sizes that were included when cast member 11 was created:

```
put member(11).bitmapSizes
-- [12, 14, 18]
```

See also **recordFont**, **characterSet**, **originalFont**

bitNot()

Syntax
```
(integer).bitNot
bitNot(integer)
```

Description Function; converts the specified integer to a 32-bit binary number and reverses the value of each binary digit, replacing 1's with 0's and 0's with 1's. The result is the new binary number, which Lingo displays as a base 10 integer.

Integer	Binary number
1	00000000000000000000000000000001
Result	
-2	11111111111111111111111111111110

Description

Example This statement inverts the binary representation of the integer 1 and returns a new number.

```
put (1).bitNot
-- -2
```

See also **bitAnd()**,**bitOr()**,**bitXor()**

bitOr()

Syntax `bitOr(integer1, integer2)`

Description Function; converts the two specified integers to 32-bit binary numbers and returns a binary number whose digits are 1's in the positions where either number had a 1, and 0's in every other position. The result is the new binary number, which Lingo displays as a base 10 integer.

Integer	Binary number (abbreviated)
5	0101
6	0110
Result	
7	0111

Description

Example This statement compares the 32-bit binary versions of 5 and 6 and returns the result as an integer:

```
put bitOr(5, 6)
-- 7
```

See also **bitNot()**,**bitAnd()**,**bitXor()**

bitRate

Syntax

```
member(whichCastMember).bitRate
```

```
the bitRate of member whichCastMember
```

Description

Shockwave Audio (SWA) cast member property; returns the bit rate, in kilobits per second (Kbps), of the specified SWA cast member that has been preloaded from the server.

The `bitRate` member property returns 0 until streaming begins.

Example

This behavior outputs the bit rate of an SWA cast member when the sprite is first encountered.

Dot syntax:

```
property spriteNum
on beginSprite me
  memName = sprite(spriteNum).member.name
  put "The bitRate of
member"&&memName&&"is"&&member(memName).bitRate
end
```

Verbose syntax:

```
property spriteNum
on beginSprite me
  memName = sprite(spriteNum).member.name
  put "The bitRate of
member"&&memName&&"is"&&member(memName).bitRate
end
```

bitsPerSample

Syntax

```
member(whichCastMember).bitsPerSample
```

```
the bitsPerSample of member whichCastMember
```

Description

Shockwave Audio (SWA) cast member property; indicates the bit depth of the original file that has been encoded for Shockwave Audio (SWA). This property is available only after the SWA sound begins playing or after the file has been preloaded using the `preLoadBuffer` command.

This property can be tested but not set.

Example

This statement assigns the original bit rate of the file used in SWA streaming cast member Paul Robeson to the field cast member How Deep.

Dot syntax:

```
put member "Paul Robeson".bitsPerSample into member "How Deep"
```

Verbose syntax:

```
put the bitsPerSample of member "Paul Robeson" into member "How
Deep"
```

bitXor()

Syntax `bitXor(integer1, integer2)`

Description Function; converts the two specified integers to 32-bit binary numbers and returns a binary number whose digits are 1's in the positions where the given numbers' digits do not match, and 0's in the postions where the digits are the same. The result is the new binary number, which Lingo displays as a base 10 integer.

Integer	**Binary number (abbreviated)**
5	0101
6	0110
Result	
3	0011

Description

Example This statement compares the 32-bit binary versions of 5 and 6 and returns the result as an integer:

```
put bitXor(5, 6)
-- 3
```

See also **bitNot()**,**bitOr()**,**bitAnd()**

blend

Syntax
```
sprite(whichSprite).blend
the blend of sprite whichSprite
```

Description
Sprite property; sets or determines a sprite's blend value, from 0 to 100, corresponding to the blend values in the Sprite Properties dialog box.

The possible colors depend on the colors available in the palette, regardless of the monitor's color depth.

The Director player for Java supports the `blend` sprite property for bitmap sprites only.

For best results, use the blend ink with images that have a color depth greater than 8-bit.

Example
This statement sets the blend value of sprite 3 to 40 percent.

Dot syntax:

```
sprite(3).blend = 40
```

Verbose syntax:

```
set the blend of sprite 3 to 40
```

Example
This statement displays the blend value of sprite 3 in the Message window:

```
put the blend of sprite 3
```

See also
blendLevel

blendLevel

Syntax `sprite(`*`whichSpriteNumber`*`).blendLevel`

`the blendLevel of sprite` *`whichSpriteNumber`*

Description Sprite property; allows the current blending value of a sprite to be set or accessed. The possible range of values is from 0 to 255. This differs from the Sprite Inspector, which shows values in the range 0 to 100. The results are the same, the scales simply differ.

This property is the equivalent of the `blend` sprite property.

Example sprite(3).blendlevel = 99

See also **blend**

border

Syntax

```
member(whichFieldCastmember).border
```

```
the border of member whichFieldCastmember
```

Description

Field cast member property; indicates the width, in pixels, of the border around the specified field cast member.

Example

This statement makes the border around the field cast member Title 10 pixels wide.

Dot syntax:

```
member("Title").border = 10
```

Verbose syntax:

```
set the border of member "Title" to 10
```

bottom

Syntax

```
sprite(whichSprite).bottom
```

```
the bottom of sprite whichSprite
```

Description

Sprite property; specifies the bottom vertical coordinate of the bounding rectangle of the sprite specified by *whichSprite*.

When a movie plays back as an applet, this property's value is measured from the upper left corner of the applet.

This property can be tested and set.

Example

This statement assigns the vertical coordinate of the bottom of the sprite numbered (i + 1) to the variable named `lowest`.

Dot syntax:

```
set lowest = sprite (i + 1).bottom
```

Verbose syntax:

```
set lowest = the bottom of sprite (i + 1)
```

See also

height, **left**, **locH**, **locV**, **right**, **top**, and **width**

bottomSpacing

Syntax `chunkExpression.bottomSpacing`

Description Text cast member property; enables you to specify additional spacing applied to the bottom of each paragraph in the *`chunkExpression`* portion of the text cast member.

The value itself is an integer, where less than 0 indicates less spacing between paragraphs and greater than 0 indicates more spacing between paragraphs.

The default value is 0, which results in default spacing between paragraphs.

> Note: This property, like all text cast member properties, supports only dot syntax.

Example This example adds spacing after the first paragraph in cast member News Items.

```
member("News Items").paragraph[1].bottomSpacing=20
```

See also **topSpacing**

boxDropShadow

Syntax

```
member(whichCastMember).boxDropShadow
```

```
the boxDropShadow of member whichCastMember
```

Description

Cast member property; determines the size, in pixels, of the drop shadow for the box of the field cast member specified by *whichCastMember*.

Example

This statement makes the drop shadow of field cast member Title 10 pixels wide.

Dot syntax:

```
member("Title").boxDropShadow = 10
```

Verbose syntax:

```
set the boxDropShadow of member "Title" to 10
```

boxType

Syntax

```
member(whichCastMember).boxType
```

```
the boxType of member whichCastMember
```

Description

Cast member property; determines the type of text box used for the specified cast member. The possible values are `#adjust`, `#scroll`, `#fixed`, and `#limit`.

Example

This statement makes the box for field cast member Editorial a scrolling field.

Dot sytnax:

```
member("Editorial").boxType = #scroll
```

Verbose syntax:

```
set the boxType of member "Editorial" to #scroll
```

breakLoop()

Syntax `sound(channelNum).breakLoop()`

Description This function causes the currently looping sound in channel `channelNum` to stop looping and play through to its `endTime`. If there is no current loop, this function has no effect.

Example This handler causes the background music looping in sound channel 2 to stop looping and play through to its end:

```
on continueBackgroundMusic
  sound(2).breakLoop()
end
```

See also **end**, **loopCount**, **loopEndTime**, **loopsRemaining**, **loopStartTime**

broadcastProps

Syntax

```
member(whichVectorOrFlashMember).broadcastProps
```

```
the broadcastProps of member whichVectorOrFlashMember
```

Description

Cast member property; controls whether changes made to a Flash or Vector shape cast member are immediately broadcast to all of its sprites currently on the Stage (`TRUE`) or not (`FALSE`).

When this property is set to `FALSE`, changes made to the cast member are used only as defaults for new sprites and don't affect sprites on the Stage.

The default value for this property is `TRUE`, and it can be both tested and set.

Example

This frame script assumes that a Flash movie cast member named Navigation Movie has been set up with its `broadcastProps` property set to `FALSE`. The script momentarily allows changes to a Flash movie cast member to be broadcast to its sprites currently on the Stage. It then sets the `viewScale` property of the Flash movie cast member, and that change is broadcast to its sprite. The script then prevents the Flash movie from broadcasting changes to its sprites.

Dot syntax:

```
on enterFrame
  member("Navigation Movie").broadcastProps = TRUE
  member("Navigation Movie").viewScale = 200
  member("Navigation Movie").broadcastProps = FALSE
end
```

Verbose syntax:

```
on enterFrame
  set the broadcastProps of member "Navigation Movie" = TRUE
  set the viewScale of member "Navigation Movie" = 200
  set the broadcastProps of member "Navigation Movie" = FALSE
end
```

browserName()

Syntax

```
browserName pathName
browserName()
browserName(#enabled, trueOrFalse)
```

Description

System property, command, and function; specifies the path or location of the browser. You can use the FileIO Xtra to display a dialog box that allows the user to search for a browser. The `displayOpen()` method of the FileIO Xtra is useful for displaying an Open dialog box.

The form `browserName()` returns the name of the currently specified browser. Placing a pathname, like one found using the FileIO Xtra, as an argument in the form `browserName(fullPathToApplication)` allows the property to be set. The form `browserName(#enabled, trueOrFalse)` determines whether the specified browser launches automatically when the `goToNetPage` command is issued.

This command is only useful playing back in a projector or in Director, and has no effect when playing back in a browser.

This property can be tested and set.

Example

This statement refers to the location of the Netscape browser:

```
browserName "My Disk:My Folder:Netscape"
```

Example

This statement displays the browser name in a Message window:

```
put browserName()
```

bufferSize

Syntax

```
member(whichFlashMember).bufferSize
```

```
the bufferSize of member whichFlashMember
```

Description

Flash cast member property; controls how many bytes of a linked Flash movie are streamed into memory at one time. The `bufferSize` member property can have only integer values. This property has an effect only when the cast member's `preload` property is set to `FALSE`.

This property can be tested and set. The default value is 32,768 bytes.

Example

This `startMovie` handler sets up a Flash movie cast member for streaming and then sets its `bufferSize` property.

Dot syntax:

```
on startMovie
  member.("Flash Demo").preload = FALSE
  member.("Flash Demo").bufferSize = 65536
end
```

Verbose syntax:

```
on startMovie
  set the preload of member "Flash Demo" = FALSE
  set the bufferSize of member "Flash Demo" = 65536
end
```

See also

bytesStreamed, **preLoadRAM**, **stream**, **streamMode**

buttonsEnabled

Syntax

```
sprite(whichFlashSprite).buttonsEnabled
the buttonsEnabled of sprite whichFlashSprite
member(whichFlashMember).buttonsEnabled
the buttonsEnabled of member whichFlashMember
```

Description

Flash cast member property and sprite property; controls whether the buttons in a Flash movie are active (`TRUE`, default) or inactive (`FALSE`). Button actions are triggered only when the `actionsEnabled` property is set to `TRUE`.

This property can be tested and set.

Example

This handler accepts a sprite reference and toggles the sprite's `buttonsEnabled` property on or off.

Dot syntax:

```
on ToggleButtons whichSprite
  sprite(whichSprite).buttonsEnabled = not
sprite(whichSprite).buttonsEnabled
end
```

Verbose syntax:

```
on ToggleButtons whichSprite
  set the buttonsEnabled of sprite whichSprite = not the
buttonsEnabled of  \
  sprite whichSprite
end
```

See also

actionsEnabled

buttonStyle

Syntax `the buttonStyle`

Description Movie property; determines the visual response of buttons when the user rolls the pointer off them. This property applies only to buttons created with the Button tool in the Tool palette.

The `buttonStyle` property can have these values:

- 0 (list style: default)—Subsequent buttons are highlighted when the pointer passes over them. Releasing the mouse button activates the script associated with that button.
- `1` (dialog style)—Only the first button clicked is highlighted. Subsequent buttons are not highlighted. Releasing the mouse button while the pointer is over a button other than the original button clicked does not activate the script associated with that button.

This property can be tested and set in any type of script.

Example The following statement sets the `buttonStyle` property to 1:

```
the buttonStyle = 1
```

Example This statement remembers the current setting of the `buttonStyle` property by putting the current `buttonStyle` value in the variable `buttonStyleValue`:

```
buttonStyleValue = the buttonStyle
```

See also **checkBoxAccess**, **checkBoxType**

buttonType

Syntax

```
member(whichCastMember).buttonType
the buttonType of member whichCastMember
```

Description

Button cast member property; indicates the specified button cast member's type. Possible values are `#pushButton`, `#checkBox`, and `#radioButton`. This property applies only to buttons created with the button tool in the Tool palette.

Example

This statement makes the button cast member Editorial a check box.

Dot syntax:

```
member("Editorial").buttonType = #checkBox
```

Verbose syntax:

```
set the buttonType of member "Editorial" to #checkBox
```

bytesStreamed

Syntax

```
member(whichFlashOrSWAMember).bytesStreamed
```

```
the bytesStreamed of member whichFlashOrSWAMember
```

Description

Flash and Shockwave Audio cast member property; indicates the number of bytes of the specified cast member that have been loaded into memory. The `bytesStreamed` property returns a value only when the Director movie is playing. It returns an integer value.

This property can be tested but not set.

Example

This handler accepts a cast member reference as a parameter, and it then uses the `stream` command to load the cast member into memory. Every time it streams part of the cast member into memory, it uses the `bytesStreamed` property to report in the Message window how many bytes have been streamed.

Dot syntax:

```
on fetchMovie whichFlashMovie
  repeat while member(whichFlashMovie).percentStreamed < 100
    stream(member whichFlashMovie)
    put "Number of bytes streamed:" &&
member(whichFlashMovie).bytesStreamed
  end repeat
end
```

Verbose syntax:

```
on fetchMovie whichFlashMovie
  repeat while the percentStreamed of member whichFlashMovie <
100
    stream(member whichFlashMovie)
    put "Number of bytes streamed:" && the bytesStreamed of
member  \
    whichFlashMovie
  end repeat
end
```

See also

bufferSize, **bytesStreamed**, **percentStreamed**, **stream**

cacheDocVerify()

Syntax

```
cacheDocVerify #setting
cacheDocVerify()
```

Description

Function; sets how often the contents of a page on the Internet are refreshed with information from the projector's cache. Possible values are `#once` (default) and `#always`.

Specifying `#once` tells a movie to get a file from the Internet once and then use the file from the cache without looking for an updated version on the Internet.

Specifying `#always` tells a movie to try to get an updated version of the file each time the movie calls a URL.

The form `cacheDocVerify()` returns the current setting of the cache.

The `cacheDocVerify` function is valid only for movies running in Director or as projectors. This function is not valid for Shockwave movies because they use the network settings of the browser in which they run.

```
on resetCache
  current = cacheDocVerify()
  if current = #once then
    alert "Turning cache verification on"
    cacheDocVerify #always
  end if
end
```

See also

cacheSize(),clearCache

cacheSize()

Syntax

```
cacheSize Size
cacheSize()
```

Description

Function and command; sets Director's cache size. The value is in kilobytes.

The `cacheSize` function is valid only for movies running in Director or as projectors. This function is not valid for Shockwave movies because they use the network settings of the browser in which they run.

Example

This handler checks whether the browser's cache setting is less than 1 MB. If it is, the handler displays an alert and sets the cache size to 1 MB:

```
on checkCache if
  cacheSize()<1000 then
    alert "increasing cache to 1MB"
    cacheSize 1000
  end if
end
```

See also

cacheDocVerify(), **clearCache**

call

Syntax
```
call #handlerName, script, {args...}
call (#handlerName, scriptInstance, {args...})
```

Description Command; sends a message that invokes a handler in specified scripts where *handlerName* is the name of the handler to be activated, *script* references the script or a list of scripts, and *args* are any optional parameters to be passed to the handler.

If *script* is a single script instance, an error alert occurs if the handler is not defined in the script's ancestor script.

If *script* is a list of script instances, the message is sent to each item in the list in turn; if the handler is not defined in the ancestor script, no alert is generated.

The `call` command can use a variable as the name of the handler. Messages passed using `call` are not passed to other scripts attached to the sprite, cast member scripts, frame scripts, or movie scripts.

Example This handler sends the message bumpCounter to the first behavior script attached to sprite 1:

```
on mouseDown me
  -- get the reference to the first behavior of sprite 1
  set xref = getAt (the scriptInstanceList of sprite 1,1)
  -- run the bumpCounter handler in the referenced script,
  -- with a parameter
  call (#bumpCounter, xref, 2)
end
```

Example This example shows how a `call` statement can call handlers in a behavior or parent script and its ancestor.

- This is the parent script:

```
-- script Man
property ancestor
on new me
    set ancestor = new(script "Animal", 2)
    return me
end
on run me, newTool
    put "Man running with "&the legCount of me&" legs"
end
```

- This is the ancestor script:

```
-- script Animal
property legCount
on new me, newLegCount
    set legCount = newLegCount
    return me
end
on run me
    put "Animal running with "& legCount &" legs"
end
```

```
on walk me
    put "Animal walking with "& legCount &" legs"
end
```

- The following statements use the parent script and ancestor script.

  This statement creates an instance of the parent script:

  ```
  set m = new(script "man")
  ```

  This statement makes the man walk:

  ```
  call #walk, m
  -- "Animal walking with 2 legs"
  ```

  This statement makes the man run:

  ```
  set msg = #run
  call msg, m
  -- "Man running with 2 legs and rock"
  ```

  This statement creates a second instance of the parent script:

  ```
  set m2 = new(script "man")
  ```

  This statement sends a message to both instances of the parent script:

  ```
  call msg, [m, m2]
  -- "Man running with 2 legs "
  -- "Man running with 2 legs "
  ```

callAncestor

Syntax `callAncestor handlerName, script, {args...}`

Description Command; sends a message to a child object's ancestor script, where *handlerName* is the name of the handler to be activated, *script* references the script instance or a list of script instances, and *args* are any optional parameters to be passed to the handler.

If *script* is a single script instance, an error alert occurs if the handler is not defined in the ancestor of the script.

If *script* is a list of script instances, the message is sent to each item in the list in turn. In this case, if the handler is not defined in an ancestor script, no alert is generated.

Ancestors can, in turn, have their own ancestors.

When you use `callAncestor`, the name of the handler can be a variable, and you can explicitly bypass the handlers in the primary script and go directly to the ancestor script.

Example This example shows how a `callAncestor` statement can call handlers in the ancestor of a behavior or parent script.

- This is the parent script:

```
-- script "man"
property ancestor
on new me, newTool
  set ancestor = new(script "Animal", 2)
  return me
end
on run me
  put "Man running with "&the legCount of me&"legs"
end
```

- This is the ancestor script:

```
-- script "animal"
property legCount
on new me, newLegCount
  set legCount = newLegCount
  return me
end
on run me
  put "Animal running with "& legCount &" legs"
end
on walk me
  put "Animal walking with "& legCount &" legs"
end
```

- The following statements use the parent script and ancestor script.

  This statement creates an instance of the parent script:

```
set m = new(script "man")
```

  This statement makes the man walk:

```
call #walk, m
-- "Animal walking with 2 legs"
```

  This statement makes the man run:

```
set msg = #run
callAncestor msg, m
```

```
-- "Animal running with 2 legs"
```

This statement creates a second instance of the parent script:

```
set m2 = new(script "man")
```

This statement sends a message to the ancestor script for both men:

```
callAncestor #run,[m,m2]
-- "Animal running with 2 legs"
-- "Animal running with 2 legs"
```

See also **ancestor**, **new()**

cancelIdleLoad

Syntax `cancelIdleLoad` *`loadTag`*

Description Command; cancels the loading of all cast members that have the specified load tag.

Example This statement cancels the loading of cast members that have an idle load tag of 20:

```
cancelIdleLoad 20
```

See also **idleLoadTag**

case

Syntax

```
case expression of
  expression1 : Statement
  expression2 :
    multipleStatements
    .
    .
    .
  expression3, expression4 :
    Statement
  {otherwise:
    statement(s)}
end case
```

Description

Keyword; starts a multiple branching logic structure that is easier to write than repeated `if...then` statements.

Lingo compares the value in *`case expression`* to the expressions in the lines beneath it, starting at the beginning and continuing through each line in order, until Lingo encounters an expression that matches *`case expression`*.

When Lingo finds a matching expression, it executes the corresponding statement or statements that follow the colon after the matching expression. When only one statement follows the matching expression, the matching expression and its corresponding statement may appear on the same line. Multiple statements must appear on indented lines immediately below the matching expression.

When more than one possible match could cause Lingo to execute the same statements, the expressions must be separated by commas. (The syntax line containing *`expression3`* and *`expresssion4`* is an example of such a situation.)

After Lingo encounters the first match, it stops testing for additional matches.

If the optional `otherwise` statement is included at the end of the case structure, the statements following `otherwise` are executed if there are no matches.

If a `case` statement tests cases that aren't all integer constants, the Export Xtra for Java converts the `case` statement to an `if...then` statement.

Example

The following handler tests which key the user pressed most recently and responds accordingly.

- If the user pressed A, the movie goes to the frame labeled Apple.
- If the user pressed B or C, the movie performs the specified transition and then goes to the frame labeled Oranges.
- If the user pressed any other key, the computer beeps.

```
on keyDown
  case (the key) of
    "A": go to frame "Apple"
    "B", "C":
      puppetTransition 99
      go to frame "Oranges"
    otherwise beep
  end case
end keyDown
```

Example

This `case` statement tests whether the cursor is over sprite 1, 2, or 3 and runs the corresponding Lingo if it is:

```
case the rollOver of
  1: puppetSound "Horn"
  2: puppetSound "Drum"
  3: puppetSound "Bongos"
end case
```

castLib

Syntax `castLib whichCast`

Description Keyword; indicates that the cast specified in *`whichCast`* is a cast.

The default cast is cast number 1. To specify a cast member in a cast other than cast 1, set `castLib` to specify the alternative cast.

Example This statement displays the number of the Buttons cast in the Message window.

Dot syntax:

```
put castLib("Buttons").number
```

Verbose syntax:

```
put the number of castLib "Buttons"
```

Example This statement assigns cast member 5 in castLib 4 to sprite 10:

```
sprite(10).member = member(5, 4)
```

castLibNum

Syntax

```
member(whichCastMember).castLibNum
the castLibNum of member whichCastMember
sprite(whichSprite).castLibNum
the castLibNum of sprite whichSprite
```

Description

Cast member and sprite property; determines the number of the cast that contains the specified cast member, or which castLib is currently associated with the specified sprite.

If you change `the castLibNum` sprite property without changing `the memberNum` sprite property, Director uses the cast member that has the same cast member number in the new cast. This is useful for movies that you use as templates and update by supplying new casts. If you organize the cast contents so that each cast member has a cast member number that corresponds to its role in the movie, Director automatically inserts the new cast members correctly. To change the cast member assigned to a sprite regardless of its cast, set the `member` sprite property.

For a castmember, this property can be tested but not set. It can be both tested and set for a sprite.

Example

This statement determines the number of the cast to which cast member Jazz is assigned.

Dot syntax:

```
put member("Jazz").castLibNum
```

Verbose syntax:

```
put the castLibNum of member "Jazz"
```

Example

This statement changes the cast member assigned to sprite 5 by switching its cast to Wednesday Schedule.

Dot syntax:

```
sprite(5).castLibNum = castLib("Wednesday Schedule").number
```

Verbose syntax:

```
set the castLibNum of sprite 5 to the number of castLib "Wednesday
Schedule"
```

castMemberList

Syntax

```
member(whichCursorCastMember).castmemberList
the castmemberList of member whichCursorCastMember
```

Description

Cursor cast member property; specifies a list of cast members that make up the frames of a cursor. For *whichCursorCastMember*, substitute a cast member name (within quotation marks) or a cast member number. You can also specify cast members from different casts.

The first cast member in the list is the first frame of the cursor, the second cast member is the second frame, and so on.

If you specify cast members that are invalid for use in a cursor, they will be ignored, and the remaining cast members will be used.

This property can be tested and set.

Example

This command sets a series of four cast members for the animated color cursor cast member named myCursor.

Dot syntax:

```
member("myCursor").castmemberList =  \
[member 1, member 2, member 1 of castlib 2, member 2 of castlib 2]
```

Verbose syntax:

```
set the castmemberList of member "myCursor" =  \
[member 1, member 2, member 1 of castlib 2, member 2 of castlib 2]
```

center

Syntax `member(whichCastMember).center`

`the center of member whichCastMember`

Description Cast member property; interacts with the `crop` cast member property.

- When `the crop` property is `FALSE`, `the center` property has no effect.
- When `crop` is `TRUE` and `center` is `TRUE`, cropping occurs around the center of the digital video cast member.
- When `crop` is `TRUE` and `center` is `FALSE`, the digital video's right and bottom sides are cropped.

This property can be tested and set.

Example This statement causes the digital video cast member Interview to be displayed in the top left corner of the sprite.

Dot syntax:

```
member("Interview").center = FALSE
```

Verbose syntax:

```
set the center of member "Interview" to FALSE
```

See also **crop (cast member property)**, **centerRegPoint**, **regPoint**, **scale**

centerRegPoint

Syntax

```
member(whichCastMember).centerRegPoint
the centerRegPoint of member whichCastMember
```

Description

Flash, vector shape, and bitmap cast member property; automatically centers the registration point of the cast member when you resize the sprite (`TRUE`, default); or repositions the registration point at its current point value when you resize the sprite, set the `defaultRect` property, or set the `regPoint` property (`FALSE`).

This property can be tested and set.

Example

This script checks to see if a Flash movie's `centerRegPoint` property is set to `TRUE`. If it is, the script uses the `regPoint` property to reposition the sprite's registration point to its upper left corner. By checking the `centerRegPoint` property, the script ensures that it does not reposition a registration point that had been previously set using the `regPoint` property.

Dot syntax:

```
on beginSprite me
  if sprite(the spriteNum of me).member.centerRegPoint = TRUE
then
    sprite(the spriteNum of me).member.regPoint = point(0,0)
  end if
end"
```

Verbose syntax:

```
on beginSprite me
  if the centerRegPoint of member the memberNum of me = TRUE then
    set the regPoint of member the memberNum of me = point(0,0)
  end if
end
```

See also

regPoint

centerStage

Syntax `the centerStage`

Description Movie property; determines whether the Stage is centered on the monitor when the movie is loaded (`TRUE`, default) or not centered (`FALSE`). Place the statement that includes this property in the movie that precedes the movie you want it to affect.

This property is useful for checking the Stage location before a movie plays from a projector.

This property can be tested and set.

Note: Be aware that behavior while playing back in a projector differs between Windows and Macintosh systems. Settings selected during creation of the projector may override this property.

Example This statement sends the movie to a specific frame if the Stage is not centered:

```
if the centerStage = FALSE then go to frame "Off Center"
```

Example This statement changes the `centerStage` property to the opposite of its current value:

```
set the centerStage to (not the centerStage)
```

See also **fixStageSize**

changeArea

Syntax

```
member(whichCastMember).changeArea
```

```
the changeArea of member whichCastMember
```

Description

Transition cast member property; determines whether a transition applies only to the changing area on the Stage (`TRUE`) or to the entire Stage (`FALSE`). Its effect is similar to selecting the Changing Area Only option in the Frame Properties Transition dialog box.

This property can be tested and set.

Example

This statement makes the transition cast member Wave apply only to the changing area on the Stage.

Dot syntax:

```
member("Wave").changeArea = TRUE
```

Verbose syntax:

```
set the changeArea of member "Wave" to TRUE
```

channelCount

Syntax

```
member(whichCastMember).channelCount
the channelCount of member whichCastMember
sound(channelNum).channelCount
```

Description

Sound channel and cast member property; for sound channels, determines the number of channels in the currently playing or paused sound in the given sound channel. For sound cast members, determines the number of channels in the specified cast member.

This is useful for determining whether a sound is in monaural or in stereo. This property can be tested but not set.

Example

This statement determines the number of channels in the sound cast member, Jazz.

Dot syntax:

```
put member("Jazz").channelCount
```

Verbose syntax:

```
put the channelCount of member "Jazz"
```

Example

This statement determines the number of channels in the sound member currently playing in sound channel 2:

```
put sound(2).channelCount
```

char...of

Syntax
```
textMemberExpression.char[whichCharacter]
char whichCharacter of fieldOrStringVariable
textMemberExpression.char[firstCharacter..lastCharac ter]
char firstCharacter to lastCharacter of fieldOrStringVariable
```

Description Keyword; identifies a character or a range of characters in a chunk expression. A chunk expression is any character, word, item, or line in any source of text (such as field cast members and variables) that holds a string.

- An expression using *whichCharacter* identifies a specific character.
- An expression using *firstCharacter* and *lastCharacter* identifies a range of characters.

The expressions must be integers that specify a character or range of characters in the chunk. Characters include letters, numbers, punctuation marks, spaces, and control characters such as Tab and Return.

You can test but not set the `char...of` keyword. Use the `put...into` command to modify the characters in a string.

To see an example of `char...of` used in a completed movie, see the Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement displays the first character of the string $9.00:

```
put ("$9.00").char[1..1]
-- "$"
```

Example This statement displays the entire string $9.00:

```
put ("$9.00").char[1..5]
-- "$9.00"
```

Example This statement changes the first five characters of the second word in the third line of a text cast member:

```
member("quiz").line[3].word[2].char[1..5] = "?????"
```

See also **mouseMember**, **mouseItem**, **mouseLine**, **mouseWord**

characterSet

Syntax
```
member(whichFontMember).characterSet
the characterSet of member whichFontMember
```

Description Font cast member property; returns a string containing the characters included for import when the cast member was created. If all characters in the original font were included, the result is an empty string.

Example This statement displays the characters included when cast member 11 was created. The characters included during import were numerals and Roman characters.

```
put member(11).characterSet
--
"1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
```

See also **recordFont**, **bitmapSizes**, **originalFont**

charPosToLoc()

Syntax

```
member(whichCastMember).charPosToLoc(nthCharacter)
charPosToLoc(member whichCastMember, nthCharacter)
```

Description

Field function; returns the point in the entire field cast member (not just the part that appears on the Stage) that is closest to the character specified by *nthCharacter*. This is useful for determining the location of individual characters.

Values for `charPosToLoc` are in pixels from the top left corner of the field cast member. The *nthCharacter* parameter is 1 for the first character in the field, 2 for the second character, and so on.

Example

The following statement determines the point where the fiftieth character in the field cast member Headline appears and assigns the result to the variable location:

```
location = charPosToLoc(member "Headline", 50)
```

chars()

Syntax `chars(stringExpression, firstCharacter, lastCharacter)`

Description Function; identifies a substring of characters in *`stringExpression`*. The substring starts at *`firstCharacter`* and ends at *`lastCharacter`*. The expressions *`firstCharacter`* and *`lastCharacter`* must specify a position in the string.

If *`firstCharacter`* and *`lastCharacter`* are equal, then a single character is returned from the string. If *`lastCharacter`* is greater than the string length, only a substring up to the length of the string is identified. If *`lastCharacter`* is before *`firstCharacter`*, the function returns the value `EMPTY`.

Example This statement identifies the sixth character in the word *Macromedia*:

```
put chars("Macromedia", 6, 6)
-- "m"
```

Example This statement identifies the sixth through tenth characters of the word *Macromedia*:

```
put chars("Macromedia", 6, 10)
-- "media"
```

Example This statement tries to identify the sixth through twentieth characters of the word Macromedia. Because the word has only 10 characters, the result includes only the sixth through tenth characters.

```
put chars ("Macromedia", 6, 20)
-- "media"
```

See also **char...of**; **length()**and **offset() (string function)**; **number (characters)**

charSpacing

Syntax `chunkExpression.charSpacing`

Description Text cast member property; enables specifying any additional spacing applied to each letter in the *`chunkExpression`* portion of the text cast member.

A value less than 0 indicates less spacing between letters. A value greater than 0 indicates more spacing between letters.

The default value is 0, which results in default spacing between letters.

Example The following handler increases the current character spacing of the third through fifth words within the text cast member myCaption by a value of 2:

```
on myCharSpacer
  mySpaceValue = member("myCaption").word[3..5].charSpacing
  member("myCaption").word[3..5].charSpacing = (mySpaceValue + 2)
end
```

charToNum()

Syntax

```
(stringExpression).charToNum
```

```
charToNum(stringExpression)
```

Description

Function; returns the ASCII code that corresponds to the first character of *stringExpression*.

The `charToNum()` function is especially useful for testing the ASCII value of characters created by combining keys, such as the Control key and another alphanumeric key.

Director treats uppercase and lowercase letters the same if you compare them using the equal sign (=) operator; for example, the statement `put ("M" = "m")` returns the result 1 or `TRUE`.

Avoid problems by using `charToNum()` to return the ASCII code for a character and then use the ASCII code to refer to the character.

Example

This statement displays the ASCII code for the letter A:

```
put ("A").charToNum
-- 65
```

Example

The following comparison determines whether the letter entered is a capital A, and then navigates to either a correct sequence or incorrect sequence in the Score:

```
on CheckKeyHit theKey
  if (theKey).charToNum = 65 then
    go "Correct Answer"
  else
    go "Wrong Answer"
  end if
end
```

See also

numToChar()

checkBoxAccess

Syntax `the checkBoxAccess`

Description Movie property; specifies one of three possible results when the user clicks a check box or radio button created with button tools in the Tools window:

- `0` (default)—Lets the user set check boxes and radio buttons on and off.
- `1`—Lets the user set check boxes and radio buttons on but not off.
- `2`—Prevents the user from setting check boxes and radio buttons at all; the buttons can be set only by scripts.

This property can be tested and set.

Example This statement sets the `checkBoxAccess` property to 1, which lets the user click check boxes and radio buttons on but not off:

```
the checkBoxAccess to 1
```

Example This statement records the current setting of the `checkBoxAccess` property by putting the value in the variable `oldAccess`:

```
oldAccess to the checkBoxAccess
```

See also **hilite (cast member property)**, **checkBoxType**

checkBoxType

Syntax `the checkBoxType`

Description Movie property; specifies one of three ways to indicate whether a check box is selected:

- `0` (default)—Creates a standard check box that contains an X when the check box is selected.
- `1`—Creates a check box that contains a black rectangle when the check box is selected.
- `2`—Creates a check box that contains a filled black rectangle when the check box is selected.

This property can be tested and set.

Example This statement sets the `checkBoxType` property to 1, which creates a black rectangle in check boxes when the user clicks them:

```
the checkBoxType to 1
```

See also **hilite (cast member property)**, **checkBoxAccess**

checkMark

Syntax `the checkMark of menuItem whichItem of menu whichMenu`

Description Menu item property; determines whether a check mark appears next to the custom menu item (`TRUE`) or not (`FALSE`, default).

The *`whichItem`* value can be either a menu item name or a menu item number. The *`whichMenu`* value can be either a menu name or a menu number.

This property can be tested and set.

Note: Menus are not available in Shockwave

Example This handler turns off any items that are checked in the custom menu specified by the argument `theMenu`. For example, `unCheck ("Format")` turns off all the items in the Format menu.

```
on unCheck theMenu
  set n = the number of menuItems of menu theMenu
  repeat with i = 1 to n
    set the checkMark of menuItem i of menu theMenu to FALSE
  end repeat
end unCheck
```

See also **installMenu**; **enabled**, **name (menu item property)**, **number (menu items)**, and **script**; **name (menu item property)** and **number (menu items)**; **menu**

chunkSize

Syntax

```
member(whichCastMember).chunkSize
the chunkSize of member whichCastMember
```

Description

Transition cast member property; determines the transition's chunk size in pixels from 1 to 128 and is equivalent to setting the smoothness slider in the Frame Properties: Transition dialog box. The smaller the chunk size, the smoother the transition appears.

This property can be tested and set.

Example

This statement sets the chunk size of the transition cast member Fog to 4 pixels.

Dot syntax:

```
member("Fog").chunkSize = 4
```

Verbose syntax:

```
set the chunkSize of member "Fog" to 4
```

clearCache

Syntax `clearCache`

Description Command; clears Director's network cache.

The `clearCache` command clears only Director's cache, which is separate from the browser's cache.

If a file is in use, it remains in the cache until it is no longer in use.

Example This handler clears the cache when the movie starts:

```
on startMovie
  clearCache
end
```

See also **cacheDocVerify()**,**cacheSize()**

clearError

Syntax

```
member(whichFlashMember).clearError()
clearError (member whichFlashMember)
```

Description

Flash command; resets the error state of a streaming Flash cast member to 0.

When an error occurs while a cast member is streaming into memory, Director sets the cast member's `state` property to -1 to indicate that an error occurred. When this happens, you can use the `getError` function to determine what type of error occurred and then use the `clearError` command to reset the cast member's error state to 0. After you clear the member's error state, Director tries to open the cast member if it is needed again in the Director movie. Setting a cast member's `pathName`, `linked`, and `preload` properties also automatically clears the error condition.

Example

This handler checks to see if an out-of-memory error occurred for a Flash cast member named Dali, which was streaming into memory. If a memory error occurred, the script uses the `unloadCast` command to try to free some memory; it then branches the playback head to a frame in the Director movie named Artists, where the Flash movie sprite first appears, so Director can again try to play the Flash movie. If something other than an out-of-memory error occurred, the script goes to a frame named Sorry, which explains that the requested Flash movie can't be played.

```
on CheckFlashStatus
  if member("Dali").getError() = #memory then
    member("Dali").clearError()
    unloadCast
    go to frame "Artists"
  else
    go to frame "Sorry"
  end if
end
```

See also

state, **getError()**

clearFrame

Syntax `clearFrame`

Description Command; erases everything in the current frame's sprite and affects channels during Score recording only.

Example The following handler clears the content of each frame before it edits that frame during Score generation:

```
on newScore
  beginRecording
  repeat with counter = 1 to 50
    clearFrame
    the frameScript to 25
    updateFrame
  end repeat
  endRecording
end
```

See also **beginRecording** and **endRecording**; **updateFrame**

clearGlobals

Syntax `clearGlobals`

Description Command; sets all global variables to `VOID`.

This command can be useful when initializing global variables or when opening a new movie that requires a new set of global variables.

Example This statement sets all global variables to `VOID`:

```
clearGlobals
```

clickLoc

Syntax `the clickLoc`

Description Function; identifies as a point the last place on the screen where the mouse was clicked.

Example The following `on mouseDown` handler displays the last mouse click location:

```
on mouseDown
  put the clickLoc
end mouseDown
```

If the click were 50 pixels from the left end of the Stage and 100 pixels from the top of the Stage, the Message window would display the following:

```
-- point(50, 100)
```

clickMode

Syntax

```
sprite(whichFlashSprite).clickMode
the clickMode of sprite whichFlashSprite
member(whichFlashMember).clickMode
the clickMode of member whichFlashMember
```

Description

Flash cast member and sprite property; controls when the Flash movie sprite detects mouse click events (mouseUp and mouseDown) and when it detects rollovers (mouseEnter, mouseWithin, and mouseLeave). The `clickMode` property can have these values:

- `#boundingBox`—Detects mouse click events anywhere within the sprite's bounding rectangle and detects rollovers at the sprite's boundaries.
- `#opaque` (default)—Detects mouse click events only when the pointer is over an opaque portion of the sprite and detects rollovers at the boundaries of the opaque portions of the sprite if the sprite's ink effect is set to Background Transparent. If the sprite's ink effect is not set to Background Transparent, this setting has the same effect as `#boundingBox`.
- `#object`—Detects mouse click events when the mouse pointer is over any filled (nonbackground) area of the sprite and detects rollovers at the boundaries of any filled area. This setting works regardless of the sprite's ink effect.

This property can be tested and set.

Example

This script checks to see if the sprite, which is specified with an ink effect of Background Transparent, is currently set to be rendered direct to Stage. If the sprite is not rendered direct to Stage, the sprite's `clickMode` is set to `#opaque`. Otherwise (because ink effects are ignored for Flash movie sprites that are rendered direct to Stage), the sprite's `clickMode` is set to `#boundingBox`.

Dot syntax:

```
on beginSprite me
  if sprite(the spriteNum of me).directToStage = FALSE then
    sprite(the spriteNum of me).clickMode = #opaque
  else
    sprite(the spriteNum of me).clickMode = #boundingBox
  end if
end
```

Verbose syntax:

```
on beginSprite me
  if the directToStage of sprite the spriteNum of me = FALSE then
    set the clickMode of sprite the spriteNum of me = #opaque
  else
    set the clickMode of sprite the spriteNum of me =
#boundingBox
  end if
end
```

clickOn

Syntax `the clickOn`

Description Function; returns the last active sprite clicked by the user. An active sprite is a sprite that has a sprite or cast member script associated with it.

When the user clicks the Stage, `clickOn` returns 0. To detect whether the user clicks a sprite with no script, you must assign a placeholder script to it (`"- -,"` for example) so that it can be detected by the `clickOn` function.

The `clickOn` can be checked within a repeat loop. However, neither `clickOn` nor `clickLoc` functions change value when the handler is running. The value that you obtain is the value from before the handler started.

Example This statement checks whether sprite 7 was the last active sprite clicked:

```
if the clickOn = 7 then alert "Sorry, try again."
```

Example This statement sets the `foreColor` property of the last active sprite that was clicked to a random color:

```
sprite(the clickOn).foreColor = random(255)-1
```

See also **doubleClick**, **the mouseDown (system property)**, **mouseMember**, **the mouseUp (system property)**

closed

Syntax `member(whichCastMember).closed`

Description Vector shape cast member property; indicates whether the end points of a path are closed or open.

Vector shapes must be closed in order to contain a fill.

The value can be:

- `TRUE`—the end points are closed.
- `FALSE`—the end points are open.

close window

Syntax `window(windowIdentifier).close()`

close window windowIdentifier

Description Window command; closes the window specified by *windowIdentifier*.

- To specify a window by name, use the syntax `close window` *name*, where you replace *name* with the name of a window. Use the complete pathname.
- To specify a window by its number in `windowList`, use the syntax `close window` *number*, where you replace *number* with the window's number in `windowList`.

Closing a window that is already closed has no effect.

Be aware that closing a window does not stop the movie in the window nor clear it from memory. This command simply closes the window in which the movie is playing. You can reopen it quickly by using the `open window` command. This allows rapid access to windows that you want to keep available.

If you want to completely dispose of a window and clear it from memory, use the `forget window` command. Make sure that nothing refers to the movie in that window if you use the `forget window` command, or you will generate errors when scripts try to communicate or interact with the forgotten window.

Example This statement closes the window named Panel, which is in the subfolder MIAW Sources within the current movie's folder:

```
window("@/MIAW Sources/Panel").close()
```

Example This statement closes the window that is number 5 in `windowList`:

```
window(5).close()
```

See also **forget window** and **open window**; **windowList**

on closeWindow

Syntax

```
on closeWindow
  statement(s)
end
```

Description

System message and event handler; contains statements that run when the user closes the window for a movie by clicking the window's close box.

The `on closeWindow` handler is a good place to put Lingo commands that you want executed every time the movie's window closes.

Example

This handler tells Director to `forget` the current window when the user closes the window that the movie is playing in:

```
on closeWindow
  -- perform general housekeeping here
  forget the activeWindow
end
```

closeXlib

Syntax `closeXlib *whichFile*`

Description Command; closes the Xlibrary file specified by the string `*whichFile*`. If the Xlibrary file is in a folder other than that for the current movie, `*whichFile*` must specify a pathname. If no file is specified, all open Xlibraries are closed.

Xtras are stored in Xlibrary files. Xlibrary files are resource files that contain Xtras. HyperCard XCMDs and XFCNs can also be stored in Xlibrary files.

The `closeXlib` command doesn't work for URLs.

In Windows, using the DLL extension for Xtras is optional.

It is good practice to close any file you have opened as soon as you have finished using it.

Note: This command is not supported in Shockwave.

Example This statement closes all open Xlibrary files:

```
closeXlib
```

Example This statement closes the Xlibrary Video Disc Xlibrary when it is in the same folder as the movie:

```
closeXlib "Video Disc Xlibrary"
```

Example This statement closes the Xlibrary Transporter Xtras in the folder New Xtras, which is in the same folder as the movie. The disk is identified by the variable `*currentDrive*`:

```
closeXlib "@:New Xtras:Transporter Xtras"
```

See also **interface()**,**openXlib**, **showXlib**

color()

Syntax

```
color(#rgb, redValue, greenValue, blueValue)
color(#paletteIndex, paletteIndexNumber)
rgb(rgbHexString)
rgb(redValue, greenValue, blueValue)
paletteIndex(paletteIndexNumber)
```

Description

Function and data type; determines an object's color as either RGB or 8-bit palette index values. These are the same values as those used in the `color` member and `color` sprite properties, the `bgColor` member and `bgColor` sprite properties, and the `bgColor` Stage property.

The `color` function allows for either 24-bit or 8-bit color values to be manipulated as well as applied to cast members, sprites, and the Stage.

For RGB values, each color component has a range from 0 to 255, and all other values are truncated. For `paletteIndex` types, an integer from 0 to 255 is used to indicate the index number in the current palette, and all other values are truncated.

Example

This statement performs a math operation:

```
palColorObj = paletteIndex(20)
put palColorObj
-- paletteIndex(20)
put palColorObj / 2
-- paletteIndex(10)
```

Example

This statement converts one color type to another type:

```
newColorObj = color(#rgb, 155, 0, 75)
put newColorObj
-- rgb(155, 0, 75)
newColorObj.colorType = #paletteIndex
put newColorObj
-- paletteIndex(106)
```

Example

This statement obtains the hexadecimal representation of a color regardless of its type:

```
someColorObj = color(#paletteIndex, 32)
put someColorObj.hexString()
-- "#FF0099"
```

Example

This statement determines individual RGB components and the `paletteIndex` value of a color regardless of its type:

```
newColorObj = color(#rgb, 155, 0, 75)
put newColorObj.green
-- 0
put newColorObj.paletteIndex
-- 106
newColorObj.green = 100
put newColorObj.paletteIndex
-- 94
put newColorObj
-- rgb(155, 100, 75)
newColorObj.paletteIndex = 45
put newColorObj
```

```
-- paletteIndex(45)
```

Example This statement changes the color of the fourth through the seventh characters of text member myQuotes:

```
member("myQuotes").char[4..7].color = rgb(200, 150, 75)
```

Example This Lingo displays the color of sprite 6 in the Message window, and then sets the color of sprite 6 to a new RGB value:

```
"put sprite(6).color
-- rgb( 255, 204, 102 )
sprite(6).color = rgb(122, 98, 210)"
```

Note: Setting the `paletteIndex` value of an RGB color type changes `colorType` to `paletteIndex`. Setting the RGB color type of a `paletteIndex` color sets its `colorType` value to RGB.

See also **bgColor**

color (sprite property)

Syntax
```
sprite(whichSpriteNumber).color
the color of sprite whichSpriteNumber
```

Description
Sprite property; determines the foreground color of the sprite specified by *whichSprite*. Setting the `foreColor` sprite property is equivalent to choosing the foreground color from the Tools window when the sprite is selected on the Stage.

This property has the equivalent functionality of the `foreColor` sprite property, but the color value returned is a color object of whatever type has been set for that sprite.

This property can be tested and set.

See also **color()**,**bgColor**, **foreColor**

colorDepth

Syntax `the colorDepth`

Description System property; determines the color depth of the computer's monitor.

- In Windows, using this property lets you check and set the monitor's color depth. Some video card and driver combinations may not enable you to set the `colorDepth` property. Always verify that the color depth has actually changed after you attempt to set it.
- On the Macintosh, this property lets you check the color depth of different monitors and change it when appropriate.

Possible values are the following:

1	Black and white
2	4 colors
4	16 colors
8	256 colors
16	32,768 or 65,536 colors
32	16,777,216 colors

If you try to set a monitor's color depth to a value that monitor does not support, the monitor's color depth doesn't change.

On computers with more than one monitor, the `colorDepth` property refers to the monitor displaying the Stage. If the Stage spans more than one monitor, the `colorDepth` property indicates the greatest depth of those monitors; `colorDepth` tries to set all those monitors to the specified depth.

This property can be tested and set.

Example This statement tells Director to play the segment Full color only if the monitor color depth is set to 256 colors:

```
if the colorDepth = 8 then play movie "Full color"
```

Example The following handler tries to change the color depth, and if it can't, it displays an alert:

```
on TryToSetColorDepth desiredDepth
  the colorDepth = desiredDepth
  if the colorDepth = desiredDepth then
    return true
  else
    alert "Please change your system to" && desiredDepth &&"color
depth and reboot."
    return false
  end if
end
```

When changing the user's monitor color depth settings, it is good practice to restore the original depth when the movie has finished. In Windows, the command `set the colorDepth = 0` restores the user's preferred settings from the control panel.

See also **switchColorDepth**

commandDown

Syntax `the commandDown`

Description Function; determines whether the Control key (Windows) or the Command key (Macintosh) is being pressed (`TRUE`) or not (`FALSE`).

You can use `commandDown` together with the element `the key` to determine when the Control or Command key is pressed in combination with another key. This lets you create handlers that are executed when the user presses specified Control or Command key combinations.

Control or Command key equivalents for Director's authoring menus take precedence while the movie is playing, unless you have installed custom Lingo menus or are playing a projector version of the movie.

For a movie playing back with the Director player for Java, this function returns `TRUE` only if a second key is pressed simultaneously with the Control or Command key. If the Control or Command key is pressed by itself, `commandDown` returns `FALSE`. This is because the browser receives the keys before the movie and thus responds to and intercepts any key combinations that are also browser keyboard shortcuts. For example, if the user presses Control+R or Command+R, the browser reloads the current page; the movie never receives the key combination.

For a demonstration of modifier keys in Lingo, see the sample movie "Keyboard Lingo" in Director Help.

Example These statements pause a projector whenever the user presses Control+ or Command+A. By setting the `keyDownScript` property to `doCommandKey`, the `on prepareMovie` handler makes the `doCommandKey` handler the first event handler executed when a key is pressed. The `doCommandKey` handler checks whether the Control+A or Command+A keys are pressed at the same time and pauses the movie if they are.

```
on prepareMovie
  the keyDownScript = "doCommandKey"
end

on doCommandKey
  if (the commandDown) and (the key = "a") then go to the frame
end
```

See also **controlDown**, **key()**,**keyCode()**,**optionDown**, **shiftDown**

comments

Syntax `member.comments`

the comments of *member*

Description This cast member property provides a place to store any comments you want to maintain about the given cast member, or any other strings you want to associate with the member. This property can be tested and set. It can also be set in the Property Inspector's Member tab.

Example This statement sets the comments of the member Backdrop to the string "Still need to license this artwork":

```
member("Backdrop").comments = "Still need to license this artwork"
```

See also **creationDate**, **modifiedBy**, **modifiedDate**

constrainH()

Syntax `constrainH (whichSprite, integerExpression)`

Description Function; evaluates *`integerExpression`* and then returns a value that depends on the horizontal coordinates of the left and right edges of *`whichSprite`*, as follows:

- When the value is between the left and right coordinates, the value doesn't change.
- When the value is less than the left horizontal coordinate, the value changes to the value of the left coordinate.
- When the value is greater than the right horizontal coordinate, the value changes to the value of the right coordinate.

The `constrainH` and `constrainV` functions constrain only one axis each; the `constraint` sprite property limits both. Note that this function does not change the sprite's properties.

Example These statements check the `constrainH` function for sprite 1 when it has left and right coordinates of 40 and 60:

```
put constrainH(1, 20)
-- 40

put constrainH(1, 55)
-- 55

put constrainH(1, 100)
-- 60
```

Example This statement constrains a moveable slider (sprite 1) to the edges of a gauge (sprite 2) when the mouse pointer goes past the edge of the gauge:

```
set the locH of sprite 1 to constrainH(2, the mouseH)
```

See also **constrainV()**,**constraint**, **left**, **right**

constraint

Syntax
```
sprite(whichSprite).constraint
the constraint of sprite whichSprite
```

Description Sprite property; determines whether the registration point of the sprite specified by *whichSprite* is constrained to the bounding rectangle of another sprite (1 or TRUE) or not (0 or FALSE, default).

The constraint sprite property is useful for constraining a moveable sprite to the bounding rectangle of another sprite to simulate a track for a slider control or to restrict where on the screen a user can drag an object in a game.

The constraint sprite property affects moveable sprites and the locH and locV sprite properties. The constraint point of a moveable sprite cannot be moved outside the bounding rectangle of the constraining sprite. (The constraint point for a bitmap sprite is the registration point. The constraint point for a shape sprite is its top left corner.) When a sprite has a constraint set, the constraint limits override any locH and locV sprite property settings.

This property can be tested and set.

Example This statement removes a constraint sprite property:

Dot syntax:

```
sprite(whichSprite).constraint = 0
```

Verbose syntax:

```
set the constraint of sprite whichSprite to 0
```

Example This statement constrains sprite (i + 1) to the boundary of sprite 14:

```
sprite(i + 1).constraint = 14
```

Example This statement checks whether sprite 3 is constrained and activates the handler showConstraint if it is: (The operator <> performs a not-equal-to operation.)

```
if sprite(3).constraint <> 0 then showConstraint
```

See also **constrainH()**,**constrainV()**;**locH**, **locV**

constrainV()

Syntax `constrainV (whichSprite, integerExpression)`

Description Function; evaluates *`integerExpression`* and then returns a value that depends on the vertical coordinates of the top and bottom edges of the sprite specified by *`whichSprite`*, as follows:

- When the value is between the top and bottom coordinates, the value doesn't change.
- When the value is less than the top coordinate, the value changes to the value of the top coordinate.
- When the value is greater than the bottom coordinate, the value changes to the value of the bottom coordinate.

This function does not change the sprite properties.

Example These statements check the `constrainV` function for sprite 1 when it has top and bottom coordinates of 40 and 60:

```
put constrainV(1, 20)
-- 40

put constrainV(1, 55)
-- 55

put constrainV(1, 100)
-- 60
```

Example This statement constrains a moveable slider (sprite 1) to the edges of a gauge (sprite 2) when the mouse pointer moves past the edge of the gauge:

```
set the locV of sprite 1 to constrainV(2, the mouseV)
```

See also **bottom**, **constraint**, **top**; **constrainH()**

contains

Syntax `stringExpression1 contains stringExpression2`

Description Operator; compares two strings and determines whether *`stringExpression1`* contains *`stringExpression2`* (`TRUE`) or not (`FALSE`).

The `contains` comparison operator has a precedence level of 1.

The `contains` comparison operator is useful for checking whether the user types a specific character or string of characters. You can also use the `contains` operator to search one or more fields for specific strings of characters.

Example This example determines whether a character passed to it is a digit:

```
on isNumber aLetter
  digits = "1234567890"
  if digits contains aLetter then
    return TRUE
  else
    return FALSE
  end if
end
```

Note: The string comparison is not sensitive to case or diacritical marks; “a” and Å are treated the same.

See also **offset() (string function)**, **starts**

continue

This is obsolete. Use `go to the frame +1`.

controlDown

Syntax `the controlDown`

Description Function; determines whether the Control key is being pressed (`TRUE`) or not (`FALSE`).

You can use the `controlDown` function together with `the key` to check for combinations of the Control key and another key.

For a movie playing back with the Director player for Java, this function returns `TRUE` only if a second key is pressed simultaneously with the Control key. If the Control key is pressed by itself, `controlDown` returns `FALSE`. The Director player for Java supports key combinations with the Control key. However, the browser receives the keys before the movie and thus responds to and intercepts any key combinations that are also browser keyboard shortcuts.

For a demonstration of modifier keys and Lingo, see the sample movie Keyboard Lingo in Director Help.

Example This `on keyDown` handler checks whether the pressed key is the Control key, and if it is, the handler activates the `on doControlKey` handler. The argument `(the key)` identifies which key was pressed in addition to the Control key.

```
on keyDown
  if (the controlDown)then doControlKey (the key)
end
```

See also **charToNum()**,**commandDown**, **key()**,**keyCode()**,**optionDown**, **shiftDown**

controller

Syntax `member(whichCastMember).controller`

`the controller of member whichCastMember`

Description Digital video cast member property; determines whether a digital video movie cast member shows or hides its controller. Setting this property to 1 shows the controller; setting it to 0 hides the controller.

The `controller` member property applies to a QuickTime digital video only.

- Setting `the controller` member property for a Video for Windows digital video performs no operation and generates no error message.
- Checking the `controller` member property for a Video for Windows digital video always returns `FALSE`.

The digital video must be in direct-to-stage playback mode to display the controller.

Example This statement causes the QuickTime cast member Demo to display its controller.

Dot syntax:

```
member("Demo").controller = 1
```

Verbose syntax:

```
set the controller of member "Demo" to 1
```

See also **directToStage**

copyPixels()

Syntax `imageObject.copyPixels(sourceImageObject, destinationRect, sourceRect {, parameterList})`

imageObject.copyPixels(sourceImageObject, destinationQuad, sourceRect {, parameterList })

Description This function copies the contents of the *sourceRect* from the *sourceImageObject* into the given *imageObject*. The pixels are copied from the *sourceRect* in the *sourceImageObject* and placed into the *destinationRect* or *destinationQuad* in the given *imageObject*. See `quad` for information on using quads.

You can include an optional property list of parameters in order to manipulate the pixels being copied before they are placed into the *destinationRect*. The property list may contain any or all of the following parameters:

Property	Use and Effect
#color	The foreground color to apply for colorization effects. The default color is black.
#bgColor	The background color to apply for colorization effects or background transparency. The default background color is white.
#ink	The type of ink to apply to the copied pixels. This can be an ink symbol or the corresponding numeric ink value. The default ink is #copy. See **ink** for the list of possible values.
#blendLevel	The degree of blend (transparency) to apply to the copied pixels. The range of values is from 0 to 255. The devault value is 255 (opaque). Using a value less than 255 forces the #ink setting to be #blend, or #blendTransparent if it was originally #backgroundTransparent. You may also substitute #blend as the property name and use a value range of 0 to 100. See **blend** for more information.
#dither	A TRUE or FALSE value that determines whether the copied pixels will be dithered when placed into the *destinationRect* in 8- and 16-bit images. The default value is FALSE, which maps the copied pixels directly into the *imageObject*'s color palette.
#useFastQuads	A TRUE or FALSE value that determines whether quad calculations are made using Director's faster but less precise method when copying pixels into a *destinationQuad*. Set this to TRUE if you are using quads for simple rotation and skew operations. Leave it FALSE (the default value) for arbitrary quads, such as those used for perspective transformations. See **useFastQuads**.
#maskImage	Used to specified a mask or matte object created with the `createMask()` or `createMatte()` functions to be used as a mask for the pixels being copied. This allows you to duplicate the effects of mask and matte

sprite inks. Note that if the source image has an alpha channel and its `useAlpha` property is TRUE, the alpha channel is used and the specified mask or matte is ignored. The default is no mask.

#maskOffset	A point indicating the amount of x and y offset to apply to the mask specified by #maskImage. The offset is relative to the upper left corner of the *`sourceImage`*. The default offset is (0, 0).

When copying pixels from one area of a cast member to another area of the same member, it is best to copy the pixels first into a duplicate image object before copying them back into the original member. Copying directly from one area to another in the same image is not recommended. See `duplicate()`.

To simulate matte ink with `copyPixels()`, create a matte object with `createMatte()` and then pass that object as the `#maskImage` parameter with `copyPixels()`.

Copying pixels from an image object into itself is not recommended. Use separate image objects instead.

Example This statement copies the entire image of member Happy into the rectangle of member flower. If the members are different sizes, the image of member Happy will be resized to fit the rectangle of member flower.

```
member("flower").image.copyPixels(member("Happy").image,  \
member("flower").rect, member("Happy").rect)
```

Example This statement copies part of the image of member Happy into part of member flower. The part of the image copied from Happy is within rectangle(0, 0, 200, 90). It is pasted into rectangle(20, 20, 100, 40) within the image of member flower. The copied portion of Happy is resized to fit the rectangle into which it is pasted.

```
member("flower").image.copyPixels(member("Happy").image,\
rect(20, 20, 100, 40), rect(0, 0, 200, 90))
```

Example This statement copies the entire image of member Happy into a rectangle within the image of member flower. The rectangle into which the copied image of member Happy is pasted is the same size as the rectangle of member Happy, so the copied image is not resized. The blend level of the copied image is 50, so it is semi-transparent, revealing the part of member flower it is pasted over.

```
member("flower").image.copyPixels(member("Happy").image,\
member("Happy").rect, member("Happy").rect, [#blendLevel: 50])
```

See also **ink**, **color()**

copyrightInfo

Syntax

```
member(whichCastMember).copyrightInfo
copyrightInfo of member whichCastMember
```

Description

Shockwave Audio (SWA) cast member property; displays the copyright text in a SWA file. This property is available only after the SWA sound begins playing or after the file has been preloaded using the `preLoadBuffer` command.

This property can be tested and set.

Example

This statement tells Director to display the copyright information for the Shockwave Audio file SWAfile in a field cast member named Info Display.

Dot syntax:

```
set whatState = the state of member "SWAfile"
if whatState > 1 AND whatState < 9 then
  put member("SWAfile").copyrightInfo into member("Info Display")
end if
```

Verbose syntax:

```
set whatState = the state of member "SWAfile"
if whatState > 1 AND whatState < 9 then
  put the copyrightInfo of member "SWAfile" into member "Info
Display"
end if
```

copyToClipBoard

Syntax `member(whichCastMember).copyToClipBoard()`

`copyToClipBoard member whichCastMember`

Description Command; copies the specified cast member to the Clipboard without requiring that the cast window is active. You can use this command to copy cast members between movies or applications.

Example This statement copies the cast member named chair to the Clipboard:

```
member("chair").copyToClipboard()
```

Example This statement copies cast member number 5 to the Clipboard:

```
member(5).copyToClipboard()
```

See also **pasteClipBoardInto**

cos()

Syntax
```
(angle).cos
cos (angle)
```

Description Function; calculates the cosine of the specified angle, which must be expressed in radians.

Example The following statement calculates the cosine of `PI` divided by 2 and displays it in the Message window:

```
put (PI/2).cos
```

See also **atan()**, **PI**, **sin()**

count()

Syntax

```
list.count
```

count (*list*)

```
count(theObject)
```

```
object.count
```

```
textExpression.count
```

Description

Function; returns the number of entries in a linear or property list, the number of properties in a parent script without counting the properties in an ancestor script, or the chunks of a text expression such as characters, lines, or words.

The `count` command works with linear and property lists, objects created with parent scripts, and `the globals` property.

To see an example of `count` used in a completed movie, see the Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement displays the number 3, the number of entries:

```
put [10,20,30].count
-- 3
```

See also

globals

cpuHogTicks

Syntax `the cpuHogTicks`

Description System property; determines how often Director releases control of the CPU to let the computer process background events, such as events in other applications, network events, clock updates, and other keyboard events.

The default value is 20 ticks. To give more time to Director before releasing the CPU to background events or to control how the computer responds to network operations, set `cpuHogTicks` to a higher value.

To create faster auto-repeating key performance but slower animation, set `cpuHogTicks` to a lower value. In a movie, when a user holds down a key to generate a rapid sequence of auto-repeating key presses, Director typically checks for auto-repeating key presses less frequently than the rate set in the computer's control panel.

The `cpuHogTicks` property works only on the Macintosh.

Example This statement tells Director to release control of the CPU every 6 ticks, or every 0.10 of a second:

```
the cpuHogTicks = 6
```

See also **ticks**

createMask()

Syntax *imageObject*.createMask()

Description This function creates and returns a mask object for use with the copyPixels() function.

Mask objects aren't image objects; they're useful only with the copyPixels() function for duplicating the effect of mask sprite ink. To save time, if you plan to use the same image as a mask more than once, it's best to create the mask object and save it in a variable for reuse.

Example This statement copies the entire image of member Happy into a rectangle within the image of member brown square. Member gradient2 is used as a mask with the copied image. The mask is offset by 10 pixels up and to the left of the the rectangle into which the image of member Happy is pasted.

```
member("brown square").image.copyPixels(member("Happy").image, \
rect(20, 20, 150, 108), member("Happy").rect, \
[#maskImage:member("gradient2").image.createMask(),
maskOffset:point(-10, -10)])
```

See also **copyPixels()**,**createMatte()**,**ink**

createMatte()

Syntax `imageObject.createMatte({alphaThreshold})`

Description This function creates and returns a matte object that you can use with `copyPixels()` to duplicate the effect of the matte sprite ink. The matte object is created from the specified image object's alpha layer. The optional parameter *`alphaThreshold`* excludes from the matte all pixels whose alpha channel value is below that threshold. It is used only with 32-bit images that have an alpha channel. The *`alphaThreshold`* must be a value between 0 and 255.

Matte objects aren't image objects; they are useful only with the `copyPixels()` function. To save time, if you plan to use the same image as a matte more than once, it's best to create the matte and save it in a variable for reuse.

Example This statement creates a new matte object from the alpha layer of the image object testImage and ignores pixels with alpha values below 50%:

```
newMatte = testImage.createMatte(128)
```

See also **copyPixels()**,**createMask()**

creationDate

Syntax *member*.creationDate

the creationDate of *member*

Description This cast member property records the date and time that the cast member was first created, using the system time on the computer. You can use this property to schedule a project; Director doesn't use it for anything.

This property can be tested and set.

Example Although you typically inspect the creationDate property using the Property Inspector or the Cast window list view, you can check it in the Message window:

```
put member(1).creationDate
  -- date( 1999, 12, 8 )
```

See also **comments**, **modifiedBy**, **modifiedDate**

crop() (image object command)

Syntax `imageObject.crop(rectToCropTo)`

Description When used with an image object, returns a new image object that contains a copy of the given image object, cropped to the given rect. The original image object is unchanged. The new image object does not belong to any cast member and has no association with the Stage. If you wish to assign it to a cast member you can do so by setting the `image` property of that cast member.

This is different from using `crop` with a cast member, which crops the cast member itself, altering the original.

Example This Lingo takes a snapshot of the Stage and crops it to the `rect` of sprite 10, capturing the current appearance of that sprite on the Stage:

```
set stageImage = (the stage).image
set spriteImage = stageImage.crop(sprite(10).rect)
member("sprite snapshot").image = spriteImage
```

Example This statement uses the rectangle of cast member Happy to crop the image of cast member Flower, then sets the image of cast member Happy to the result.

```
member("Happy").image =
member("Flower").image.crop(member("Happy").rect)
```

crop() (member command)

Syntax

```
member(whichMember).crop(rectToCropTo)
```

```
crop member whichMember, rectToCropTo
```

Description

Bitmap command; allows a bitmap cast member to be cropped to a specific size.

You can use `crop` to trim existing cast members, or in conjunction with the picture of the Stage to grab a snapshot and then crop it to size for display.

The registration point is kept in the same location so the bitmap does not move in relation to the original position.

Example

This statement sets an existing bitmap member to a snapshot of the Stage then crops the resulting image to a rectangle equal to sprite 10:

```
member("stage image").picture = (the stage).picture
member("stage image").crop(sprite(10).rect)
```

See also

picture (cast member property)

crop (cast member property)

Syntax

```
member(whichCastMember).crop
```

```
the crop of member whichCastMember
```

Description

Cast member property; scales a digital video cast member to fit exactly inside the sprite rectangle in which it appears (`FALSE`), or it crops but doesn't scale the cast member to fit inside the sprite rectangle (`TRUE`).

This property can be tested and set.

Example

This statement instructs Lingo to crop any sprite that refers to the digital video cast member Interview.

Dot syntax:

```
member("Interview").crop = TRUE
```

Verbose syntax:

```
set the crop of member "Interview" to TRUE
```

See also

center

on cuePassed

Syntax

```
on cuePassed(channelID, cuePointNumber,cuePointName)
  statement(s)
end

on cuePassed(me,channelID, cuePointNumber,cuePointName)
  statement(s)
end
```

Description System message and event handler; contains statements that run each time a sound or sprite passes a cue point in its media.

- `me`—The optional `me` parameter is the `scriptInstanceRef` value of the script being invoked. You must include this parameter when using the message in a behavior. If this parameter is omitted, the other arguments will not be processed correctly.
- *`channelID`*—The number of the sound or sprite channel for the file where the cue point occurred.
- *`cuePointNumber`*—The ordinal number of the cue point that triggers the event in the list of the cast member's cue points.
- *`cuePointName`*—The name of the cue point that was encountered.

The message is passed—in order—to sprite, cast member, frame, and movie scripts. For the sprite to receive the event, it must be the source of the sound, like a QuickTime movie or SWA cast member. Use the `isPastCuePoint` property to check cues in behaviors on sprites that don't generate sounds.

Example This handler placed in a Movie or Frame script reports any cue points in sound channel 1 to the Message window:

```
on cuePassed channel, number, name
  if (channel = #Sound1) then
    put "CuePoint" && number && "named" && name && "occurred in
sound 1"
  end if
end
```

See also **scriptInstanceList**, **cuePointNames**, **cuePointTimes**, **isPastCuePoint()**

cuePointNames

Syntax

```
member(whichCastMember).cuePointNames
```

```
the cuePointNames of member whichCastMember
```

Description

Cast member property; creates list of cue point names, or if a cue point is not named, inserts an empty string ("") as a placeholder in the list. Cue point names are useful for synchronizing sound, QuickTime, and animation.

This property is supported by SoundEdit cast members, QuickTime digital video cast members, and Xtras cast members that contain cue points. Xtras that generate cue points at run time may not be able to list cue point names.

Example

This statement obtains the name of the third cue point of a cast member.

Dot syntax:

```
put member("symphony").cuePointNames[3]
```

Verbose syntax:

```
put (getAt(the cuePointNames of member "symphony",3))
```

See also

cuePointTimes, **mostRecentCuePoint**

cuePointTimes

Syntax

```
member(whichCastMember).cuePointTimes
the cuePointTimes of member whichCastMember
```

Description

Cast member property; lists the times of the cue points, in milliseconds, for a given cast member. Cue point times are useful for synchronizing sound, QuickTime, and animation.

This property is supported by SoundEdit cast members, QuickTime digital video cast members, and Xtras cast members that support cue points. Xtras that generate cue points at run time may not be able to list cue point names.

Example

This statement obtains the time of the third cue point for a sound cast member.

Dot syntax:

```
put member("symphony").cuePointTimes[3]
```

Verbose syntax:

```
put (getAt(the cuePointTimes of member "symphony",3))
```

See also

cuePointNames, **mostRecentCuePoint**

currentSpriteNum

Syntax `the currentSpriteNum`

Description Movie property; indicates the channel number of the sprite whose script is currently running. It is valid in behaviors and cast member scripts. When used in frame scripts or movie scripts, the `currentSpriteNum` property's value is 0.

The `currentSpriteNum` property is similar to `spriteNum of me`, but it doesn't require the `me` reference.

This property can be tested but not set.

Note: This property was more useful during transitions from older movies to Director 6, when behaviors were introduced. It allowed some behavior-like functionality without having to completely rewrite Lingo code. It is not necessary when authoring with behaviors and is therefore less useful than in the past.

Example The following handler in a cast member or movie script switches the cast member assigned to the sprite involved in the `mouseDown` event:

```
on mouseDown
  sprite(the currentSpriteNum).member = member "DownPict"
end
```

See also **me**, **spriteNum**

currentTime

Syntax

```
sprite(whichSprite).currentTime
the currentTime of sprite whichSprite
sound(channelNum).currentTime
```

Description

Sprite and sound channel property; returns the current playing time, in milliseconds, for a sound sprite, QuickTime digital video sprite, or any Xtra that supports cue points. For a sound channel, returns the current playing time of the sound member currently playing in the given sound channel.

This property can be tested and set. When this property is set, the range of allowable values is from zero to the `duration` of the member.

Shockwave Audio (SWA) sounds can appear as sprites in sprite channels, but they play sound in a sound channel. You should refer to SWA sound sprites by their sprite channel number rather than by a sound channel number.

Example

This statement displays the current time, in seconds, of the sound sprite in sprite channel 10.

Dot syntax:

```
member("time").text = string(sprite (10).currentTime/ 1000)
```

Verbose syntax:

```
set the text of member "time" to (the currentTime of sprite 10) /
1000
```

Example

This statement causes the sound playing in sound channel 2 to skip to the point 2.7 seconds from the beginning of the sound cast member:

```
sound(2).currentTime = 2700
```

See also

movieTime, **duration**

cursor (command)

Syntax `cursor [castNumber, maskCastNumber]`

`cursor whichCursor`

`cursor (member whichCursorCastMember)`

Description Command; changes the cast member or built-in cursor that is used for a cursor and stays in effect until you turn it off by setting the cursor to 0.

- Use the syntax `cursor [castNumber, maskCastNumber]` to specify the number of a cast member to use as a cursor and its optional mask. The cursor's hot spot is the registration point of the cast member.

  The cast member that you specify must be a 1-bit cast member. If the cast member is larger than 16 by 16 pixels, Director crops it to a 16-by-16-pixel square, starting in the upper left corner of the image. The cursor's hot spot is still the registration point of the cast member.

- Use the syntax `cursor whichCursor` to specify default system cursors. The term `whichCursor` must be one of the following integer values:

0*	No cursor set
-1	Arrow (pointer) cursor
1	I-beam cursor
2	Crosshair cursor
3*	Crossbar cursor
4	Watch cursor (Macintosh only)
200*	Blank cursor (hides cursor)

- * The Director player for Java does not support these cursor types and displays an arrow cursor instead.

- Use the syntax `cursor (member whichCursorCastMember)` for the custom cursors available through the Cursor Xtra.

  Be sure not to confuse the syntax `cursor 1` with `cursor [1]`. The first selects the I-beam from the system cursor set; the second uses cast member 1 as the custom cursor.

  Note: Although the Cursor Xtra allows cursors of different cast types, text cast members cannot be used as cursors.

  During system events such as file loading, the operating system may display the watch cursor and then change to the pointer cursor when returning control to the application, overriding the `cursor` command settings from the previous movie. To use `cursor` at the beginning of any new movie that is loaded in a presentation using a custom cursor for multiple movies, store any special cursor resource number as a global variable that remains in memory between movies.

  Cursor commands can be interrupted by an Xtra or other external agent. If the cursor is set to a value in Director and an Xtra or external agent takes control of the cursor, resetting the cursor to the original value has no effect because Director doesn't perceive that the cursor has changed. To work around this, explicitly set the cursor to a third value and then reset it to the original value.

Example This statement changes the cursor to a watch cursor on the Macintosh, and hourglass in Windows, whenever the value in the variable named status equals 1:

```
if status = 1 then cursor 4
```

This handler checks whether the cast member assigned to the variable is a 1-bit cast member and then uses it as the cursor if it is:

```
on myCursor someMember
  if the depth of member someMember = 1 then
    cursor[someMember]
      else
    beep
  end if
end
```

See also **cursor (sprite property)**, **rollOver()**

cursor (sprite property)

Syntax

```
sprite(whichSprite).cursor = [castNumber, maskCastNumber]
set the cursor of sprite whichSprite to [castNumber, maskCastNumber]
sprite(whichSprite).cursor = whichCursor
set the cursor of sprite whichSprite to whichCursor
```

Description

Sprite property; determines the cursor used when the pointer is over the sprite specified by the integer expression *whichSprite*. This property stays in effect until you turn it off by setting the cursor to 0. Use the `cursor` sprite property to change the cursor when the mouse pointer is over specific regions of the screen and to indicate regions where certain actions are possible when the user clicks on them.

When you set the `cursor` sprite property in a given frame, Director keeps track of the sprite rectangle to determine whether to alter the cursor. This rectangle persists when the movie enters another frame unless you set the `cursor` sprite property for that channel to 0.

- Use the syntax `cursor of sprite...[castNumber, maskCastNumber]` to specify the number of a cast member to use as a cursor and its optional mask.
- Use the syntax `cursor of sprite...whichCursor` to specify default system cursors. The term *whichCursor* must be one of the following integer values:

0*	No cursor set
-1	Arrow (pointer) cursor
1	I-beam cursor
2	Crosshair cursor
3*	Crossbar cursor
4	Watch cursor (Macintosh only)
200*	Blank cursor (hides cursor)

- The Director player for Java does not support these cursor types and displays an arrow cursor instead.

To use custom cursors, set the `cursor` sprite property to a list containing the cast member to be used as a cursor or to the number that specifies a system cursor. In Windows, a cursor must be a cast member, not a resource; if a cursor is not available because it is a resource, Director displays the standard arrow cursor instead. For best results, don't use custom cursors when creating cross- platform movies.

If the sprite is a bitmap that has matte ink applied, the cursor changes only when the cursor is over the matte portion of the sprite.

When the cursor is over the location of a sprite that has been removed, rollover still occurs. Avoid this problem by not performing rollovers at these locations or by relocating the sprite up above the menu bar before deleting it.

On the Macintosh, you can use a numbered cursor resource in the current open movie file as the cursor by replacing *whichCursor* with the number of the cursor resource.

This property can be tested and set.

Example

This statement changes the cursor that appears over sprite 20 to a watch cursor.

Dot syntax:

```
sprite(20).cursor = 4
```

Verbose syntax:

```
set the cursor of sprite 20 to 4
```

See also **cursor (command)**

cursorSize

Syntax

```
member(whichCursorCastMember).cursorSize
the cursorSize of member whichCursorCastMember
```

Description

Cursor cast member property; specifies the size of the animated color cursor cast member `whichCursorCastMember`.

Specify size:	For cursors up to:
16	16 by 16 pixels
32	32 by 32 pixels

Description

Bitmap cast members smaller than the specified size are displayed at full size, and larger ones are scaled proportionally to the specified size.

The default value is 32 for Windows and 16 for the Macintosh. If you set an invalid value, an error message appears when the movie runs (but not when you compile).

This property can be tested and set.

Example

This command resizes the animated color cursor stored in cast member 20 to 32 by 32 pixels.

Dot syntax:

```
member(20).cursorSize = 32
```

Verbose syntax:

```
set the cursorSize of member 20 = 32
```

curve

Syntax `member.curve[curveListIndex]`

Description This property contains the `vertexList` of an individual curve (shape) from a vector shape cast member. You can use the curve property along with the `vertex` property to get individual vertices of a specific curve in a vector shape.

A `vertexList` is a list of vertices, and each vertex is a property list containing up to three properties: a #vertex property with the location of the vertex, a #handle1 property with the location of the first control point for that vertex, and a #handle2 property with the location of the second control point for that vertex. See `vertexList`.

Example This statement displays the list of vertices of the third curve of vector shape member SimpleCurves:

```
put member("SimpleCurves").curve[3]
-- [[#vertex: point(113.0000, 40.0000), #handle1: point(32.0000,
10.0000), #handle2: point(- 32.0000, -10.0000)], [#vertex:
point(164.0000, 56.0000)]]
```

Example This statement moves the first vertex of the first curve in a vector shape down and to the right by 10 pixels:

```
member(1).curve[1].vertex[1] = member(1).curve[1].vertex[1] +
point(10, 10)
```

Example The code below moves a sprite to the location of the first vertex of the first curve in a vector shape. The vector shape's `originMode` must be set to `#topLeft` for this to work.

```
vertexLoc = member(1).curve[1].vertex[1]
spriteLoc = mapMemberToStage(sprite(3), vertexLoc)
sprite(7).loc = spriteLoc
```

See also **vertex**, **vertexList**

date() (system clock)

Syntax
```
the abbr date
the abbrev date
the abbreviated date
the date
the long date
the short date
```

Description Function; returns the current date in the system clock in one of three formats: `abbreviated`, `long`, or `short` (default). The abbreviated format can also be referred to as `abbrev` and `abbr`.

In Java, the `date` function is available, but it doesn't accept `abbrev`, `long`, or `short` modifiers. When the movie plays back as an applet, the date's format is `MM/DD/YY`, where `MM` represents the month, `DD` represents the day, and `YY` represents the last two digits of the current year. For the months January through September, the value for `MM` is a single digit.

The format Director uses for the date varies, depending on how the date is formatted on the computer.

- In Windows, you can customize the date display by using the International control panel. (Windows stores the current short date format in the System.ini file. Use this value to determine what the parts of the short date indicate.)
- On the Macintosh, you can customize the date display by using the Date and Time control panel.

Example This statement displays the abbreviated date:

```
put the abbreviated date
-- "Sat, Sep 7, 1991"
```

Example This statement displays the long date:

```
put the long date
-- "Saturday, September 7, 1991"
```

Example This statement displays the short date:

```
put the short date
-- "9/7/91"
```

Example This statement tests whether the current date is January 1 by checking whether the first four characters of the date are 1/1. If it is January 1, the alert "Happy New Year!" appears:

```
if char 1 to 4 of the date = "1/1/" then alert "Happy New Year!"
```

Note: The three date formats vary, depending on the country for which your operating system was designed. These examples are for the United States. Use the `date` object to create and manipulate dates in a standard format.

See also **time()**,**date() (formats)**, **systemDate**

date() (formats)

Syntax

```
date(ISOFormatString)

date(ISOFormatInteger)

date(ISOFormatIntegerYear, ISOFormatIntegerMonth,
ISOFormatIntegerDay)
```

Description

Function and data type; creates a standard, formatted date object instance for use with other date object instances in arithmetic operations and for use in manipulating dates across platforms and in international formats.

When creating the date, use four digits for the year, two digits for the month, and two digits for the day. The following expressions are equivalent:

integer:	set vacationStart = date(19980618)
string:	set vacationStart = date("19980618")
comma separated:	set vacationStart = date(1998, 06, 18)

Addition and subtraction operations on the date are interpreted as the addition and subtraction of days.

The individual properties of the date object instance returned are:

#year	Integer representing the year
#month	Integer representing the month of the year
#day	Integer representing the day of the month

Example

These statements create and determine the number of days between two dates:

```
myBirthDay = date(19650712)
yourBirthDay = date(19450529)
put "There are" && abs(yourBirthday - myBirthday) && "days between
our birthdays."
```

Example

These statements access an individual property of a date:

```
myBirthDay = date(19650712)
put "I was born in month number"&&myBirthday.month
```

See also

date() (system clock)

deactivateApplication

Syntax `on deactivateApplication`

Description Built-in handler; runs when the projector is sent to the background. This handler is useful when a projector runs in a window and the user can send it to the background to work with other applications. Any MIAWs running in the projector can also make use of this handler.

During authoring, this handler is called only if Animate in Background is turned on in General Preferences.

On Windows, this handler is not called if the projector is merely minimized and no other application is brought to the foreground.

Example This handler plays a sound each time the user sends the projector to the background:

```
on deactivateApplication
  sound(1).queue(member("closeSound"))
  sound(1).play()
end
```

See also **activateApplication**, **on activateWindow**, **on deactivateWindow**

on deactivateWindow

Syntax

```
on deactivateWindow
  statement(s)
end
```

Description

System message and event handler; contains statements that run when the window that the movie is playing in is deactivated. The `on deactivate` event handler is a good place for Lingo that you want executed whenever a window is deactivated.

Example

This handler plays the sound Snore when the window that the movie is playing in is deactivated:

```
on deactivateWindow
  puppetSound 2, "Snore"
end
```

defaultRect

Syntax

```
member(whichFlashOrVectorShapeMember).defaultRect
the defaultRect of member whichFlashOrVectorShapeMember
```

Description

Cast member property; controls the default size used for all new sprites created from a Flash movie or vector shape cast member. The `defaultRect` setting also applies to all existing sprites that have not been stretched on the Stage. You specify the property values as a Director rectangle; for example, rect(0,0,32,32).

The `defaultRect` member property is affected by the cast member's `defaultRectMode` member property. The `defaultRectMode` property is always set to `#Flash` when a movie is inserted into a cast, which means the original `defaultRect` setting is always the size of the movie as it was originally created in Flash. Setting `defaultRect` after that implicitly changes the cast member's `defaultRectMode` property to `#fixed`.

This property can be tested and set.

Example

This handler accepts a cast reference and a rectangle as parameters. It then searches the specified cast for Flash cast members and sets their `defaultRect` property to the specified rectangle.

```
on setDefaultFlashRect whichCast, whichRect
  repeat with i = 1 to the number of members of castLib whichCast
    if member(i, whichCast).type = #flash then
      member(i, whichCast).defaultRect = whichRect
    end if
  end repeat
end
```

See also

defaultRectMode, **flashRect**

defaultRectMode

Syntax
```
member(whichVectorOrFlashMember).defaultRectMode
the defaultRectMode of member whichVectorOrFlashMember
```

Description Cast member property; controls how the default size is set for all new sprites created from Flash movie or vector shape cast members. You specify the property value as a Director rectangle; for example, rect(0,0,32,32).

The `defaultRectMode` property does not set the actual size of a Flash movie's default rectangle; it only determines how the default rectangle is set. The `defaultRectMode` member property can have these values:

- `#flash` (default)—Sets the default rectangle using the size of the movie as it was originally created in Flash.
- `#fixed`—Sets the default rectangle using the fixed size specified by the `defaultRect` member property.

The `defaultRect` member property is affected by the cast member's `defaultRectMode` member property. The `defaultRectMode` property is always set to `#flash` when a movie is inserted into a cast, which means the original `defaultRect` setting is always the size of the movie as it was originally created in Flash. Setting `defaultRect` after that implicitly changes the cast member's `defaultRectMode` property to `#fixed`.

This property can be tested and set.

Example This handler accepts a cast reference and a rectangle as parameters. It then searches the specified cast for Flash cast members, sets their `defaultRectMode` property to `#fixed`, and then sets their `defaultRect` property to rect(0,0,320,240).

```
on setDefaultRectSize whichCast
  repeat with i = 1 to the number of members of castLib whichCast
    if member(i, whichCast).type = #flash then
      member(i, whichCast).defaultRectMode = #fixed
      member(i, whichCast).defaultRect = rect(0,0,320,240)
    end if
  end repeat
end
```

See also **flashRect**, **defaultRect**

delay

Syntax `delay numberOfTicks`

Description Command; pauses the playback head for a given amount of time. The integer expression *`numberOfTicks`* specifies the number of ticks to wait, where each tick is 1/60 of a second. The only mouse and keyboard activity possible during this time is stopping the movie by pressing Control+Alt+period (Windows) or Command+period (Macintosh). Because it increases the time of individual frames, the `delay` command is useful for controlling the playback rate of a sequence of frames.

The `delay` command can be applied only when the playback head is moving. However, when `delay` is in effect, handlers still run: only the playback head halts, not script execution. Place scripts that use the `delay` command in either an `on enterFrame` or `on exitFrame` handler.

To mimic the behavior of a halt in a handler when the playback head is not moving, use the `startTimer` command or assign the current value of `timer` to a variable and wait for the specified amount of time to pass before exiting the frame.

Example This handler delays the movie for 2 seconds when the playback head exits the current frame:

```
on exitFrame
  delay 2 * 60
end
```

Example This handler, which can be placed in a frame script, delays the movie a random number of ticks:

```
on keyDown
  if the key = RETURN then delay random(180)
end
```

Example The first of these handlers sets a timer when the playback head leaves a frame. The second handler, assigned to the next frame, loops in the frame until the specified amount of time passes:

```
--script for first frame

on exitFrame
  global gTimer
  set gTimer = the ticks
end

--script for second frame

on exitFrame
  global gTimer
  if the ticks < gTimer + (10 * 60) then
    go to the frame
  end if
end
```

See also **startTimer**, **ticks**, **timer**

delete

Syntax `delete chunkExpression`

Description Command; deletes the specified chunk expression (character, word, item, or line) in any string container. Sources of strings include field cast members and variables that hold strings.

To see an example of `delete` used in a completed movie, see the Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement deletes the first word of line 3 in the field cast member Address:

```
delete word 1 of line 3 of member "Address"
```

Example The same chunk of text may also be deleted with the syntax:

```
delete member("Address").line[3].word[1]
```

Example This statement deletes the first character of the string in the variable `bidAmount` if that character is the dollar sign ("$"):

```
if bidAmount.char[1] = "$" then delete bidAmount.char[1]
```

See also **char...of**, **field**, **item...of**, **line...of**, **word...of**; **hilite (cast member property)**, **paragraph**

deleteAll

Syntax `list.deleteAll()`

`deleteAll list`

Description List command; deletes all items in the specified list without changing the list type.

Example This statement deletes every item in the list named propList:

```
propList.deleteAll()
```

deleteAt

Syntax

```
list.deleteAt(number)
deleteAt list, number
```

Description

List command; deletes the item in the position specified by *number* from the linear or property list specified by *list*.

The `deleteAt` command checks whether an item is in a list; if you try to delete an object that isn't in the list, Director displays an alert.

Example

This statement deletes the second item from the list named designers, which contains [gee, kayne, ohashi]:

```
designers = ["gee", "kayne", "ohashi"]
designers.deleteAt(2)
```

The result is the list [gee, ohashi].

This handler checks whether an object is in a list before attempting to delete it:

```
on myDeleteAt theList, theIndex
  if theList.count < theIndex then
    beep
  else
    theList.deleteAt(theIndex)
  end if
end
```

See also **addAt**

deleteFrame

Syntax `deleteFrame`

Description Command; deletes the current frame and makes the next frame the new current frame during a Score generation session only.

Example The following handler checks whether the sprite in channel 10 of the current frame has gone past the right edge of a 640-by-480-pixel Stage and deletes the frame if it has:

```
on testSprite
  beginRecording
    if sprite(10).locH > 640 then deleteFrame
  endRecording
end
```

See also **beginRecording**, **endRecording**, **updateFrame**

deleteOne

Syntax

```
list.deleteOne(value)
deleteOne list, value
```

Description

List command; deletes a value from a linear or property list. For a property list, `deleteOne` also deletes the property associated with the deleted value. If the value appears in the list more than once, `deleteOne` deletes only the first occurrence.

Attempting to delete a property has no effect.

Example

The first statement creates a list consisting of the days Tuesday, Wednesday, and Friday. The second statement deletes the name Wednesday from the list.

```
days = ["Tuesday", "Wednesday", "Friday"]
days.deleteOne("Wednesday")
put days
```

The `put days` statement causes the Message window to display the result:

```
-- ["Tuesday", "Friday"].
```

deleteProp

Syntax `list.deleteProp(item)`

`deleteProp list, item`

Description List command; deletes the specified item from the specified list.

- For linear lists, replace *item* with the number identifying the list position of the item to be deleted. The `deleteProp` command for linear lists is the same as the `deleteAt` command. If the number is greater than the number of items in the list, a script error occurs.
- For property lists, replace *item* with the name of the property to be deleted. Deleting a property also deletes its associated value. If the list has more than one of the same property, only the first property in the list is deleted.

Example This statement deletes the color property from the list [#height:100, #width: 200, #color: 34, #ink: 15], which is called spriteAttributes:

```
spriteAttributes.deleteProp(#color)
```

The result is the list [#height:100, #width: 200, #ink: 15].

See also **deleteAt**

deleteVertex()

Syntax `member(memberRef).deleteVertex(indexToRemove)`

`deleteVertex(member memberRef, indexToRemove)`

Description Vector shape command; removes an existing vertex of a vector shape cast member in the index position specified.

Example This line removes the second vertex point in the vector shape Archie:

```
member("Archie").deleteVertex(2)
```

See also **addVertex**, **moveVertex()**,**originMode**, **vertexList**

depth

Syntax
```
imageObject.depth
member(whichCastMember).depth
the depth of member whichCastMember
```

Description Image object or bitmap cast member property; displays the color depth of the given image object or bitmap cast member.

Depth	Number of Colors
1	Black and white
2	4 colors
4, 8	16 or 256 palette-based colors, or gray levels
16	Thousands of colors
32	Millions of colors

Description This property can be tested but not set.

Example This statement displays the color depth of the image object stored in the variable newImage. The output appears in the Message window.

```
put newImage.depth
```

Example This statement displays the color depth of the cast member Shrine in the Message window:

```
put member("Shrine").depth
```

deskTopRectList

Syntax
```
the deskTopRectList
```

Description
System property; displays the size and position on the desktop of the monitors connected to a computer. This property is useful for checking whether objects such as windows, sprites, and pop-up windows appear entirely on one screen.

The result is a list of rectangles, where each rectangle is the boundary of a monitor. The coordinates for each monitor are relative to the upper left corner of monitor 1, which has the value (0,0). The first set of rectangle coordinates is the size of the first monitor. If a second monitor is present, a second set of coordinates shows where the corners of the second monitor are relative to the first monitor.

This property can be tested but not set.

Example
This statement tests the size of the monitors connected to the computer and displays the result in the Message window:

```
put the deskTopRectList
-- [rect(0,0,1024,768), rect(1025, 0, 1665, 480]
```

The result shows that the first monitor is 1024 by 768 pixels and the second monitor is 640 by 480 pixels.

Example
This handler tells how many monitors are in the current system:

```
on countMonitors
  return the deskTopRectList.count
end
```

digitalVideoTimeScale

Syntax `the digitalVideoTimeScale`

Description System property; determines the time scale, in units per second, that the system uses to track digital video cast members.

For example, if `digitalVideoTimeScale` is set to:

- 100—The time scale is 1/100 of a second (and the movie is tracked in 100 units per second).
- 500—The time scale is 1/500 of a second (and the movie is tracked in 500 units per second).
- 0—Director uses the time scale of the movie that is currently playing.

Set `digitalVideoTimeScale` to precisely access tracks by ensuring that the system's time unit for video is a multiple of the digital video's time unit.

This property can be tested and set.

Example This statement sets the time scale that the system uses to measure digital video to 600 units per second:

```
the digitalVideoTimeScale to 600
```

digitalVideoType

Syntax

```
member(whichCastMember).digitalVideoType
the digitalVideoType of member whichCastMember
```

Description

Cast member property; indicates the format of the specified digital video. Possible values are `#quickTime` or `#videoForWindows`.

This property can be tested but not set.

Example

The following statement tests whether the cast member Today's Events is a QuickTime or AVI (Audio-Video Interleaved) digital video and displays the result in the Message window:

```
put member("Today's Events").digitalVideoType
```

See also

quickTimeVersion()

directToStage

Syntax
```
member(whichCastMember).directToStage
the directToStage of member whichCastMember
sprite(whichSprite).directToStage
the directToStage of sprite whichSprite
```

Description Sprite property and member property; determines the layer where a digital video, animated GIF, vector shape, or Flash Asset cast member plays. If this property is `TRUE` (1), the cast member plays in front of all other layers on the Stage, and ink effects have no affect. If this property is `FALSE` (0), the cast member can appear in any layer of the Stage's animation planes, and ink effects affect the appearance of the sprite.

- Use the syntax `member(whichCastMember).directToStage` for digital video or animated GIFs.
- Use the syntax `sprite(whichSprite).directToStage` for Flash or vector shapes.

Using this property may improve the playback performance of supported cast members.

No other cast member can appear in front of a `directToStage` sprite. Also, ink effects do not affect the appearance of a `directToStage` sprite.

When a sprite's `directToStage` property is `TRUE`, Director draws the sprite directly to the screen without first compositing it in Director's offscreen buffer. The result can be similar to the trails ink effect of the Stage.

Explicitly refresh a trailed area by turning the `directToStage` property off and on, using a full-screen transition, or "wiping" another sprite across this area. (In Windows, you can branch to another similar screen, and the video won't completely disappear.)

To see an example of `directToStage` used in a completed movie, see the QT and Flash movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement makes the QuickTime movie The Residents always play in the top layer of the Stage:

```
member("The Residents").directToStage = 1
```

dither

Syntax

```
member(whichMember).dither
the dither of member whichMember
```

Description

Bitmap cast member property; dithers the cast member when it is displayed at a color depth of 8 bits or less (256 colors) if the display must show a color gradation not in the cast member (`TRUE`), or tells Director to choose the nearest color out of those available in the current palette (`FALSE`).

For both performance and quality reasons, you should set `dither` to `TRUE` only when higher display quality is necessary. Dithering is slower than remapping, and artifacts may be more apparent when animating over a dithered image.

If the color depth is greater than 8 bits, this property has no effect.

See also **depth**

do

Syntax `do stringExpression`

Description Command; evaluates *`stringExpression`* and executes the result as a Lingo statement. This command is useful for evaluating expressions that the user has typed and for executing commands stored in string variables, fields, arrays, and files.

Using uninitialized local variables within a `do` command creates a compile error. Initialize any local variables in advance.

Note: This command does not allow global variables to be declared; these variables must be declared in advance.

The `do` command works with multiple-line strings as well as single lines.

Example This statement performs the statement contained within quotation marks:

```
do "beep 2"
do commandList[3]
```

doubleClick

Syntax `the doubleClick`

Description Function; tests whether two mouse clicks within the time set for a double-click occurred as a double-click rather than two single clicks (`TRUE`), or if they didn't occur within the time set, treats them as single clicks (`FALSE`).

Example This statement branches the playback head to the frame Enter Bid when the user double-clicks the mouse button:

```
if the doubleClick then go to frame "Enter Bid"
```

Example The following handler tests for a double-click. When the user clicks the mouse, a repeat loop runs for the time set for a double-click (20 ticks in this case). If a second click occurs within 20 ticks, the `doubleClickAction` handler runs. If a second click doesn't occur within the specified period, the `singleClickAction` handler runs:

```
on mouseUp
  if the doubleClick then exit
  startTimer
  repeat while the timer < 20
    if the mouseDown then
      doubleClickAction
      exit
    end if
  end repeat
  singleClickAction
end mouseUp
```

See also **clickOn**, **the mouseDown (system property)**, **the mouseUp (system property)**

downloadNetThing

Syntax `downloadNetThing` *`URL`*`, `*`localFile`*

Description Command; copies a file from the Internet to a file on the local disk, while the current movie continues playing. Use `netDone` to find out whether downloading is finished.

- *`URL`*—The file name of any object that can be downloaded: for example, an FTP or HTTP server, an HTML page, an external cast member, a Director movie, or a graphic
- *`localFile`*—The pathname and file name for the file on the local disk

Director movies in authoring mode and projectors support the `downLoadNetThing` command, but the Shockwave player does not. This protects users from unintentionally copying files from the Internet.

Although many network operations can be active at one time, running more than four concurrent operations usually slows down performance unacceptably.

Neither the Director movie's cache size nor the setting for the Check Documents option affects the behavior of the `downloadNetThing` command.

Note: Director for Java does not support the downloadNetThing command.

Example These statements download an external cast member from a URL to the Director application folder and then make that file the external cast member named Cast of Thousands:

```
downLoadNetThing("http://www.cbDeMille.com/Thousands.cst", the  \
applicationPath&"Thousands.Cst")
castLib("Cast of Thousands").fileName = the
applicationPath&"Thousands.Cst"
```

See also **importFileInto**, **netDone()**,**preloadNetThing()**

draw()

Syntax

```
imageObject.draw(x1, y1, x2, y2, colorObjectOrParameterList)
imageObject.draw(point(x, y), point(x, y),
colorObjectOrParameterList)
imageObject.draw(rect, colorObjectOrParameterList)
```

Description

This function draws a line or an unfilled shape of color *colorObject* in a rectangular region of the given image object, as specified in any of the three ways shown. The `draw` returns a value of 1 if there is no error. You can use the optional property list *ParameterList* function to specify the following shape properties:

Property	Description
#shapeType	A symbol value of #oval, #rect, #roundRect, or #line. The default is #line.
#lineSize	The width of the line to use in drawing the shape.
#color	A color object, which determines the color of the shape border.

Description

If you do not provide a parameter list, this function draws a 1-pixel line between the first and second points given or between the upper left and lower right corners of the given rectangle.

For best performance, with 8-bit or lower images the *colorObject* should contain an indexed color value. For 16- or 32-bit images, use an RGB color value.

If you want to fill a solid region, use the `fill()` function.

Example

This statement draws a 1-pixel, dark red, diagonal line from point (0, 0) to point (128, 86) within the image of member Happy.

```
member("Happy").image.draw(0, 0, 128, 86, rgb(150,0,0))
```

Example

This statement draws a dark red, 3-pixel unfilled oval within the image of member Happy. The oval is drawn within the rectangle (0, 0, 128, 86).

```
member("Happy").image.draw(0, 0, 128, 86, [#shapeType:#oval,
#lineSize:3, \
#color: rgb(150, 0, 0)])
```

See also **color()**,**copyPixels()**,**fill()**,**setPixel()**

drawRect

Syntax
```
window windowName.drawRect
the drawRect of window windowName
```

Description Window property; identifies the rectangular coordinates of the Stage of the movie that appears in the window. The coordinates are given as a rectangle, with entries in the order left, top, right, and bottom.

This property is useful for scaling or panning movies, but it does not rescale text and field cast members. Scaling bitmaps can affect performance.

This property can be tested and set.

Example This statement displays the current coordinates of the movie window called Control Panel:

```
put the drawRect of window "Control Panel"
-- rect(10, 20, 200, 300).
```

Example This statement sets the rectangle of the movie to the values of the rectangle named movieRectangle. The part of the movie within the rectangle is what appears in the window.

```
set the drawRect of window "Control Panel" to movieRectangle
```

Example The following lines will cause the Stage to fill the main monitor area:

```
(the stage).drawRect = the desktopRectList[1]
(the stage).rect = the desktopRectList[1]
```

See also **deskTopRectList**, **rect()**,**sourceRect**

dropShadow

Syntax
```
member(whichCastMember).dropShadow
the dropShadow of member whichCastMember
```

Description Cast member property; determines the size of the drop shadow in pixels, for text in a field cast member.

Example This statement sets the drop shadow of the field cast member Comment to 5 pixels:

```
member("Comment").dropShadow = 5
```

duplicateFrame

Syntax `duplicateFrame`

Description Command; duplicates the current frame and its content, inserts the duplicate frame after the current frame, and then makes the duplicate frame the current frame. This command can be used during Score generation only.

The `duplicateFrame` command performs the same function as the `insertFrame` command.

Example When used in the following handler, the `duplicateFrame` command creates a series of frames that have cast member Ball in the external cast Toys assigned to sprite channel 20. The number of frames is determined by the argument `numberOfFrames`.

```
on animBall numberOfFrames
  beginRecording
    sprite(20).member = member("Ball", "Toys")
    repeat with i = 0 to numberOfFrames
      duplicateFrame
    end repeat
  endRecording
end
```

duplicate() (list function)

Syntax

```
(oldList).duplicate()
duplicate(oldList)
```

Description

List function; returns a copy of a list and copies nested lists (list items that also are lists) and their contents. The function is useful for saving a list's current content.

When you assign a list to a variable, the variable contains a reference to the list, not the list itself. This means any changes to the copy also affect the original list.

To see an example of `duplicate()` used in a completed movie, see the Imaging movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement makes a copy of the list CustomersToday and assigns it to the variable `CustomerRecord`:

```
CustomerRecord = CustomersToday.duplicate()
```

duplicate() (image function)

Syntax `imageObject.duplicate()`

Description This function creates and returns a copy of the given *imageObject*. The new image is completely independent of the original, and isn't linked to any cast member.

If you plan to make a lot of changes to an image, it's better to make a copy that's independent of a cast member.

Example This statement creates a new image object from the image of cast member Lunar Surface and places the new image object into the variable `workingImage`:

```
workingImage = member("Lunar Surface").image.duplicate()
```

See also **duplicate member**

duplicate member

Syntax

```
member(originalMember).duplicate()
member(originalMember).duplicate({new})
duplicate member original {, new}
```

Description

Command; makes a copy of the cast member specified by *original*. The optional *new* parameter specifies a specific Cast window location for the duplicate cast member. If the *new* parameter isn't included, the duplicate cast member is placed in the first open Cast window position.

This command is best used during authoring rather than run time because it creates another cast member in memory, which could result in memory problems. Use the command during authoring if you want the changes to the cast to be permanently saved with the file.

Example

This statement makes a copy of cast member Desk and places it in the first empty Cast window position:

```
member("Desk").duplicate()
```

Example

This statement makes a copy of cast member Desk and places it in the Cast window at position 125:

```
member("Desk").duplicate(125)
```

duration

Syntax `member(whichCastMember).duration`

`the duration of member whichCastMember`

Description Cast member property; determines the duration of the specified sound, Shockwave Audio (SWA), transition, or QuickTime cast member.

- When *whichCastMember* is a sound cast member or a a streaming Shockwave Audio file, this property indicates the duration of the sound. For Shockwave Audio, the `duration` property returns 0 until streaming begins. Setting `preLoadTime` to 1 second allows the bit rate to return the actual duration.
- When *whichCastMember* is a digital video cast member, this property indicates the digital video's duration. The value is in ticks.
- When *whichCastMember* is a transition cast member, this property indicates the transition's duration. The value for the transition is in milliseconds. During playback, this setting has the same effect as the Duration setting in the Frame Transition dialog box.

This property can be tested for all cast members that support it, but only set for transitions.

To see an example of `duration` used in a completed movie, see the QT and Flash movie in the Learning\Lingo Examples folder inside the Director application folder.

Example If the SWA cast member Louie Prima has been preloaded, this statement displays the sound's duration in the field cast member Duration Displayer:

```
on exitFrame
  if member("Louie Prima").state = 2 then
    member("Duration Displayer").text = member("Louie
Prima").duration
  end if
end
```

Example You can use a behavior on a digital video sprite to loop the playback head in the current frame until the movie is finished playing, allowing it to continue when the end is reached:

```
property spriteNum
on exitFrame me
  myMember = sprite(spriteNum).member
  myDuration = member(myMember).duration
  myMovietime = sprite(spriteNum).movieTime
  if myDuration > myMovietime then
    go to the frame
  else
    go to the frame + 1
  end if
end
```

editable

Syntax

```
member(whichCastMember).editable
the editable of member whichCastMember
sprite(whichSprite).editable
the editable of sprite whichSprite
```

Description

Cast member and sprite property; determines whether the specified field cast member can be edited on the Stage (`TRUE`) or not (`FALSE`).

When the cast member property is set, the setting is applied to all sprites that contain the field. When the sprite property is set, only the specified sprite is affected.

You can also make a field cast member editable by using the Editable option in the Field Cast Member Properties dialog box.

You can make a field sprite editable by using the Editable option in the Score.

For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet.

This property can be tested and set.

Example

This statement makes the field cast member Answer editable:

```
member("Answer").editable = TRUE
```

Example

This handler first makes the sprite channel a puppet and then makes the field sprite editable:

```
on myNotes
  puppetSprite 5, TRUE
  sprite(5).editable = TRUE
end
```

Example

This statement checks whether a field sprite is editable and displays a message if it is:

```
if sprite(13).editable = TRUE then
  member("Notice").text = "Please enter your answer below."
end if
```

editShortCutsEnabled

Syntax `the editShortCutsEnabled`

Description Movie property; determines whether cut, copy, and paste operations and their keyboard shortcuts function in the current movie. When set to `TRUE`, these text operations fuction. When set to `FALSE`, these operations are not allowed.

This property can be tested and set. The default is `TRUE` for movies made in Director 8, `FALSE` for movies made in versions of Director prior to Director 8.

Example This statement disables cut, copy, and paste operations:

```
the editShortCutsEnabled = 0
```

elapsedTime

Syntax

```
sound(channelNum).elapsedTime
the elapsedTime of sound channelNum
```

Description

This read-only property gives the time, in milliseconds, that the current sound member in the given sound channel has been playing. It starts at 0 when the sound begins playing and increases as the sound plays, regardless of any looping, setting of the `currentTime` or other manipulation. Use the `currentTime` to test for the current absolute time within the sound.

> The value of this property is a floating-point number, allowing for measurement of sound playback to fractional milliseconds.

Example

This idle handler displays the elapsed time for sound channel 4 in a field on the Stage during idles:

```
on idle
  member("time").text = string(sound(4).elapsedTime)
end idle
```

See also

currentTime, **loopCount**, **loopsRemaining**, **rewind()**

EMPTY

Syntax `EMPTY`

Description Character constant; represents the empty string, `""`, a string with no characters.

Example This statement erases all characters in the field cast member Notice by setting the field to `EMPTY`:

```
member("Notice").text = EMPTY
```

emulateMultiButtonMouse

Syntax `the emulateMultiButtonMouse`

Description System property; determines whether a movie interprets a mouse click with the Control key pressed on the Macintosh the same as a right mouse click in Windows (`TRUE`) or not (`FALSE`).

Right-clicking has no direct Macintosh equivalent.

Setting this property to `TRUE` lets you provide consistent mouse button responses for cross-platform movies.

Example The following statement checks if the computer is a Macintosh and if so, sets the `emulateMultiButtonMouse` property to `TRUE`:

```
if the platform contains "Macintosh" then the
emulateMultiButtonMouse = TRUE
```

See also **keyPressed()**,**rightMouseDown (system property)**, **rightMouseUp (system property)**

enabled

Syntax `the enabled of menuItem whichItem of menu whichMenu`

Description Menu item property; determines whether the menu item specified by *whichItem* is displayed in plain text and is selectable (`TRUE`, default) or appears dimmed and is not selectable (`FALSE`).

The expression *whichItem* can be either a menu item name or a menu item number. The expression *whichMenu* can be either a menu name or a menu number.

The enabled property can be tested and set.

Note: Menus are not available in Shockwave

Example This handler enables or disables all the items in the specified menu. The argument `theMenu` specifies the menu; the argument `Setting` specifies `TRUE` or `FALSE`. For example, the calling statement `ableMenu ("Special", FALSE)` disables all the items in the Special menu.

```
on ableMenu theMenu, vSetting
  set n = the number of menuItems of menu theMenu
  repeat with i = 1 to n
    set the enabled of menuItem i of menu theMenu to vSetting
  end repeat
end ableMenu
```

See also **name (menu property)**, **number (menus)**, **checkMark**, **script**, **number (menu items)**

enableHotSpot

Syntax `enableHotSpot(sprite whichQTVRSprite, hotSpotID, trueOrFalse)`

Description QTVR (QuickTime VR) command; determines whether the specified hot spot for the specified QTVR sprite is enabled (`TRUE`), or disabled (`FALSE`).

end

Syntax `end`

Description Keyword; marks the end of handlers and multiple-line control structures.

end case

Syntax `end case`

Description Keyword; ends a `case` statement.

Example This handler uses the `end case` keyword to end the `case` statement:

```
on keyDown
  case the key
    of "A": go to frame "Apple"
    of "B", "C" :
      puppetTransition 99
      go to frame "Mango"
    otherwise beep
  end case
end keyDown
```

See also **case**

endColor

Syntax `the endColor of member whichCastMember`

Description Vector shape cast member property; the ending color of a gradient shape's fill specified as an RGB value.

`endColor` is only valid when the `fillMode` is set to `#gradient`, and the starting color is set with `fillColor`.

This property can be tested and set.

To see an example of `endColor` used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.

See also **color()**,**fillColor**, **fillMode**

endFrame()

Syntax `sprite(whichSprite).endFrame`

Description Function; returns the frame number of the end frame of the sprite span.

This function is useful in determining the span in the Score of a particular sprite. This function is available only in a frame that contains the sprite. It cannot be applied to sprites in different frames of the movie, nor is it possible to set this value.

Example This statement output reports the ending frame of the sprite in channel 5 in the Message window:

```
put sprite(5).endFrame
```

See also **startFrame**

endRecording

Syntax `endRecording`

Description Keyword; ends a Score update session. You can resume control of Score channels through puppetting after the `endRecording` keyword is issued.

Example When used in the following handler, the `endRecording` keyword ends the Score generation session:

```
on animBall numberOfFrames
  beginRecording
    horizontal = 0
    vertical = 100
    repeat with i = 1 to numberOfFrames
      go to frame i
      sprite(20).member = member "Ball"
      sprite(20).locH = horizontal
      sprite(20).locV = vertical
      horizontal = horizontal + 3
      vertical = vertical + 2
      updateFrame
    end repeat
  endRecording
end
```

See also **beginRecording**, **scriptNum**, **tweened**, **scriptNum**, **updateFrame**

end repeat

See **repeat while**, **repeat with**, **repeat with...in list**, **repeat with...down to**

on endSprite

Syntax

```
on endSprite
  statement(s)
end
```

Description

System message and event handler; contains Lingo that runs when the playback head leaves a sprite and goes to a frame in which the sprite doesn't exist. It is generated after `exitFrame`.

Place `on endSprite` handlers in a behavior script.

Director destroys instances of any behavior scripts attached to the sprite immediately after the `endSprite` event occurs.

The event handler is passed the behavior or frame script reference `me` if used in a behavior. This `endSprite` message is sent after the `exitFrame` message if the playback head plays to the end of the frame.

The `go`, `play`, and `updateStage` commands are disabled in an `on endSprite` handler.

Example

This handler runs when the playback head exits a sprite:

```
on endSprite me
  -- clean up
  gNumberOfSharks = gNumberOfSharks - 1
  puppetSound(5,0)
end
```

See also

on beginSprite, **on exitFrame**

endTime

Syntax

```
sound(channelNum).endTime
```

```
the endTime of sound channelNum
```

Description

This property is the specified end time of the currently playing, paused or queued sound. This is the time within the sound member when it will stop playing. It's a floating-point value, allowing for measurement and control of sound playback to fractions of milliseconds. The default value is the normal end of the sound.

This property may be set to a value other than the normal end of the sound only when passed as a parameter with the `queue()` or `setPlayList()` commands.

Example

This Lingo checks whether the sound member Jingle is set to play all the way through in sound channel 1:

```
if sound(1).startTime > 0 and sound(1).endTime <
member("Jingle").duration then
  alert "Not playing the whole sound"
end if
```

See also

setPlaylist(), **queue()**, **startTime**

ENTER

Syntax `ENTER`

Description Character constant; represents the Enter key (Windows) or the Return key (Macintosh) for a carriage return.

On PC keyboards, the element `ENTER` refers only to the Enter key on the numeric keypad.

For a movie that plays back as an applet, use `RETURN` to specify both the Return key in Windows and the Enter key on the Macintosh.

Example This statement checks whether the Enter key is pressed and if it is, sends the playback head to the frame addSum:

```
on keyDown
  if the key = ENTER then go to frame "addSum"
end
```

See also **RETURN (constant)**

on enterFrame

Syntax

```
on enterFrame
  statement(s)
end
```

Description

System message and event handler; contains statements that run each time the playback head enters the frame.

Place `on enterFrame` handlers in behavior, frame, or movie scripts, as follows:

- To assign the handler to an individual sprite, put the handler in a behavior attached to the sprite.
- To assign the handler to an individual frame, put the handler in the frame script.
- To assign the handler to every frame (unless you explicitly instruct the movie otherwise), put the `on enterFrame` handler in a movie script. The handler executes every time the playback head enters a frame unless the frame script has its own handler. If the frame script has its own handler, the `on enterFrame` handler in the frame script overrides the `on enterFrame` handler in the movie script.

The order of frame events is stepFrame, prepareFrame, enterFrame, and exitFrame.

This event is passed the object reference `me` if used in a behavior.

Example

This handler turns off the puppet condition for sprites 1 through 5 each time the playback head enters the frame:

```
on enterFrame
  repeat with i = 1 to 5
    puppetSprite i, FALSE
  end repeat
end
```

environment

Syntax

```
the environment
the envrionment.propertyName
```

Description

System property; this property contains a list with information about the environment under which the Director content is currently running.

This design enables Macromedia to add information to the `environment` property in the future, without affecting existing movies.

The information is in the form of property and value pairs for that area.

#shockMachine	Integer TRUE or FALSE value indicating whether the movie is playing in ShockMachine.
#shockMachineVersion	String indicating the installed version number of ShockMachine.
#platform	String containing "Macintosh,PowerPC", or "Windows,32". This is based on the current OS and hardware that the movie is running under.
#runMode	String containing "Author", "Projector", "Plugin", or "Java Applet". This is based on the current application that the movie is running under.
#colorDepth	Integer representing the bit depth of the monitor the Stage appears on. Possible values are 1, 2, 4, 8, 16, or 32.
#internetConnected	Symbol indicating whether the computer the movie is playing on has an active Internet connection. Possible values are `#online` and `#offline`.
#uiLanguage	String indicating the language the computer is using to display its user interface. This can be different from the `#osLanguage` on computers with specific language kits installed.
#osLanguage	String indicating the native language of the computer's operating system.
#productBuildVersion	String indicating the internal build number of the playback application.

The properties contain exactly the same information as the properties and functions of the same name.

Example

This statement displays the environment list in the Message window:

```
put the environment
-- [#shockMachine: 0, #shockMachineVersion: "", #platform:
"Macintosh,PowerPC", #runMode: "Author", #colorDepth: 32,
#internetConnected: #online, #uiLanguage: "English", #osLanguage:
"English", #productBuildVersion: "151"]
```

See also

colorDepth, **platform**, **runMode**

erase member

Syntax
```
member(whichCastMember).erase()
```
```
erase member whichCastMember
```

Description
Command; deletes the specified cast member and leaves its slot in the Cast window empty.

For best results, use this command during authoring and not in projectors, which can cause memory problems.

Example
This statement deletes the cast member named Gear in the Hardware cast:

```
member("Gear", "Hardware").erase()
```

Example
This handler deletes cast members start through finish:

```
on deleteMember start, finish
  repeat with i = start to finish
    member(i).erase()
  end repeat
end on deleteMember
```

See also
new()

on EvalScript

Syntax

```
on EvalScript aParam
  statement(s)
end
```

Description System message and event handler; in a Shockwave movie, contains statements that run when the handler receives an `EvalScript` message from a browser. The parameter is a string passed in from the browser.

- The `EvalScript` message can include a string that Director can interpret as a Lingo statement. Lingo cannot accept nested strings. If the handler you are calling expects a string as a parameter, pass the parameter as a symbol.
- The `on EvalScript` handler is called by the `EvalScript()` scripting method from JavaScript or VBScript in a browser.

The Director player for Java doesn't support the `on EvalScript` handler. To enable communication between an applet and a browser, use Java, JavaScript, or VBScript.

Include only those behaviors in `on EvalScript` that you want users to control; for security reasons, don't give complete access to behaviors.

Note: If you place a return at the end of your EvalScript handler, the value returned can be used by JavaScript in the browser.

Example This shows how to make the playback head jump to a specific frame depending on what frame is passed in as the parameter:

```
on EvalScript aParam
  go frame aParam
end
```

Example This handler runs the statement `go frame aParam` if it receives an `EvalScript` message that includes dog, cat, or tree as an argument:

```
on EvalScript aParam
  case aParam of
    "dog", "cat", "tree": go frame aParam
  end case
end
```

A possible calling statement for this in JavaScript would be `EvalScript ("dog")`.

Example This handler takes an argument that can be a number or symbol:

```
on EvalScript aParam
  if word 1 of aParam = "myHandler" then
    do aParam
  end if
end
```

Example This handler normally requires a string as its argument. The argument is received as a symbol and then converted to a string within the handler by the `string` function:

```
on myHandler aParam
  go to frame string(aParam)
end
```

See also **externalEvent**, **return (keyword)**

eventPassMode

Syntax
```
sprite(whichFlashSprite).eventPassMode
the eventPassMode of sprite whichFlashSprite
member(whichFlashMember).eventPassMode
the eventPassMode of member whichFlashMember
```

Description Flash cast member property and sprite property; controls when a Flash movie passes mouse events to behaviors that are attached to sprites that lie underneath the flash sprite. The `eventPassMode` property can have these values:

- `#passAlways` (default)—Always passes mouse events.
- `#passButton`—Passes mouse events only when a button in the Flash movie is clicked.
- `#passNotButton`—Passes mouse events only when a nonbutton object is clicked.
- `#passNever`—Never passes mouse events.

This property can be tested and set.

Example This frame script checks to see whether the buttons in a Flash movie sprite are currently enabled, and if so, sets `eventPassMode` to `#passNotButton`; if the buttons are disabled, the script sets `eventPassMode` to `#passAlways`. The effect of this script is that:

- Mouse events on nonbutton objects always pass to sprite scripts.
- Mouse events on button objects are passed to sprite scripts when the buttons are disabled. When the buttons are enabled, mouse events on buttons are stopped.

```
on enterFrame
  if sprite(5).buttonsEnabled = TRUE then
    sprite(5).eventPassMode= #passNotButton
  else
    sprite(5).eventPassMode = #passAlways
  end if
end
```

exit

Syntax `exit`

Description Keyword; instructs Lingo to leave a handler and return to where the handler was called. If the handler is nested within another handler, Lingo returns to the main handler.

Example The first statement of this script checks whether the monitor is set to black and white and then exits if it is:

```
on setColors
  if the colorDepth = 1 then exit
  sprite(1).foreColor = 35
end
```

See also **abort**, **halt**, **quit**, **pass**, **return (keyword)**

on exitFrame

Syntax

```
on exitFrame
  statement(s)
end
```

Description System message and event handler; contains statements that run each time the playback head exits the frame that the `on exitFrame` handler is attached to. The `on exitFrame` handler is a useful place for Lingo that resets conditions that are no longer appropriate after leaving the frame.

Place `on exitFrame` handlers in behavior, frame, or movie scripts, as follows:

- To assign the handler to an individual sprite, put the handler in a behavior attached to the sprite.
- To assign the handler to an individual frame, put the handler in the frame script.
- To assign the handler to every frame unless explicitly instructed otherwise, put the handler in a movie script. The `on exitFrame` handler then executes every time the playback head exits the frame unless the frame script has its own `on exitFrame` handler. When the frame script has its own `on exitFrame` handler, the `on exitFrame` handler in the frame script overrides the one in the movie script.

This event is passed the sprite script or frame script reference `me` if it is used in a behavior. The order of frame events is `prepareFrame`, `enterFrame`, and `exitFrame`.

Example This handler turns off all puppet conditions when the playback head exits the frame:

```
on exitFrame me
  repeat with i = 48 down to 1
    sprite(i).puppet = FALSE
  end repeat
end
```

Example This handler branches the playback head to a specified frame if the value in the global variable `vTotal` exceeds 1000 when the playback head exits the frame:

```
global vTotal
on exitFrame
  if vTotal > 1000 then go to frame "Finished"
end
```

See also **on enterFrame**

exitLock

Syntax

```
the exitLock
```

Description

Movie property; determines whether a user can quit to the Windows desktop or Macintosh Finder from projectors (`TRUE`) or not (`FALSE`, default).

The user can quit to the desktop by pressing Control+period (Windows) or Command+period (Macintosh), Control+Q (Windows) or Command+Q (Macintosh), or Control+W (Windows) or Command+W (Macintosh); the Escape key is also supported in Windows.

This property can be tested and set.

Example

This statement sets the `exitLock` property to `TRUE`:

```
set the exitLock to TRUE
```

Example

Assuming that `exitLock` is set to `TRUE`, nothing occurs automatically when the Control+period/Q/W, Esc, or Command+period/Q/W keys are used. This handler checks keyboard input for keys to exit and takes the user to a custom quit sequence:

```
on checkExit
  if the commandDown and (the key = "." or the key = "q") and the
exitLock = TRUE then go to frame "quit sequence"
end
```

exit repeat

Syntax `exit repeat`

Description Keyword; instructs Lingo to leave a repeat loop and go to the statement following the `end repeat` statement but to remain within the current handler or method.

The `exit repeat` keyword is useful for breaking out of a repeat loop when a specified condition—such as two values being equal or a variable being a certain value—exists.

Example This handler searches for the position of the first vowel in a string represented by the variable `testString`. As soon as the first vowel is found, the `exit repeat` command instructs Lingo to leave the repeat loop and go to the statement `return i`:

```
"on findVowel testString
  repeat with i = 1 to testString.char[testString.char.count]
    if ""aeiou"" contains testString.char[i] then exit repeat
  end repeat
  return i
end"
```

See also **repeat while**, **repeat with**

exp()

Syntax

```
(integerOrFloat).exp
```

exp(integerOrFloat)

Description Function; calculates e, the natural logarithm base, to the power specified by *integerOrFloat*.

Example The following statement calculates the value of e to the exponent 5:

```
put (5).exp
-- 148.4132
```

externalEvent

Syntax `externalEvent "string"`

Description Command; sends a string to the browser that the browser can interpret as a scripting language instruction, allowing a movie playing or a browser to communicate with the HTML page in which it is embedded. The string sent by `externalEvent` must be in a scripting language supported by the browser.

This command works only for movies in browsers. To enable communication between an applet and a browser, use Java, JavaScript, or VBScript.

Note: The `externalEvent` command does not produce a return value. There is no immediate way to determine whether the browser handled the event or ignored it. Use `on EvalScript` within the browser to return a message to the movie.

Example These statements use `externalEvent` in the LiveConnect scripting environment, which is supported by Netscape 3.x and later.

LiveConnect evaluates the string passed by `externalEvent` as a function call. JavaScript authors must define and name this function in the HTML header. In the movie, the function name and parameters are defined as a string in `externalEvent`. Because the parameters must be interpreted by the browser as separate strings, each parameter is surrounded by single quotation marks.

Statements within HTML:

```
function MyFunction(parm1, parm2) {
  //script here
}
```

Statements within a script in the movie:

```
externalEvent ("MyFunction('parm1','parm2')")
```

Example These statements use `externalEvent` in the ActiveX scripting environment used by Internet Explorer in Windows. ActiveX treats `externalEvent` as an event and processes this event and its string parameter the same as an `onClick` event in a button object.

- Statements within HTML:

  In the HTML header, define a function to catch the event; this example is in VBScript:

  ```
  Sub
  NameOfShockwaveInstance_externalEvent(aParam)
    'script here
  End Sub
  ```

  Alternatively, define a script for the event:

  ```
  <SCRIPT FOR="NameOfShockwaveInstance"
  EVENT="externalEvent(aParam)"
  LANGUAGE="VBScript">
    'script here
  </SCRIPT>
  ```

  Within the movie, include the function and any parameters as part of the string for `externalEvent`:

  ```
  externalEvent ("MyFunction ('parm1','parm2')")
  ```

See also **on EvalScript**

externalParamCount()

Syntax `externalParamCount()`

Description Function; returns the number of parameters that an HTML <EMBED> or <OBJECT> tag is passing to a Shockwave movie.

This function is valid only for Shockwave movies that are running in a browser. It doesn't work for movies during authoring or for projectors.

Example This handler determines whether an <OBJECT> or <EMBED> tag is passing any external parameters to a Shockwave movie and runs Lingo statements if parameters are being passed:

```
if externalParamCount() > 0 then
  -- perform some action
end if
```

See also **externalParamName()**,**externalParamValue()**

externalParamName()

Syntax `externalParamName(n)`

Description Function; returns the name of a specific parameter in the list of external parameters from an HTML <EMBED> or <OBJECT> tag. This function is valid only for Shockwave movies that are running in a browser. It cannot be used with Director movies or projectors.

- If *n* is an integer, `externalParamName` returns the *n*th parameter name in the list.
- If *n* is a string, `externalParamName` returns `n` if any of the external parameter names matches *n*. The match is not case sensitive. If no matching parameter name is found, `externalParamName` returns `VOID`.

Example This statement places the value of a given external parameter in the variable `myVariable`:

```
if externalParamName ("swURLString") = "swURLString" then
  myVariable = externalParamValue ("swURLString")
end if
```

See also **externalParamCount()**,**externalParamValue()**

externalParamValue()

Syntax `externalParamValue(n)`

Description Function; returns a specific value from the external parameter list in an HTML <EMBED> or <OBJECT> tag. This function is valid only for Shockwave movies that are running in a browser. It can't be used with movies running in the authoring environment or projectors.

- If *n* is an integer, `externalParamValue` returns the *n*th parameter value from the external parameter list.
- If *n* is a string, `externalParamValue` returns the value associated with the first name that matches *n*. The match isn't case sensitive. If no such parameter value exists, `externalParamValue` returns `VOID`.

This function's behavior in an applet differs from that in other Director movies. In an applet, `externalParamValue` does the following:

- Returns the applet's parameters instead of the <EMBED> tag parameters.
- Accepts only string parameters.
- Returns a zero-length string rather than `VOID`.

See "Parameters for OBJECT and EMBED tags" and "Parameters accessible from Lingo" on the Director Developers Center Web site.

Example This statement places the value of an external parameter in the variable `myVariable`:

```
if externalParamName ("swURLString") = "swURLString" then
  myVariable = externalParamValue ("swURLString")
end if
```

See also **externalParamCount()**,**externalParamName()**

extractAlpha()

Syntax *imageObject*.extractAlpha()

Description This function copies the alpha channel from the given 32-bit image and returns it as a new image object. The result is an 8-bit grayscale image representing the alpha channel.

This function is useful for downsampling 32-bit images with alpha channels.

Example This statement places the alpha channel of the image of member 1 into the variable mainAlpha.

```
mainAlpha = member(1).image.extractAlpha()
```

See also **setAlpha()**, **useAlpha**

fadeIn()

Syntax
```
sound(channelNum).fadeIn({milliseconds})
fadeIn(sound(channelNum) {, milliseconds })
```

Description This function immediately sets the `volume` of sound channel *`channelNum`* to zero and then brings it back to the current volume over the given number of milliseconds. The default is 1000 milliseconds (1 second) value is given.

The current pan setting is retained for the the entire fade.

Example This Lingo fades in sound channel 3 over a period of 3 seconds from the beginning of cast member introMusic2.

```
sound(3).play(member("introMusic2"))
sound(3).fadeIn(3000)
```

See also **fadeOut()**,**fadeTo()**,**pan (sound property)**, **volume (sound channel)**

fadeOut()

Syntax `sound(channelNum).fadeOut({milliseconds})`

`fadeOut(sound(`*`channelNum`*`) {, `*`milliseconds`*` })`

Description This function gradually reduces the `volume` of sound channel *`channelNum`* to zero over the given number of milliseconds, or 1000 milliseconds (1 second) if no value is given.

The current pan setting is retained for the the entire fade.

Example This statement fades out sound channel 3 over a period of 5 seconds:

```
sound(3).fadeOut(5000)
```

See also **fadeIn()**,**fadeTo()**,**pan (sound property)**, **volume (sound channel)**

fadeTo()

Syntax

```
sound(channelNum).fadeTo(volume {, milliseconds })
fadeTo(sound(channelNum), volume {, milliseconds })
```

Description

This function gradually changes the `volume` of sound channel `channelNum` to the specified volume over the given number of milliseconds, or 1000 milliseconds (1 second) if no value is given. The range of values for volume is 0-255.

The current pan setting is retained for the the entire fade.

To see an example of `fadeTo()` used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement changes the volume of sound channel 4 to 150 over a period of 2 seconds. It can be a fade up or a fade down, depending on the original volume of sound channel 4 when the fade begins.

```
sound(4).fadeTo(150, 2000)
```

See also

fadeIn(), **fadeOut()**, **pan (sound property)**, **volume (sound channel)**

FALSE

Syntax `FALSE`

Description Constant; applies to an expression that is logically `FALSE`, such as 2 > 3. When treated as a number value, `FALSE` has the numerical value of 0. Conversely, 0 is treated as `FALSE`.

Example This statement turns off the `soundEnabled` property by setting it to `FALSE`:

```
the soundEnabled = FALSE
```

See also **if**, **not**, **TRUE**

field

Syntax `field whichField`

Description Keyword; refers to the field cast member specified by *`whichField`*.

- When *`whichField`* is a string, it is used as the cast member name.
- When *`whichField`* is an integer, it is used as the cast member number.

Character strings and chunk expressions can be read from or placed in the field.

The term `field` was used in earlier versions of Director and is maintained for backward compatibility. For new movies, use `member` to refer to field cast members.

Example This statement places the characters 5 through 10 of the field name entry in the variable `myKeyword`:

```
myKeyword = field("entry").char[5..10]
```

Example This statement checks whether the user entered the word *desk* and, if so, goes to the frame deskBid:

```
if member "bid" contains "desk" then go to "deskBid"
```

See also **char...of**, **item...of**, **line...of**, **word...of**

fieldOfView

Syntax `sprite(whichQTVRSprite).fieldOfView`

the fieldOfView of sprite *whichQTVRSprite*

Description QTVR sprite property; gives the specified sprite's current field of view in degrees.

This property can be tested and set.

fileName (cast property)

Syntax `castLib(whichCast).fileName`

`the fileName of castLib whichCast`

Description Property; specifies the file name of the specified cast.

- For an external cast, `fileName` gives the cast's full pathname and file name.
- For an internal cast, the `fileName` castLib property depends on which internal cast is specified.

For the first internal cast library, the `fileName` castLib property specifies the name of the movie. For remaining internal casts, `fileName` is an empty string.

The `fileName of castLib` property accepts URLs as references. However, to use a cast from the Internet and minimize download time, use the `downloadNetThing` or `preloadNetThing` command to download the cast's file to a local disk first and then set `fileName` castLib to the file on the disk.

If a movie sets the file name of an external cast, don't use the Duplicate Cast Members for Faster Loading option in the Projector Options dialog box.

This property can be tested and set for external casts. It can be tested only for internal casts.

Note: Director for Java does not support the downloadNetThing command.

Example This statement displays the pathname and file name of the Buttons external cast in the Message window:

```
put castLib("Buttons").fileName
```

Example This statement sets the file name of the Buttons external cast to Content.cst:

```
castLib("Buttons").fileName = the moviePath & "Content.cst"
```

The movie then uses the external cast file Content.cst as the Buttons cast.

Example These statements download an external cast from a URL to the Director application folder and then make that file the external cast named Cast of Thousands:

```
downLoadNetThing("http://www.cbDeMille.com Thousands.cst", the  \
applicationPath&"Thousands.cst")
castLib("Cast of Thousands").fileName = the applicationPath &
"Thousands.cst"
```

See also **downloadNetThing**, **preloadNetThing()**

fileName (cast member property)

Syntax

```
member(whichCastMember).fileName
the fileName of member whichCastMember
```

Description

Cast member property; refers to the name of the file assigned to the linked cast member specified by *whichCastMember*. This property is useful for switching the external linked file assigned to a cast member while a movie plays, similar to the way you can switch cast members. When the linked file is in a different folder than the movie, you must include the file's pathname.

You can also make unlinked media linked by setting the filename of those types of members that support linked media.

The `fileName` member property accepts URLs as a reference. However, to use a file from a URL and minimize download time, use the `downloadNetThing` or `preloadNetThing` command to download the file to a local disk first and then set `fileName` member property to the file on the local disk.

The Director player for Java doesn't support the `downLoadNetThing` command, so the player can't download files in the background before assigning a new file to a cast member. Changing the `fileName` member property in a movie playing as an applet can make the applet wait for the new file to download.

This property can be tested and set. After the file name is set, Director uses that file the next time the cast member is used.

Example

This statement links the QuickTime movie "ChairAnimation" to cast member 40:

```
member(40).fileName = "ChairAnimation"
```

These statements download an external file from a URL to the Director application folder and make that file the media for the sound cast member Norma Desmond Speaks:

```
downLoadNetThing("http://www.cbDeMille.com/ Talkies.AIF",the  \
applicationPath&"Talkies.AIF")
member("Norma Desmond Speaks").fileName = the applicationPath &
"Talkies.AIF"
```

See also

downloadNetThing, **preloadNetThing()**

fileName (window property)

Syntax

```
window whichWindow.fileName
```

```
the fileName of window whichWindow
```

Description

Window property; refers to the file name of the movie assigned to the window specified by *whichWindow*. When the linked file is in a different folder than the movie, you must include the file's pathname.

To be able to play the movie in a window, you must set the `fileName` window property to the movie's file name.

The `fileName of window` property accepts URLs as a reference. However, to use a movie file from a URL and minimize the download time, use the `downloadNetThing` or `preloadNetThing` command to download the movie file to a local disk first and then set `fileName` window property to the file on the local disk.

This property can be tested and set.

Example

This statement assigns the file named Control Panel to the window named Tool Box:

```
window("Tool Box").fileName = "Control Panel"
```

Example

This statement displays the file name of the file assigned to the window named Navigator:

```
put window("Navigator").fileName
```

Example

These statements download a movie file from a URL to the Director application folder and then assign that file to the window named My Close Up:

```
downLoadNetThing("http://www.cbDeMille.com/Finale.DIR",the  \
applicationPath&"Finale.DIR")
```

```
window("My Close Up").fileName = the applicationPath&"Finale.DIR"
```

See also

downloadNetThing, **preloadNetThing()**

fill()

Syntax

```
imageObject.fill(left, top, right, bottom,
colorObjectOrParameterList)

imageObject.fill(point(x, y), point(x, y),
colorObjectOrParameterList)

imageObject.fill(rect, colorObjectOrParameterList)
```

Description

This function fills a rectangular region with the color *colorObject* in the given image object. You specify the rectangle in any of the three ways shown. The points specified are relative to the upper-left corner of the given image object. The return value is 1 if there is no error, zero if there is an error.

If you provide a *parameterList* instead of a simple *colorObject*, the rectangle is filled with a shape you specify with these parameters:

Property	Description
#shapeType	A symbol value of #oval, #rect, #roundRect, or #line. The default is #line.
#lineSize	The width of the line to use in drawing the shape.
#color	A color object, which determines the fill color of the shape.
#bgColor	A color object, which determines the color of the shape's border.

For best performance, with 8-bit or lower images the *colorObject* should contain an indexed color value. For 16- or 32-bit images, use an RGB color value.

Example

This statement renders the image object in the variable *myImage* completely black:

```
myImage.fill(myImage.rect, rgb(0, 0, 0))
```

Example

This statement draws a filled oval in the image object TestImage. The oval has a green fill and a 5-pixel-wide red border.

```
TestImage.fill(0, 0, 100, 100, [#shapeType: #oval, #lineSize: 5,
#color: rgb(0, 255, 0), \
#bgColor: rgb(255, 0, 0)])
```

See also **color()**, **draw()**

fillColor

Syntax `member(whichCastMember).fillColor`

Description Vector shape cast member property; the color of the shape's fill specified as an RGB value.

It's possible to use `fillColor` when the `fillMode` property of the shape is set to `#solid` or `#gradient`, but not if it is set to `#none`. If the `fillMode` is `#gradient`, `fillColor` specifies the starting color for the gradient. The ending color is specified with `endColor`.

This property can be tested and set.

To see an example of `fillColor` used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement sets the fill color of the member Archie to a new RGB value.

```
member("Archie").fillColor = rgb( 24, 15, 153)
```

See also **endColor**, **fillMode**

fillCycles

Syntax `member(whichCastMember).fillCycles`

Description Vector shape cast member property; the number of fill cycles in a gradient vector shape's fill, as specified by an integer value from 1 to 7.

This property is valid only when the `fillMode` property of the shape is set to `#gradient`.

This property can be tested and set.

To see an example of `fillCycles` used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement sets the `fillCycles` of member Archie to 3.

```
member("Archie").fillCycles = 3
```

See also **endColor**, **fillColor**, **fillMode**

fillDirection

Syntax `member(whichCastMember).fillDirection`

Description Vector shape cast member property; specifies the amount in degrees to rotate the fill of the shape.

This property is only valid when the `fillMode` property of the shape is set to `#gradient`.

This property can be tested and set.

To see an example of `fillDirection` used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.

See also **fillMode**

filled

Syntax
```
member(whichCastMember).filled
the filled of member whichCastMember
```

Description
Shape cast member property; indicates whether the specified cast member is filled with a pattern (`TRUE`) or not (`FALSE`).

Example
The following statements make the shape cast member Target Area a filled shape and assign it the pattern numbered 1, which is a solid color:

```
member("Target Area").filled = TRUE
member("Target Area").pattern = 1
```

See also
filled

fillMode

Syntax `member(whichCastMember).fillMode`

Description Vector shape cast member property; indicates the fill method for the shape, using the following possible values:

- `#none`—The shape is transparent
- `#solid`—The shape uses a single fill color
- `#gradient`—The shape uses a gradient between two colors

This property can be tested and set when the shape is closed; open shapes have no fill.

To see an example of `fillMode` used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement sets the `fillMode` of member Archie to gradient.

```
member("Archie").fillMode = #gradient
```

See also **endColor**, **fillColor**

fillOffset

Syntax `member(whichCastMember).fillOffset`

Description Vector shape cast member property; specifies the horizontal and vertical amount in pixels (within the `defaultRect` space) to offset the fill of the shape.

This property is only valid when the `fillMode` property of the shape is set to `#gradient`, but can be both tested and set.

To see an example of `fillOffset` used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement changes the fill offset of the vector shape cast member miette to a horizontal offset of 33 pixels and a vertical offset of 27 pixels:

```
member("miette").fillOffset = point(33, 27)
```

See also **defaultRect**, **fillMode**

fillScale

Syntax `member(whichCastMember).fillScale`

Description Vector shape cast member property; specifies the amount to scale the fill of the shape. This property is referred to as “spread” in the vector shape window.

This property is only valid when the `fillMode` property of the shape is set to `#gradient`, but can be both tested and set.

To see an example of `fillScale` used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement sets the `fillScale` of member Archie to 33.

```
member("Archie").fillScale = 33.00
```

See also **fillMode**

findEmpty()

Syntax `findEmpty(member whichCastMember)`

Description Function; for the current cast only, displays the next empty cast member position or the position after the cast member specified by *whichCastMember*.

Example This statement finds the first empty cast member on or after cast member 100:

```
put findEmpty(member 100)
```

findLabel()

Syntax `sprite(whichFlashSprite).findLabel(whichLableName)`

findLabel(sprite whichFlashSprite, whichLableName)

Description Function: this function returns the frame number (within the Flash movie) that is associated with the label name requested.

A 0 is returned if the label doesn't exist, or if that portion of the Flash movie has not yet been streamed in.

findPos

Syntax `list.findPos(property)`

findPos(list, property)

Description List command; identifies the position of the property specified by *`property`* in the property list specified by *`list`*.

Using `findPos` with linear lists returns a bogus number if the value of *`property`* is a number and a script error if the value of *`property`* is a string.

The `findPos` command performs the same function as the `findPosNear` command, except that `findPos` is `VOID` when the specified property is not in the list.

Example This statement identifies the position of the property c in the list Answers, which consists of [#a:10, #b:12, #c:15, #d:22]:

```
Answers.findPos(#c)
```

The result is 3, because c is the third property in the list.

See also **findPosNear**, **sort**

findPosNear

Syntax `sortedList.findPosNear(valueOrProperty)`

findPosNear(sortedList, valueOrProperty)

Description List command; for sorted lists only, identifies the position of the item specified by *`valueOrProperty`* in the specified sorted list.

The `findPosNear` command works only with sorted lists. Replace *`valueOrProperty`* with a value for sorted linear lists, and with a property for sorted property lists.

The `findPosNear` command is similar to the `findPos` command, except that when the specified property is not in the list, the `findPosNear` command identifies the position of the value with the most similar alphanumeric name. This command is useful in finding the name that is the closest match in a sorted directory of names.

Example This statement identifies the position of a property in the sorted list Answers, which consists of [#Nile:2, #Pharaoh:4, #Raja:0]:

```
Answers.findPosNear(#Ni)
```

The result is 1, because Ni most closely matches Nile, the first property in the list.

See also **findPos**

finishIdleLoad

Syntax `finishIdleLoad` *`loadTag`*

Description Command; forces completion of loading for all the cast members that have the specified load tag.

Example This statement completes the loading of all cast members that have the load tag 20:

```
finishIdleLoad 20
```

See also **idleHandlerPeriod**, **idleLoadDone()**,**idleLoadMode**, **idleLoadPeriod**, **idleLoadTag**, **idleReadChunkSize**

firstIndent

Syntax `chunkExpression.firstIndent`

Description Text cast member property; contains the number of pixels the first indent in *`chunkExpression`* is offset from the left margin of the *`chunkExpression`*.

The value is an integer: less than 0 indicates a hanging indent, 0 is no indention, and greater than 0 is a normal indention.

This property can be tested and set.

Example This statement sets the indent of the first line of member Desk to 0 pixels.

```
member("Desk").firstIndent = 0
```

See also **leftIndent**, **rightIndent**

fixedLineSpace

Syntax `chunkExpression.fixedLineSpace`

Description Text cast member property; controls the height of each line in the *chunkExpression* portion of the text cast member.

The value itself is an integer, indicating height in absolute pixels of each line.

The default value is 0, which results in natural height of lines.

Example This statement sets the height in pixels of each line of member Desk to 24:

```
member("Desk").fixedLineSpace = 24
```

fixedRate

Syntax

```
sprite(whichFlashOrGIFSprite).fixedRate
the fixedRate of sprite whichFlashOrGIFSprite
member(whichFlashOrGIFMember).fixedRate
the fixedRate of member whichFlashOrGIFMember
```

Description

Cast member property and sprite property; controls the frame rate of a Flash movie or animated GIF. The `fixedRate` property can have integer values. The default value is 15.

This property is ignored if the sprite's `playbackMode` property is anything other than `#fixed`.

This property can be tested and set.

Example

This handler adjusts the frame rate of a Flash movie sprite. As parameters, the handler accepts a sprite reference, an indication of whether to speed up or slow down the Flash movie, and the amount to adjust the speed.

```
on adjustFixedRate whichSprite, adjustType, howMuch
  case adjustType of
    #faster:
      sprite(whichSprite).fixedRate =
sprite(whichSprite).fixedRate + howMuch
    #slower:
      sprite(whichSprite).fixedRate =
sprite(whichSprite).fixedRate - howMuch
  end case
end
```

See also

playBackMode

fixStageSize

Syntax

```
the fixStageSize
```

Description

Movie property; determines whether the Stage size remains the same when you load a new movie (`TRUE`, default), or not (`FALSE`), regardless of the Stage size saved with that movie, or the setting for the centerStage.

The `fixStageSize` property cannot change the Stage size for a movie that is currently playing.

This property can be tested and set.

Example

This statement determines whether the `fixStageSize` property is turned on. If `fixStageSize` is `FALSE`, it sends the playback head to a specified frame.

```
if the fixStageSize = FALSE then go to frame "proper size"
```

This statement sets the `fixStageSize` property to the opposite of its current setting:

```
the fixStageSize = not the fixStageSize
```

See also

centerStage

flashRect

Syntax

```
member(whichVectorOrFlashMember).flashRect
the flashRect of member whichVectorOrFlashMember
```

Description

Cast member property; indicates the size of a Flash movie or vector shape cast member as it was originally created. The property values are indicated as a Director rectangle: for example, rect(0,0,32,32).

For linked Flash cast members, the `FlashRect` member property returns a valid value only when the cast member's header has finished loading into memory.

This property can be tested but not set.

Example

This sprite script resizes a Flash movie sprite so that it is equal to the original size of its Flash movie cast member:

```
on beginSprite me
  sprite(me.spriteNum).rect =
sprite(me.spriteNum).member.FlashRect
end
```

See also

defaultRect, **defaultRectMode**, **state**

flashToStage()

Syntax `sprite(whichFlashSprite).flashToStage(pointInFlashMo vie)`

flashToStage (sprite whichFlashSprite, pointInFlashMovie)

Description Function; returns the coordinate on the Director Stage that corresponds to a specified coordinate in a Flash movie sprite. The function accepts both the Flash channel and movie coordinate and returns the Director Stage coordinate as Director point values: for example, point(300,300).

Flash movie coordinates are measured in Flash movie pixels, which are determined by a movie's original size when it was created in Flash. For the purpose of calculating Flash movie coordinates, point(0,0) of a Flash movie is always at its upper left corner. (The cast member's `originPoint` property is used only for rotation and scaling, not to calculate movie coordinates.)

The `flashToStage` and the corresponding `stageToFlash` functions are helpful for determining which Flash movie coordinate is directly over a Director Stage coordinate. For both Flash and Director, point(0,0) is the upper left corner of the Flash Stage or Director Stage. These coordinates may not match on the Director Stage if a Flash sprite is stretched, scaled, or rotated.

Example This handler accepts a point value and a sprite reference as a parameter, and it then sets the upper left coordinate of the specified sprite to the specified point within a Flash movie sprite in channel 10:

```
on snapSprite whichFlashPoint, whichSprite
  sprite(whichSprite).loc =
sprite(1).FlashToStage(whichFlashPoint)
  updatestage
end
```

See also **stageToFlash()**

flipH

Syntax
```
sprite(whichSpriteNumber).flipH
```

the flipH of sprite *whichSpriteNumber*

Description
Sprite property; indicates whether a sprite's image has been flipped horizontally on the Stage (`TRUE`) or not (`FALSE`).

The image itself is flipped around its registration point.

This means any rotation or skew remains constant; only the image data itself is flipped.

Example
This statement displays the `flipH` of sprite 5:

```
put sprite (5).flipH
```

See also
flipV, **rotation**, **skew**

flipV

Syntax `sprite(whichSpriteNumber).flipV`

the flipV of sprite *whichSpriteNumber*

Description Sprite property; indicates whether a sprite's image has been flipped vertically on the Stage (`TRUE`) or not (`FALSE`).

The image itself is flipped around its registration point.

This means any rotation or skew remains constant; only the image data itself is flipped.

Example This statement displays the `flipV` of sprite 5:

```
sprite (5).flipV = 1
```

See also **flipH**, **rotation**, **skew**

float()

Syntax `(expression).float`

float (expression)

Description Function; converts an expression to a floating-point number. The number of digits that follow the decimal point (for display purposes only, calculations are not affected) is set using the `floatPrecision` property.

Example This statement converts the integer 1 to the floating-point number 1:

```
put (1).float
-- 1.0
```

Example Math operations can be performed using `float`; if any of the terms is a `float` value, the entire operation is performed with `float`:

```
"the floatPrecision = 1
put 2 + 2
-- 4
put (2).float + 2
-- 4.0
the floatPrecision = 4
put 22/7
-- 3
put (22).float / 7
-- 3.1429"
```

See also **floatPrecision**, **ilk()**

floatP()

Syntax

```
(expression).floatP
```

floatP(expression)

Description

Function; indicates whether the value specified by *`expression`* is a floating- point number (1 or `TRUE`) or not (0 or `FALSE`).

The *P* in `floatP` stands for *predicate*.

Example

This statement tests whether 3.0 is a floating-point number. The Message window displays the number 1, indicating that the statement is `TRUE`.

```
put (3.0).floatP
-- 1
```

This statement tests whether 3 is a floating-point number. The Message window displays the number 0, indicating that the statement is `FALSE`.

```
put (3).floatP
-- 0
```

See also

float(),**ilk()**,**integerP()**,**objectP()**,**stringP()**,**symbolP()**

floatPrecision

Syntax `the floatPrecision`

Description Movie property; rounds off the display of floating-point numbers to the number of decimal places specified. The value of `floatPrecision` must be an integer. The maximum value is 15 significant digits; the default value is 4.

The `floatPrecision` property determines only the number of digits used to display floating-point numbers; it does not change the number of digits used to perform calculations.

- If `floatPrecision` is a number from 1 to 15, floating-point numbers display that number of digits after the decimal point. Trailing zeros remain.
- If `floatPrecision` is zero, floating-point numbers are rounded to the nearest integer. No decimal points appear.
- If `floatPrecision` is a negative number, floating-point numbers are rounded to the absolute value for the number of decimal places. Trailing zeros are dropped.

This property can be tested and set.

Example This statement rounds off the square root of 3.0 to three decimal places:

```
the floatPrecision = 3
x = sqrt(3.0)
put x
-- 1.732
```

Example This statement rounds off the square root of 3.0 to eight decimal places:

```
the floatPrecision = 8
put x
-- 1.73205081
```

flushInputEvents

Syntax `flushInputEvents()`

Description This command will flush any waiting mouse or keyboard events from Director's message queue. Generally this is useful when Lingo is in a tight repeat loop and the author wants to make sure any mouse clicks or keyboard presses don't get through. This command operates at runtime only and has no effect during authoring.

Example This Lingo disables mouse and keyboard events while a repeat loop executes:

```
repeat with i = 1 to 10000
  flushInputEvents()
  sprite(1).loc = sprite(1).loc + point(1, 1)
end repeat
```

See also **the mouseDown (system property)**, **the mouseUp (system property)**, **on keyDown**, **on keyUp**

font

Syntax

```
member(whichCastMember).font
the font of member whichCastMember
```

Description

Text and field cast member property; determines the font used to display the specified cast member and requires that the cast member contain characters, if only a space. The parameter *whichCastMember* can be either a cast member name or number.

The Director player for Java doesn't map to other fonts when converting a movie; Java substitutes the default font for any unsupported font. Use only Java's supported fonts as values for the `font` member property in a movie that plays back as an applet.

Java offers only the following cross-platform fonts:

Java font	Windows font	Macintosh font
Helvetica	Arial	Helvetica
TimesRoman	Times New Roman	Times
Courier	Courier-New	Courier
Dialog	MS Sans Serif	Chicago or Charcoal
DialogInput	MS Sans Serif	Geneva
ZapfDingbats	WingDings	Zapf Dingbats
Default	Arial	Helvetica

The `font` member property can be tested and set.

To see an example of `font` used in a completed movie, see the Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement sets the variable named `oldFont` to the current `font` setting for the field cast member Rokujo Speaks:

```
oldFont = member("Rokujo Speaks").font
```

See also

text; **alignment**, **fontSize**, **fontStyle**, **lineHeight (cast member property)**

fontSize

Syntax
```
member(whichCastMember).fontSize
the fontSize of member whichCastMember
```

Description
Field cast member property; determines the size of the font used to display the specified field cast member and requires that the cast member contain characters, if only a space. The parameter *whichCastMember* can be either a cast member name or number.

This property can be tested and set. When tested, it returns the height of the first line in the field. When set, it affects every line in the field.

To see an example of `fontSize` used in a completed movie, see the Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example
This statement sets the variable named `oldSize` to the current `fontSize of member` setting for the field cast member Rokujo Speaks:

```
oldSize = member("Rokujo Speaks").fontSize
```

Example
This statement sets the third line of the text cast member myMenu to 24 points:

```
member("myMenu").fontSize = 12
```

See also
text; **alignment**, **font**, **lineHeight (cast member property)**

fontStyle

Syntax

```
member(whichCastMember).fontStyle
the fontStyle of member whichCastMember
member(whichCastMember).char[whichChar].fontStyle
the fontStyle of char whichChar
member(whichCastMember).word[whichWord].fontStyle
the fontStyle of word whichWord
member(whichCastMember).line[whichLine].fontStyle
the fontStyle of line whichLine
```

Description

Cast member property; determines the styles applied to the font used to display the specified field cast member, character, line, word, or other chunk expression and requires that the field cast member contain characters, if only a space.

The value of the property is a string of styles delimited by commas. Lingo uses a font that is a combination of the styles in the string. The available styles are plain, bold, italic, underline, shadow, outline, and extended; on the Macintosh, condensed also is available.

Use the style plain to remove all currently applied styles. The parameter *whichCastMember* can be either a cast member name or number.

For a movie playing back as an applet, plain, bold, and italic are the only valid styles for the `fontStyle` member property. The Director player for Java doesn't support underline, shadow, outline, extended, or condensed font styles.

This property can be tested and set.

To see an example of `fontStyle` used in a completed movie, see the Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement sets the variable named `oldStyle` to the current `fontStyle` setting for the field cast member Rokujo Speaks:

```
oldStyle = member("Rokujo Speaks").fontStyle
```

Example

This statement sets the `fontStyle` member property for the field cast member Poem to bold italic:

```
member("Poem").fontStyle = [#bold, #italic]
```

Example

This statement sets the `fontStyle` property of the third word of the cast member Son's Names to italic:

```
member("Son's Names").word[3].fontStyle = [#italic]
```

Example

This statement sets the `fontStyle` member property of word 1 through word 4 of text member myNote to bold italic:

```
member("myNote").word[1..4].fontstyle = [#bold, #italic]
```

See also

text; **alignment**, **font**, **lineHeight (cast member property)**, **fontSize**

foreColor

Syntax
```
member(castName).foreColor = colorNumber
set the foreColor of member castName to colorNumber
sprite whichSprite.foreColor
the foreColor of sprite whichSprite
```

Description Cast member property; sets the foreground color of a field cast member.

For a movie that plays back as an applet, specify colors for the `foreColor` sprite property as the decimal equivalent of the 24-bit hexadecimal values used in an HTML document.

It is not recommended to apply this property to bitmap cast members deeper than 1-bit, as the results are difficult to predict.

To see an example of `foreColor` used in a completed movie, see the Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example The hexadecimal value for pure red, FF0000, is equivalent to 16711680 in decimal numbers. This statement specifies pure red as a cast member's forecolor:

```
member(20).foreColor = 16711680
```

Example This statement changes the color of the field in cast member 1 to the color in palette entry 250:

```
member(1).foreColor = 250
```

Example The following statement sets the variable `oldColor` to the foreground color of sprite 5:

```
oldColor = sprite(5).foreColor
```

Example The following statement makes 36 the number for the foreground color of a random sprite from sprites 11 to 13.

```
sprite(10 + random(3)).foreColor = 36
```

Example The following statement sets the `foreColor` of word 3 of line 2 of text member myDescription to a value of 27:

```
member("myDescription").line[2].word[3].forecolor = 27
```

See also **backColor**, **color (sprite property)**

forget()

Syntax `timeout("timeoutName").forget()`

`forget(timeout("timeoutName"))`

Description This timeout object function removes the given *`timeoutObject`* from `the timeoutList`, and prevents it from sending further timeout events.

Example This statement deletes the timeout object named AlarmClock from `the timeoutList`:

```
timeout("AlarmClock").forget()
```

See also **timeout()**,**timeoutHandler**, **timeoutList**, **new()**

forget window

Syntax `window(whichWindow).forget()`

forget window *whichWindow*

Description Window property; instructs Lingo to close and delete the window specified by *whichWindow* when it's no longer in use and no other variables refer to it.

When a `forget window` command is given, the window and the movie in a window (MIAW) disappear without calling the `on stopMovie`, `on closeWindow`, or `on deactivateWindow` handlers.

If there are many global references to the movie in a window, the window doesn't respond to the `forget` command.

Example This statement instructs Lingo to delete the window Control Panel when the movie no longer uses the window:

```
window("Control Panel").forget()
```

See also **close window**, **open window**

frame() (function)

Syntax `the frame`

Description Function; returns the number of the current frame of the movie.

Example This statement sends the playback head to the frame before the current frame:

```
go to (the frame - 1)
```

See also **go**, **label()**,**marker()**

frame (sprite property)

Syntax

```
sprite(whichFlashSprite).frame
```

the frame of sprite *whichFlashSprite*

Description

Sprite property; controls which frame of the current Flash movie is displayed. The default value is 1.

This property can be tested and set.

Example

This frame script checks to see if a Flash movie has finished playing (by checking to see if the current frame is equal to the total number of frames in the movie). If the movie has not finished, the playback head continues to loop in the current frame; when the movie finishes, the playback head continues to the next frame. (This script assumes that the movie was designed to stop on its final frame and that it has not been set for looped playback.)

```
on exitFrame
  if sprite(5).frame < sprite(5).member.frameCount then
    go to the frame
  end if
end
```

frameCount

Syntax

```
member(whichFlashMember).frameCount
the frameCount of member whichFlashMember
```

Description

Flash cast member property; indicates the number of frames in the Flash movie cast member. The `frameCount` member property can have integer values.

This property can be tested but not set.

Example

This sprite script displays, in the Message window, the channel number and the number of frames in a Flash movie.

```
"property spriteNum
on beginSprite me
  put ""The Flash movie in channel"" && spriteNum && has"" &&
sprite(spriteNum).member.frameCount && ""frames.""
end"
```

frameLabel

Syntax `the frameLabel`

Description Frame property; identifies the label assigned to the current frame. When the current frame has no label, the value of the `frameLabel` property is 0.

This property can be tested at any time. It can be set during a Score generation session.

Example This statement checks the label of the current frame. In this case, the current `frameLabel` value is Start:

```
put the frameLabel
-- "Start"
```

See also **labelList**

framePalette

Syntax
```
the framePalette
```

Description
Frame property; identifies the cast member number of the palette used in the current frame, which is either the current palette or the palette set in the current frame.

Because the browser controls the palette for the entire Web page, the Director player for Java always uses the browser's palette. For the most reliable color when authoring a movie for playback as a Director player for Java, use the default palette for the authoring system. When you want exact control over colors, use Shockwave instead of Java.

This property can be tested. It can also be set during a Score generation session.

Example
This statement checks the palette used in the current frame. In this case, the palette is cast member 45.

```
put the framePalette
-- 45
```

Example
This statement makes palette cast member 45 the palette for the current frame:

```
the framePalette = 45
```

See also
puppetPalette

frameRate

Syntax
```
member(whichCastMember).frameRate
the frameRate of member whichCastMember
```

Description Cast member property; specifies the playback frame rate for the specified digital video, or Flash movie cast member.

The possible values for the frame rate of a digital video member correspond to the radio buttons for selecting digital video playback options.

- When the `frameRate` member property is between 1 and 255, the digital video movie plays every frame at that frame rate. The `frameRate` member property cannot be greater than 255.
- When the `frameRate` member property is set to -1 or 0, the digital video movie plays every frame at its normal rate. This allows the video to sync to its soundtrack. When the frameRate is set to any value other than -1 or 0, the digital video soundtrack will not play.
- When the `frameRate` member property is set to -2, the digital video movie plays every frame as fast as possible.

For Flash movie cast members, the property indicates the frame rate of the movie created in Flash.

This property can be tested but not set.

Example This statement sets the frame rate of the QuickTime digital video cast member Rotating Chair to 30 frames per second:

```
member("Rotating Chair").frameRate = 30
```

Example This statement instructs the QuickTime digital video cast member Rotating Chair to play every frame as fast as possible:

```
member("Rotating Chair").frameRate = -2
```

Example This sprite script checks to see if the sprite's cast member was originally created in Flash with a frame rate of less than 15 frames per second. If the movie's frame rate is slower than 15 frames per second, the script sets the `playBackMode` property for the sprite so it can be set to another rate. The script then sets the sprite's `fixedRate` property to 15 frames per second.

```
property spriteNum

on beginSprite me
  if sprite(spriteNum).member.frameRate < 15 then
    sprite(spriteNum).playBackMode = #fixed
    sprite(spriteNum).fixedRate = 15
  end if
end
```

See also **fixedRate**, **movieRate**, **movieTime**, **playBackMode**

frameReady()

Syntax

```
frameReady(frameN)
frameReady(frameN, frameZ)
frameReady()
frameReady(sprite whichFlashSprite, frameNumber)
```

Description

Function; for a Flash movie, determines whether a streaming movie is ready for display. If enough of a sprite has streamed into memory to render the frame (integer for frame number, string for label) specified in the `frameNumber` parameter, this function is `TRUE`; otherwise it is `FALSE`. For a Director movie, this function determines whether all the cast members for *`frameN`* (the number of the frame) are downloaded from the Internet and available locally.

This function is useful only when streaming a movie, range of frames, cast, or linked cast member. To activate streaming, set the Movie:Playback properties in the Modify menu to Use Media as Available or Show Placeholders.

For Director movies, projectors, and Shockwave movies:

- `frameReady (`*`frameN`*`)`—Determines whether the cast members for *`frameN`* are downloaded.
- `frameReady (`*`frameN, frameZ`*`)`—Determines whether the cast members for *`frameN`* through *`frameZ`* are downloaded.
- `frameReady()`—Determines if cast member used in any frame of the Score are downloaded.

For a demonstration of the `frameReady` function used with a Director movie, see the **frameReady** movie.

This function can be tested but not set.

Example

This statement determines whether the cast members for frame 20 are downloaded and ready to be viewed:

```
on exitFrame
  if frameReady(20) then
    -- go to frame 20 if all the required
    --castmembers are                    locally available
    go to frame 20
  else
    -- resume animating loop while background
    --is streaming
    got to frame 1
  end if
end
```

Example

This frame script checks to see if frame 25 of a Flash movie sprite in channel 5 can be rendered. If it can't, the script keeps the playback head looping in the current frame of the Director movie. When frame 25 can be rendered, the script starts the movie and lets the playback head proceed to the next frame of the Director movie.

```
on exitFrame
  if the frameReady(sprite 5, 25) = FALSE then
    go to the frame
  else
    play sprite 5
  end if
end
```

See also **mediaReady**

frameScript

Syntax
```
the frameScript
```

Description
Frame property; contains the unique cast member number of the frame script assigned to the current frame.

The `frameScript` property can be tested. During a Score recording session, you can also assign a frame script to the current frame by setting the `frameScript` property.

Example
This statement displays the number of the script assigned to the current frame. In this case, the script number is 25.

```
put the frameScript
-- 25
```

Example
This statement makes the script cast member Button responses the frame script for the current frame:

```
the frameScript = member "Button responses"
```

frameSound1

Syntax
```
the frameSound1
```

Description
Frame property; determines the number of the cast member assigned to the first sound channel in the current frame.

This property can be tested and set. This property can also be set during a Score recording session.

Example
As part of a Score recording session, this statement assigns the sound cast member Jazz to the first sound channel:

```
the frameSound1 = member("Jazz").number
```

frameSound2

Syntax
```
the frameSound2
```

Description
Frame property; determines the number of the cast member assigned to the second sound channel for the current frame.

This property can be tested and set. This property can also be set during a Score recording session.

Example
As part of a Score recording session, this statement assigns the sound cast member Jazz to the second sound channel:

```
the frameSound2 = member("Jazz").number
```

framesToHMS()

Syntax `framesToHMS(frames, tempo, dropFrame, fractionalSeconds)`

Description Function; converts the specified number of frames to their equivalent length in hours, minutes, and seconds. This function is useful for predicting the actual playtime of a movie or controlling a video playback device.

- *frames*—Integer expression that specifies the number of frames.
- *tempo*—Integer expression that specifies the tempo in frames per second.
- *dropFrame*—Compensates for the color NTSC frame rate, which is not exactly 30 frames per second and is meaningful only if FPS is set to 30 frames per second. Normally, this argument is set to `FALSE`.
- *fractionalSeconds*—Determines whether the residual frames are converted to the nearest hundredth of a second (`TRUE`) or returned as an integer number of frames (`FALSE`).

The resulting string uses the form `sHH:MM:SS.FFD`, where:

s	A character is used if the time is less than zero, or a space if the time is greater than or equal to zero.
HH	Hours.
MM	Minutes.
SS	Seconds.
FF	Indicates a fraction of a second if *fractionalSeconds* is TRUE or frames if *fractionalSeconds* is FALSE.
D	A "d" is used if *dropFrame* is TRUE, or a space if *dropFrame* is FALSE.

Example This statement converts a 2710-frame, 30 frame-per-second movie. The *dropFrame* and *fractionalSeconds* arguments are both turned off:

```
put framesToHMS(2710, 30, FALSE, FALSE)
-- " 00:01:30.10 "
```

See also **HMStoFrames()**

frameTempo

Syntax `the frameTempo`

Description Frame property; indicates the tempo assigned to the current frame.

This property can be tested. It can be set during a Score recording session.

Example This statement checks the tempo used in the current frame. In this case, the tempo is 15 frames per second.

```
put the frameTempo
-- 15
```

See also **puppetTempo**

frameTransition

Syntax `the frameTransition`

Description Frame property; specifies the number of the transition cast member assigned to the current frame.

This property can be set during a Score recording session to specify transitions.

Example When used in a Score recording session, this statement makes the cast member Fog the transition for the frame that Lingo is currently recording:

```
set the frameTransition to member "Fog"
```

freeBlock()

Syntax `the freeBlock`

Description Function; indicates the size of the largest free contiguous block of memory, in bytes. A kilobyte (K) is 1024 bytes. A megabyte (MB) is 1024 kilobytes. Loading a cast member requires a free block at least as large as the cast member.

Example This statement determines whether the largest contiguous free block is smaller than 10K and displays an alert if it is:

```
if (the freeBlock < (10 * 1024)) then alert "Not enough memory!"
```

See also **freeBytes()**,**memorySize**, **ramNeeded()**,**size**

freeBytes()

Syntax `the freeBytes`

Description Function; indicates the total number of bytes of free memory, which may not be contiguous. A kilobyte (K) is 1024 bytes. A megabyte (MB) is 1024 kilobytes.

This function differs from `freeBlock` in that it reports all free memory, not just contiguous memory.

On the Macintosh, selecting Use System Temporary Memory in the Director General Preferences or in a projector's Options dialog box tells the `freeBytes` function to return all the free memory that is available to the application. This amount equals the application's allocation shown in its Get Info dialog box and the Largest Unused Block value in the About This Macintosh dialog box.

Example This statement checks whether more than 200K of memory is available and plays a color movie if it is:

```
if (the freeBytes > (200 * 1024)) then play movie "colorMovie"
```

See also **freeBlock()**,**memorySize**, **objectP()**,**ramNeeded()**,**size**

frontWindow

Syntax `the frontWindow`

Description System property; indicates which movie in a window (MIAW) is currently frontmost on the screen. When the Stage is frontmost, front window is `the Stage`. When a media editor or floating palette is frontmost, `frontWindow` returns `VOID`.

This property can be tested but not set.

Example This statement determines whether the window "Music" is currently the frontmost window and, if it is, brings the window "Try This" to the front:

```
if the frontWindow = "Music" then window("Try This").moveToFront
```

See also **activeWindow**, **on activateWindow**, **on deactivateWindow**, **moveToFront**

getaProp

Syntax
```
propertyList.propertyName
getaProp(list, item)
list[listPosition]
propertyList [ #propertyName ]
propertyList [ "propertyName" ]
```

Description List command; for linear and property lists, identifies the value associated with the item specified by *item*, *listPosition*, or *propertyName* in the list specified by *list*.

- When the list is a linear list, replace *item* with the number for an item's position in a list as shown by *listPosition*. The result is the value at that position.
- When the list is a property list, replace *item* with a property in the list as in *propertyName*. The result is the value associated with the property.

The `getaProp` command returns `VOID` when the specified value is not in the list.

When used with linear lists, the `getaProp` command has the same function as the `getAt` command.

Example This statement identifies the value associated with the property #joe in the property list ages, which consists of [#john:10, #joe:12, #cheryl:15, #barbara:22]:

```
put getaProp(ages, #joe)
```

The result is 12, because this is the value associated with the property #joe.

Example The same result can be achieved using bracket access on the same list:

```
put ages[#joe]
```

The result is again 12.

Example If you want the value at a certain position in the list, you can also use bracket access. To get the third value in the list, associated with the third property, use this syntax:

```
put ages[3]
-- 15
```

> Note: Unlike the `getAProp` command where `VOID` is returned when a property doesn't exist, a script error will occur if the property doesn't exist when using bracket access.

See also **getAt**, **getOne()**,**getProp()**,**setaProp**, **setAt**

getAt

Syntax

```
getAt(list, position)
list [position]
```

Description List command; identifies the item in the position specified by *position* in the specified list. If the list contains fewer elements than the specified position, a script error occurs.

The `getAt` command works with linear and property lists. This command has the same function as the `getaProp` command for linear lists.

This command is useful for extracting a list from within another list, such as the deskTopRectList.

Example This statement causes the Message window to display the third item in the answers list, which consists of [10, 12, 15, 22]:

```
put getAt(answers, 3)
-- 15
```

Example The same result can be returned using bracket access:

```
put answers[3]
-- 15
```

Example This example extracts the first entry in a list containing two entries that specify name, department, and employee number information. Then the second element of the newly extracted list is returned, identifying the department in which the first person in the list is employed. The format of the list is [[“Dennis”, “consulting”, 510], [“Sherry”, “Distribution”, 973]], and the list is called employeeInfoList.

```
firstPerson = getAt(employeeInfoList, 1)
put firstPerson
-- ["Dennis", "consulting", 510]
firstPersonDept = getAt(firstPerson, 2)
put firstPersonDept
-- "consulting"
```

Example It’s also possible to nest `getAt` commands without assigning values to variables in intermediate steps. This format can be more difficult to read and write, but less verbose.

```
firstPersonDept = getAt(getAt(employeeInfoList, 1), 2)
put firstPersonDept
 -- "consulting"
```

Example You can also use the bracket list access:

```
firstPerson = employeeInfoList[1]
put firstPerson
-- ["Dennis", "consulting", 510]
firstPersonDept = firstPerson[2]
put firstPersonDept
-- "consulting"
```

Example As with `getAt`, brackets can be nested:

```
firstPersonDept = employeeInfoList[1][2]
```

See also **getaProp**, **setaProp**, **setAt**

on getBehaviorDescription

Syntax

```
on getBehaviorDescription
  statement(s)
end
```

Description

System message and event handler; contains Lingo that returns the string that appears in a behavior's description pane in the Behavior Inspector when the behavior is selected.

The description string is optional.

Director sends the `getBehaviorDescription` message to the behaviors attached to a sprite when the Behavior Inspector opens. Place the `on getBehaviorDescription` handler within a behavior.

The handler can contain embedded Return characters for formatting multiple- line descriptions.

Example

This statement displays "Vertical Multiline textField Scrollbar" in the description pane.

```
on getBehaviorDescription
  return "Vertical Multiline textField Scrollbar"
end
```

See also

on getPropertyDescriptionList, **on getBehaviorTooltip**, **on runPropertyDialog**

on getBehaviorTooltip

Syntax

```
on getBehaviorTooltip
  statement(s)
end
```

Description

System message and event handler; contains Lingo that returns the string that appears in a tooltip for a script in the Library palette.

Director sends the `getBehaviorTooltip` message to the script when the cursor stops over it in the Library palette. Place the `on getBehaviorTooltip` handler within the behavior.

The use of the handler is optional. If no handler is supplied, the cast member name appears in the tooltip.

The handler can contain embedded Return characters for formatting multiple- line descriptions.

Example

This statement displays “Jigsaw puzzle piece” in the description pane.

```
on getBehaviorTooltip
  return "Jigsaw puzzle piece"
end
```

See also

on getPropertyDescriptionList, **on getBehaviorDescription**, **on runPropertyDialog**

getError()

Syntax

```
member(whichSWAmember).getError()
getError(member whichSWAmember)
member(whichFlashmember).getError()
getError(member whichFlashmember)
```

Description

Function; for Shockwave Audio (SWA) or Flash cast members, indicates whether an error occurred as the cast member streamed into memory and returns a value.

Shockwave Audio cast members have the following possible `getError()` integer values and corresponding `getErrorString()` messages:

getError() value	**getErrorString() message**
0	OK
1	memory
2	network
3	playback device
99	other

Flash movie cast members have the following possible `getError` values:

- `FALSE`—No error occurred.
- `#memory`—There is not enough memory to load the cast member.
- `#fileNotFound`—The file containing the cast member's assets could not be found.
- `#network`—A network error prevented the cast member from loading.
- `#fileFormat`—The file was found, but it appears to be of the wrong type, or an error occurred while reading the file.
- `#other`—Some other error occurred.

When an error occurs as a cast member streams into memory, Director sets the cast member's state property to -1. Use the `getError` function to determine what type of error occurred.

Example

This handler uses `getError` to determine whether an error involving the Shockwave Audio cast member Norma Desmond Speaks occurred and displays the appropriate error string in a field if it did:

```
on exitFrame
  if member("Norma Desmond Speaks").getError() <> 0 then
    member("Display Error Name").text = member("Norma Desmond \
Speaks").getErrorString()
  end if
end
```

Example

This handler checks to see whether an error occurred for a Flash cast member named Dali, which was streaming into memory. If an error occurred, and it was a memory error, the script uses the `unloadCast` command to try to free some memory; it then branches the playback head to a frame in the Director movie named Artists, where the Flash movie sprite first appears, so Director can again try to load and play the Flash movie. If something other than an out- of-memory error occurred, the script goes to a frame named

Sorry, which explains that the requested Flash movie can't be played.

```
on CheckFlashStatus
  errorCheck = member("Dali").getError()
  if errorCheck <> 0 then
    if errorCheck = #memory then
      member("Dali").clearError()
      unloadCast
      go to frame ("Artists")
    else
      go to frame ("Sorry")
    end if
  end if
end
```

See also **clearError**, **getErrorString()**,**state**

getErrorString()

Syntax `member(whichCastMember). getErrorString()`

`getErrorString(member whichCastMember)`

Description Function; for Shockwave Audio (SWA) cast members, returns the error message string that corresponds to the error value returned by the `getError()` function.

Possible `getError()` integer values and corresponding `getErrorString()` messages are:

getError() value	**getErrorString() message**
0	OK
1	memory
2	network
3	playback device
99	other

Example This handler uses `getError()` to determine whether an error occurred for Shockwave Audio cast member Norma Desmond Speaks, and if so, uses `getErrorString` to obtain the error message and assign it to a field cast member:

```
on exitFrame
  if member("Norma Desmond Speaks").getError() <> 0 then
    member("Display Error Name").text = member("Norma Desmond \
Speaks").getErrorString()
  end if
end
```

See also **getError()**

getFlashProperty()

Syntax `sprite(spriteNum).getFlashProperty("targetName", #property)`

Description This function allows Lingo to invoke the Flash action script function `getProperty()` on the given Flash sprite. This Flash action script function is used to get the value of properties of movie clips or levels within a Flash movie. This is similar to testing sprite properties within Director.

The *targetName* is the name of the movie clip or level whose property you want to get within the given Flash sprite.

The *#property* is the name of the property to get. These movie clip properties are gettable: #posX, #posY, #scaleX, #scaleY, #visible, #rotate , #alpha, #name, #width, #height, #target, #url, #dropTarget, #totalFrames, #currentFrame, and #lastframeLoaded.

To get a global property of the Flash sprite, pass an empty string as the *targetName*. These global Flash properies are gettable: `#focusRect` and `#spriteSoundBufferTime`.

See the Flash documentation for descriptions of these properties.

This statement gets the value of the `#rotate` property of the movie clip Star in the Flash member in sprite 3.

```
sprite(3).setFlashProperty("Star", #rotate)
```

See also **setFlashProperty()**

getFrameLabel

Syntax

```
sprite(whichFlashSprite).getFrameLabel(whichFlashFra meNumber)
getFrameLabel(sprite whichFlashSprite, whichFlashFrameNumber)
```

Description

Function; returns the frame label within a Flash movie that is associated with the frame number requested. If the label doesn't exist, or that portion of the Flash movie has not yet been streamed in, this function returns an empty string.

Example

The following handler looks to see if the marker on frame 15 of the Flash movie playing in sprite 1 is called "Lions". If it is, the Director movie navigates to frame "Lions". If it isn't, the Director movie stays in the current frame and the Flash movie continues to play.

```
on exitFrame
  if sprite(1).getFrameLabel(15) = "Lions" then
    go "Lions"
  else
    go the frame
  end if
end
```

getHotSpotRect()

Syntax
```
sprite(whichQTVRSprite).getHotSpotRect(hotSpotID)
getHotSpotRect(whichQTVRSprite, hotSpotID)
```

Description QuickTime VR function; returns an approximate bounding rectangle for the hot spot specified by *hotSpotId*. If the hot spot doesn't exist or isn't visible on the Stage, this function returns rect(0, 0, 0, 0). If the hot spot is partially visible, this function returns the bounding rectangle for the visible portion.

getLast()

Syntax
```
list.getLast()
getLast(list)
```

Description List function; identifies the last value in a linear or property list specified by *list*.

Example This statement identifies the last item, 22, in the list Answers, which consists of [10, 12, 15, 22]:

```
put Answers.getLast()
```

Example This statement identifies the last item, 850, in the list Bids, which consists of [#Gee:750, #Kayne:600, #Ohashi:850]:

```
put Bids.getLast()
```

getLatestNetID

Syntax `getLastestNetID`

Description This function returns an identifier for the last network operation that started.

The identifier returned by `getLatestNetID` can be used as a parameter in the `netDone, netError,` and `netAbort` functions to identify the last network operation.

Note: This function is included for backward compatibility. It is recommended that you use the network ID returned from a net lingo function rather than getLatestNetID. However, if you use getLatestNetID, use it immediately after issuing the netLingo command.

Example This script assigns the network ID of a `getNetText` operation to the field cast member Result so results of that operation can be accessed later.

```
on startOperation
  global gNetID
  getNetText("url")
  set gNetID = getLatestNetID()
end

on checkOperation
  global gNetID
  if netDone(gNetID) then
    put netTextResult into member "Result"
  end if
end
```

netAbort command; **netDone()**and **netError()**functions

getNetText()

Syntax `getNetText(URL {, serverOSString} {, characterSet})`

getNetText(URL, propertyList {, serverOSString} {, characterSet})

Description Function; starts the retrieval of text from a file usually on an HTTP or FTP server, or initiates a CGI query.

The first syntax shown starts the text retrieval. You can submit HTTP CGI queries this way and must properly encode them in the URL. The second syntax includes a property list and submits a CGI query, providing the proper URL encoding.

Use the optional parameter *propertyList* to take a property list for CGI queries. The property list is URL encoded and the URL sent is `(urlstring & "?" & encodedproplist)`.

Use the optional parameter *serverOSString* to encode any return characters in *propertylist*. The value defaults to `UNIX` but may be set to `Win` or `Mac` and translates any carriage returns in the *propertylist* argument into those used on the server. For most applications, this setting is unnecessary because line breaks are usually not used in form responses.

The optional parameter *characterSet* applies only if the user is running Director on a `shift-JIS` (Japanese) system. Possible character set settings are `JIS`, `EUC`, `ASCII`, and `AUTO`. Lingo converts the retrieved data from `shift-JIS` to the named character set. Using the AUTO setting, character set tries to determine what character set the retrieved text is in and translate it to the character set on the local machine. The default setting is ASCII.

For a movie that plays back as an applet, the `getNetText` command retrieves text only from the domain that contains the applet. This behavior differs from Shockwave and is necessary due to Java's security model.

Use `netDone` to find out when the `getNetText` operation is complete, and `netError` to find out if the operation was successful. Use `netTextResult` to return the text retrieved by `getNetText`.

The function works with relative URLs.

To see an example of `getNetText()` used in a completed movie, see the Forms and Post movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This script retrieves text from the URL http://BigServer.com/sample.txt and updates the field cast member the mouse is on when it's clicked:

```
property spriteNum
property theNetID
on mouseUp me
  theNetID = getNetText ("http://BigServer.com/sample.txt")
end
on exitFrame me
  if netDone(theNetID) then
    sprite(spriteNum).member.text = netTextResult(theNetID)
  end if
end
```

Example This example retrieves the results of a CGI query:

```
getNetText("http://www.yourserver.com/cgi-bin/query.cgi?
name=Bill")
```

Example This is the same as the previous example, but it uses a property list to submit a CGI query, and does the URL encoding for you.

```
getNetText("http://www.yourserver.com/cgi-bin/query.cgi",
[#name:"Bill"])
```

See also **netDone()**,**netError()**,**netTextResult()**

getNthFileNameInFolder()

Syntax `getNthFileNameInFolder(folderPath, fileNumber)`

Description Function; returns a file name from the directory folder based on the specified path and number within the folder. To be found by the `getNthFileNameInFolder` function, Director movies must be set to visible in the folder structure. (On the Macintosh, other types of files are found whether they are visible or invisible.) If this function returns an empty string, you have specified a number greater than the number of files in the folder.

The `getNthFileNameInFolder` function doesn't work with URLs.

To specify other folder names, use the `@ pathname` operator or the full path defined in the format for the specific platform on which the movie is running. For example:

- In Windows, use a directory path such as C:\Director\Movies.
- On the Macintosh, use a pathname such as HardDisk:Director:Movies. To look for files on the Macintosh desktop, use the path HardDisk:Desktop Folder
- This function is not available in Shockwave.

Example The following handler returns a list of file names in the folder on the current path. To call the function, use parentheses, as in `put currentFolder()`.

```
on currentFolder
  fileList = [ ]
  repeat with i = 1 to 100
    n = getNthFileNameInFolder(the moviePath, i)
    if n = EMPTY then exit repeat
    fileList.append(n)
  end repeat
  return fileList
end currentFolder
```

See also **@ (pathname)**

getOne()

Syntax `list.getOne(value)`

`getOne(list, value)`

Description List function; identifies the position (linear list) or property (property list) associated with the value specified by *value* in the list specified by *list*.

For values contained in the list more than once, only the first occurrence is displayed. The `getOne` command returns the result 0 when the specified value is not in the list.

When used with linear lists, the `getOne` command performs the same functions as the `getPos` command.

Example This statement identifies the position of the value 12 in the linear list Answers, which consists of [10, 12, 15, 22]:

```
put Answers.getOne(12)
```

The result is 2, because 12 is the second value in the list.

Example This statement identifies the property associated with the value 12 in the property list Answers, which consists of [#a:10, #b:12, #c:15, #d:22]:

```
put Answers.getOne(12)
```

The result is #b, which is the property associated with the value 12.

See also **getPos()**

getPixel()

Syntax

```
imageObject.getPixel(x, y {, #integer})
imageObject.getPixel(point(x, y) {, #integer})
```

Description

This function returns the color value of the pixel at the specified point in the given image object. This value is normally returned as an indexed or RGB color object, depending on the bit depth of the image.

If you include the optional parameter value #integer, however, it's returned as a raw number. If you're setting a lot pixels to the color of another pixel, it's faster to set them as raw numbers. Raw integer color values are also useful because they contain alpha layer information as well as color when the image is 32-bit. The alpha channel information can be extracted from the raw integer by dividing the integer by 2^8+8+8.

`GetPixel()` returns 0 if the given pixel is outside the specified image object.

Example

These statements get the color of the pixel at point (90, 20) in member Happy and set sprite 2 to that color.

```
myColor=member("Happy").image.getPixel(90, 20)
sprite(2).color=myColor
```

Example

This statement sets the variable alpha to the alpha channel value of the point (25, 33) in the 32-bit image object myImage.

```
alpha = myImage.getPixel(25, 33, #integer) / power(2, 8+8+8)
```

See also

depth, **color()**,**setPixel()**,**power()**

getPlaylist()

Syntax

```
sound(channelNum).getPlaylist()
getPlaylist(sound(channelNum))
```

Description

This function returns a copy of the list of queued sounds for *soundObject*. This list does not include the currently playing sound.

The list of queued sounds may not be edited directly. You must use `setPlayList()`.

The playlist is a linear list of property lists. Each property list corresponds to one queued sound cast member. Each queued sound may specify these properties:

Property	Description
#member	The sound cast member to play. This property will always be present; all others are optional.
#startTime	The time within the sound at which playback begins, in milliseconds. See startTime.
#endTime	The time within the sound at which playback ends, in milliseconds. See endTime.
#loopCount	The number of times to play a portion of the sound. See loopCount.
#loopStartTime	The time within the sound at which a loop begins, in milliseconds. See loopStartTime.
#loopEndTime	The time within the sound at which a loop ends, in milliseconds. See loopEndTime.
#preloadTime	The amount of the sound to buffer before playback, in milliseconds. See preloadTime.

Example

This handler queues two sounds in sound channel 2, starts playing them, and then displays the playList in the message window. Notice that the playlist only includes the second sound cueued, because the first sound is already playing.

```
on playMusic
  sound(2).queue([#member:member("chimes")])
  sound(2).queue([#member:member("introMusic"), #startTime:3000,\
  #endTime:10000, #loopCount:5,#loopStartTime:8000,
#loopEndTime:8900])
  sound(2).play()
  put sound(2).getPlaylist()
end

-- [[#member: (member 12 of castLib 2), #startTime: 3000,
#endTime: 10000, #loopCount: 5, #loopStartTime: 8000,
#loopEndTime: 8900]]
```

See also

endTime, **loopCount**, **loopEndTime**, **loopStartTime**, **preLoadTime**, **queue()**,**setPlaylist()**,**startTime**

getPos()

Syntax

```
list.getPos(value)
getPos(list, value)
```

Description

List function; identifies the position of the value specified by *value* in the list specified by *list*. When the specified value is not in the list, the `getPos` command returns the value 0.

For values contained in the list more than once, only the first occurrence is displayed. This command performs the same function as the `getOne` command when used for linear lists.

Example

This statement identifies the position of the value 12 in the list Answers, which consists of [#a:10, #b:12, #c:15, #d:22]:

```
put Answers.getPos(12)
```

The result is 2, because 12 is the second value in the list.

See also

getOne()

getPref()

Syntax `getPref(prefFileName)`

Description Function; retrieves the content of the specified file.

When you use this function, replace *`prefFileName`* with the name of a file created by the `setPref` function. If no such file exists, `getPref` returns `VOID`.

The file name used for *`prefFileName`* must be a valid file name only, not a full path; Director supplies the path. The path to the file is handled by Director. The only valid file extensions for *`prefFileName`* are .txt and .htm; any other extension is rejected.

Do not use this command to access read-only or locked media.

Note: In a browser, data written by `setPref` is not private. Any Shockwave movie can read this information and upload it to a server. Confidential information should not be stored using `setPref`.

To see an example of `getPref` used in a completed movie, see the Read and Write Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This handler retrieves the content of the file Test and then assigns the file's text to the field Total Score:

```
on mouseUp
  theText = getPref("Test")
  member("Total Score").text = theText
end
```

See also **setPref**

getProp()

Syntax

```
getProp(list, property)
list.property
```

Description

Property list function; identifies the value associated with the property specified by *property* in the property list specified by *list*.

Almost identical to the `getaProp` command, the `getProp` command displays an error message if the specified property is not in the list or if you specify a linear list.

Example

This statement identifies the value associated with the property `#c` in the property list Answers, which consists of [#a:10, #b:12, #c:15, #d:22]:

```
getProp(Answers, #c)
```

The result is 15, because 15 is the value associated with `#c`.

See also

getOne()

getPropAt()

Syntax `list.getPropAt(index)`

`getPropAt(list, index)`

Description Property list function; for property lists only, identifies the property name associated with the position specified by *index* in the property list specified by *list*. If the specified item isn't in the list, or if you use `getPropAt()` with a linear list, a script error occurs.

Example This statement displays the second property in the given list:

```
put Answers.getPropAt(2)
-- #b
```

The result is 20, which is the value associated with `#b`.

on getPropertyDescriptionList

Syntax

```
on getPropertyDescriptionList
  statement(s)
end
```

Description

System message and event handler; contains Lingo that generates a list of definitions and labels for the parameters that appear in a behavior's Parameters dialog box.

Place the `on getPropertyDescriptionList` handler within a behavior script. Behaviors that don't contain an `on getPropertyDescriptionList` handler don't appear in the Parameters dialog box and can't be edited from Director's interface.

The `on getPropertyDescriptionList` message is sent when any action that causes the Behavior Inspector to open occurs: either when the user drags a behavior to the Score or the user double-clicks a behavior in the Behavior Inspector.

The `#default`, `#format`, and `#comment` settings are mandatory for each parameter. The following are possible values for these settings:

#default	The parameter's initial setting.
#format	#integer #float #string #symbol #member #bitmap #filmloop #field #palette #picture #sound #button #shape #movie #digitalvideo #script #richtext #ole #transition #xtra #frame #marker #ink #boolean
#comment	A descriptive string that appears to the left of the parameter's editable field in the Parameters dialog box.
#range	A range of possible values that can be assigned to a property. The range is specified as a linear list with several values or as a minimum and maximum in the form of a property list: `[#min: minValue, #max: maxValue].`

Example

This handler defines a behavior's parameters that appear in the Parameters dialog box. Each statement that begins with `addProp` adds a parameter to the list named description. Each element added to the list defines a property and the property's `#default`, `#format`, and `#comment` values:

```
on getPropertyDescriptionList
  description = [:]
  addProp description,#dynamic, [#default:1, #format:#boolean,
#comment:"Dynamic"]
  addProp description,#fieldNum, [#default:1, #format:#integer,
\ #comment:"Scroll which sprite:"]
  addProp description, #extentSprite,
[#default:1,#format:#integer,  \ #comment: "Extend Sprite:"]
  addProp description,#proportional,[#default:1,#format:#boole
an, \ #comment: "Proportional:"]
  return description
end
```

See also

addProp, **on getBehaviorDescription**, **on runPropertyDialog**

getStreamStatus()

Syntax `getStreamStatus(netID)`

`getStreamStatus(URLString)`

Description Function; returns a property list matching the format used for the globally available `tellStreamStatus` function that can be used with callbacks to sprites or objects. The list contains the following strings:

#URL	String containing the URL location used to start the network operation.
#state	String consisting of Connecting, Started, InProgress, Complete, “Error”, or “NoInformation” (this last string is for the condition when either the net ID is so old that the status information has been dropped or the URL specified in `URLString` was not found in the cache).
#bytesSoFar	Number of bytes retrieved from the network so far.
#bytesTotal	Total number of bytes in the stream, if known. The value may be 0 if the HTTP server does not include the content length in the MIME header.
#error	String containing ““ (`EMPTY`) if the download is not complete, OK if it completed successfully, or an error code if the download ended with an error.

Description For example, you can start a network operation with `getNetText()` and track its progress with `getStreamStatus()`.

Example This statement displays in the message window the current status of a download begun with `getNetText()` and the resulting net ID placed in the variable `netID`:

```
put getStreamStatus(netID)

-- [#URL: "www.macromedia.com", #state: "InProgress", #bytesSoFar:
250,  \ #bytesTotal: 50000, #error: EMPTY]
```

See also **on streamStatus**, **tellStreamStatus()**

getVariable()

Syntax `getVariable(sprite flashSpriteNum, "variableName")`

Description This function returns the current value of the given variable from the given Flash sprite. Flash variables were introduced in Flash version 4.

This statement returns the value of the variable currentURL from the Flash cast member in sprite 3 and displays it in the Message window:

```
put getVariable(sprite 3, "currentURL")
-- "http://www.macromedia.com/software/flash/"
```

See also **hitTest()**, **setVariable()**

global

Syntax `global variable1 {, variable2} {, variable3}...`

Description Keyword; defines a variable as a global variable so that other handlers or movies can share it.

Every handler that examines or changes the content of a global variable must use the `global` keyword to identify the variable as global. Otherwise, the handler treats the variable as a local variable, even if it is declared to be global in another handler.

Note: To ensure that global variables are available throughout a movie, declare and initialize them in the `prepareMovie` handler. Then, if you leave and return to the movie from another movie, your global variables will be reset to the initial values unless you first check to see that they aren't already set.

A global variable can be declared in any handler or script. Its value can be used by any other handlers or scripts that also declare the variable as global. If the script changes the variable's value, the new value is available to every other handler that treats the variable as global.

A global variable is available in any script or movie, regardless of where it is first declared; it is not automatically cleared when you navigate to another frame, movie, or window.

Any variables manipulated in the Message window are automatically global, even though they are not explicitly declared as such.

Shockwave movies playing on the Internet cannot access global variables within other movies, even movies playing on the same HTML page. The only way movies can share global variables is if an embedded movie navigates to another movie and replaces itself through either `goToNetMovie` or `go movie`.

Example This example sets the global variable StartingPoint to an initial value of 1 if it doesn't already contain a value. This allows navigation to and from the movie without loss of stored data.

```
global gStartingPoint
on prepareMovie
  if voidP(gStartingPoint) then gStartingPoint = 1
end
```

See also **showGlobals**, **property**, **gotoNetMovie**

globals

Syntax `the globals`

Description System property; this property contains a special property list of all current global variables with a value other than `VOID`. Each global variable is a property in the list, with the associated paired value.

You can use the following list operations on `globals`:

- `count()`—Returns the number of entries in the list.
- `getPropAt(n)`—Returns the name of the *n*th entry.
- `getProp(x)`—Returns the value of an entry with the specified name.
- `getAProp(x)`—Returns the value of an entry with the specified name.

Note: The `globals` property automatically contains the property #version, which is the version of Director running. This means there will always be at least one entry in the list, even if no global variables have been declared yet.

This property differs from `showGlobals` in that `the globals` can be used in contexts other than the Message window. To display `the globals` in the Message window, use `showGlobals`.

See also **showGlobals**, **clearGlobals**

go

Syntax

```
go {to} {frame} whichFrame
go {to} movie whichMovie
go {to} {frame} whichFrame of movie whichMovie
```

Description

Command; causes the playback head to branch to the frame specified by *whichFrame* in the movie specified by *whichMovie*. The expression *whichFrame* can be a marker label or an integer frame number. The expression *whichMovie* must specify a movie file. (If the movie is in another folder, *whichMovie* must specify the path.)

The phrase `go loop` tells the playback head to loop to the previous marker and is a convenient means of keeping the playback head in the same section of the movie while Lingo remains active and avoids the use of `go to the frame` in a frame that has a transition which would slow the movie and overwhelm the processor.

It is best to refer to marker labels instead of frame numbers; editing a movie can cause frame numbers to change. Using marker labels also makes it easier to read scripts.

The `go to movie` command loads frame 1 of the movie. If the command is called from within a handler, the handler in which it is placed continues executing. To suspend the handler while playing the movie, use the `play` command, which may be followed by a subsequent `play done` to return.

When you specify a movie to play, specify its path if the movie is in a different folder, but to prevent a potential load failure, don't include the movie's .dir, .dxr or .dcr file extension.

To more efficiently go to a movie at a URL, use the `downloadNetThing` command to download the movie file to a local disk first and then use the `go to movie` command to go to that movie on the local disk.

The following are reset when a movie is loaded: `beepOn` and `constraint` properties; `keyDownScript`, `mouseDownScript`, and `mouseUpScript`; `cursor` and `immediate` sprite properties; `cursor` and `puppetSprite` commands; and custom menus. However, `the timeoutScript` is not reset when loading a movie.

Example

This statement sends the playback head to the marker named start:

```
go to "start"
```

Example

This statement sends the playback head to the marker named Memory in the movie named Noh Tale to Tell:

```
go frame("Memory") of movie("Noh Tale to Tell")
```

Example

This handler tells the movie to loop in the current frame. This handler is useful for making the movie wait in a frame while it plays so the movie can respond to events.

```
on exitFrame
  go the frame
end
```

See also

downloadNetThing, **gotoNetMovie**, **label()**,**marker()**,**pathName (movie property)**, **play**, **play done**

go loop

Syntax `go loop`

Description Command; sends the playback head to the previous marker in the movie, either one marker back from the current frame if the current frame does not have a marker, or to the current frame if the current frame has a marker.

Note: This command is equivalent to marker(0) in previous versions of Director.

If no markers are to the left of the playback head, the playback head branches to:

- The next marker to the right if the current frame does not have a marker.
- The current frame if the current frame has a marker.
- Frame 1 if the movie contains no markers.

The `go loop` command is equivalent to the statement `go to the marker(0)` used in earlier versions of Lingo.

Example This statement causes the movie to loop between the current frame and the previous marker:

```
go loop
```

See also **go**, **go next**, **go previous**

go next

Syntax `go next`

Description Command; sends the playback head to the next marker in the movie. If no markers are to the right of the playback head, the playback head goes to the last marker in the movie or to frame 1 if there are no markers in the movie.

The `go next` command is equivalent to the statement `go marker(1)` that was used in earlier versions of Lingo.

Example This statement sends the playback head to the next marker in the movie:

```
go next
```

See also **go**, **go loop**, **go previous**

go previous

Syntax `go previous`

Description Command; sends the playback head to the previous marker in the movie. This marker is two markers back from the current frame if the current frame does not have a marker or one marker back from the current frame if the current frame has a marker.

Note: This command is equivalent to marker(-1) in previous versions of Director.

If no markers are to the left of the playback head, the playback head branches to:

- The next marker to the right if the current frame does not have a marker
- The current frame if the current frame has a marker
- Frame 1 if the movie contains no markers

Example This statement sends the playback head to the previous marker in the movie:

```
go previous
```

See also **go**, **go loop**, **go next**

goToFrame

Syntax

```
sprite(whichFlashSprite).goToFrame(frameNumber)
goToFrame(sprite whichFlashSprite, frameNumber)
sprite(whichFlashSprite).goToFrame(labelNameString)
goToFrame(sprite whichFlashSprite, labelNameString)
```

Description

Command; plays a Flash movie sprite beginning at the frame identified by the *frameNumber* parameter. You can identify the frame by either an integer indicating a frame number or by a string indicating a label name. Using the `goToFrame` command has the same effect as setting a Flash movie sprite's `frame` property.

Example

This handler branches to different points within a Flash movie in channel 5. It accepts a parameter that indicates which frame to go to.

```
on Navigate whereTo
  sprite(5).goToFrame(whereTo)
end
```

gotoNetMovie

Syntax

```
gotoNetMovie URL
gotoNetMovie (URL)
```

Description

Command; retrieves and plays a new Shockwave movie from an HTTP or FTP server. The current movie continues to run until the new movie is available.

Only URLs are supported as valid parameters. The URL can specify either a file name or a marker within a movie. Relative URLs work if the movie is on an Internet server, but you must include the extension with the file name.

When performing testing on a local disk or network, media must be located in a directory named dswmedia.

If a `gotoNetMovie` operation is in progress and you issue a second `gotoNetMovie` command before the first is finished, the second command cancels the first.

Example

In this statement, the URL indicates a Director file name:

```
gotoNetMovie "http://www.yourserver.com/movies/movie1.dcr"
```

Example

In this statement, the URL indicates a marker within a file name:

```
gotoNetMovie
"http://www.yourserver.com/movies/buttons.dcr#Contents"
```

Example

In this statement, `gotoNetMovie` is used as a function. The function returns the network ID for the operation.

```
myNetID = gotoNetMovie
("http://www.yourserver.com/movies/buttons.dcr#Contents")
```

gotoNetPage

Syntax `gotoNetPage "URL", {"targetName"}`

Description Command; opens a Shockwave movie or another MIME file in the browser.

Only URLs are supported as valid parameters. Relative URLs work if the movie is on an HTTP or FTP server.

The *targetName* argument is an optional HTML parameter that identifies the frame or window in which the page is loaded.

- If *targetName* is a window or frame in the browser, `gotoNetPage` replaces the contents of that window or frame.
- If *targetName* isn't a frame or window that is currently open, `goToNetPage` opens a new window.
- If *targetName* is not included, `gotoNetPage` replaces the current page, wherever it is located.

In the authoring environment, the `gotoNetPage` command launches the preferred browser if it is enabled. In projectors, this command tries to launch the preferred browser set with the Network Preferences dialog box or `browserName` command. If neither has been used to set the preferred browser, the `goToNetPage` command attempts to find a browser on the computer.

Example This script loads the file Newpage.html into the frame or window named frwin. If a window or frame in the current window called frwin exists, that window or frame is used. If the window frwin doesn't exist, a new window named frwin is created.

```
on keyDown
  gotoNetPage "Newpage.html", "frwin"
end
```

Example This handler opens a new window regardless of what window the browser currently has open:

```
on mouseUp
  goToNetPage "Todays_News.html", "_new"
end
```

See also **browserName()**,**netDone()**

gradientType

Syntax `member(whichCastMember).gradientType`

Description Vector shape cast member property; specifies the actual gradient used in the cast member's fill.

Possible values are `#linear` or `#radial`. The `gradientType` is only valid when the `fillMode` is set to `#gradient`.

This property can be tested and set.

Example This handler toggles between linear and radial gradients in cast member "backdrop".

```
on mouseUp me
  if member("backdrop").gradientType = #radial then
    member("backdrop").gradientType = #linear
  else
    member("backdrop").gradientType = #radial
  end if
end
```

See also **fillMode**

halt

Syntax `halt`

Description Command; exits the current handler and any handler that called it and stops the movie during authoring or quits the projector during run time from a projector.

Example This statement checks whether the amount of free memory is less than 50K and, if it is, exits all handlers that called it and then stops the movie:

```
if the freeBytes < 50*1024 then halt
```

See also **abort**, **exit**, **pass**, **quit**

handler()

Syntax `scriptObject.handler(#handlerSymbol)`

Description This function returns `TRUE` if the given *`scriptObject`* contains a handler whose name is #*`handlerSymbol`*, and `FALSE` if it does not. The script object must be a parent script, a child object, or a behavior.

Example This Lingo code invokes a handler on an object only if that handler exists:

```
if spiderObject.handler(#pounce) = TRUE then
  spiderObject.pounce()
end if
```

See also **handlers()**,**new()**,**rawNew()**,**script**

handlers()

Syntax `scriptObject.handlers()`

Description This function returns a linear list of the handlers in the given *`scriptObject`*. Each handler name is presented as a symbol in the list. This function is useful for debugging movies.

> Note that you cannot get the handlers of a script cast member directly. You have to get them via the `script` property of the member.

Example This statement displays the list of handlers in the child object RedCar in the Message window:

```
put RedCar.handlers()
-- [#accellerate, #turn, #stop]
```

Example This statement displays the list of handlers in the parent script member CarParentScript in the Message window:

```
put member("CarParentScript").script.handlers()
-- [#accellerate, #turn, #stop]
```

See also **handler()**, **script**

height

Syntax
```
member(whichCastMember).height
the height of member whichCastMember
imageObject.height
sprite(whichSprite).height
the height of sprite whichSprite
```

Description Cast member, image object and sprite property; for vector shape, Flash, animated GIF, bitmap, and shape cast members, determines the height, in pixels, of the cast member displayed on the Stage.

- For cast members, works with bitmap and shape cast members only. This property can be tested but not set.
- For image objects, this property can be tested but not set.
- For sprites: setting the sprite's height automatically sets the sprite's `stretch` property to `TRUE`. For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet. This property can be tested and set.

Example This statement assigns the height of cast member Headline to the variable `vHeight`:

```
vHeight = member("Headline").height
```

Example This statement sets the height of sprite 10 to 26 pixels:

```
sprite(10).height = 26
```

Example This statement assigns the height of sprite (i + 1) to the variable `vHeight`:

```
vHeight = sprite(i + 1).height
```

See also **height**, **rect (sprite)**, **width**

hilite (command)

Syntax
```
fieldChunkExpression.hilite()
```
hilite fieldChunkExpression

Description Command; highlights (selects) in the field sprite the specified chunk, which can be any chunk that Lingo lets you define, such as a character, word, or line. On the Macintosh, the highlight color is set in the Color control panel.

Example This statement highlights the fourth word in the field cast member Comments, which contains the string Thought for the Day:

```
member("Comments").word[4].hilite()
```

Example This statement causes highlighted text within the sprite for field myRecipes to be displayed without highlighting:

```
myLineCount = member("myRecipes").line.count
member("myRecipes").line[myLineCount + 1].hilite()
```

See also **char...of**, **item...of**, **line...of**, **word...of**; **delete**, **mouseChar**, **mouseLine**, **mouseWord**; **field**; **selection() (function)**; **selEnd**, **selStart**

hilite (cast member property)

Syntax

```
member(whichCastMember).hilite
the hilite of member whichCastMember
```

Description

Cast member property; determines whether a check box or radio button created with the button tool is selected (`TRUE`) or not (`FALSE`, default).

If *`whichCastMember`* is a string, it specifies the cast member name. If it is an integer, *`whichCastMember`* specifies the cast member number.

This property can be tested and set.

Example

This statement checks whether the button named Sound on is selected and, if it is, turns sound channel 1 all the way up:

```
if member("Sound on").hilite = TRUE then sound(1).volume = 255
```

Example

This statement uses Lingo to select the button cast member powerSwitch by setting the `hilite` member property for the cast member to `TRUE`:

```
member("powerSwitch").hilite = TRUE
```

See also

checkBoxAccess, **checkBoxType**

hitTest()

Syntax
```
sprite(whichFlashSprite).hitTest(point)
hitTest(sprite whichFlashSprite, point)
```

Description Function; indicates which part of a Flash movie is directly over a specific Director Stage location. The Director Stage location is expressed as a Director point value: for example, point(100,50). The `hitTest` function returns these values:

- `#background`—The specified Stage location falls within the background of the Flash movie sprite.
- `#normal`— The specified Stage location falls within a filled object.
- `#button`— The specified Stage location falls within the active area of a button.
- `#editText`— The specified Stage location falls within a Flash editable text field.

Example This frame script checks to see if the mouse is currently located over a button in a Flash movie sprite in channel 5 and, if it is, the script sets a text field used to display a status message:

```
on exitFrame
  if sprite(5).hitTest(the mouseLoc) = #button then
    member("Message Line").text = "Click here to play the movie."
    updateStage
  else
    member("Message Line").text = ""
  end if
  go the frame
end
```

HMStoFrames()

Syntax `HMStoFrames(hms, tempo, dropFrame, fractionalSeconds)`

Description Function; converts movies measured in hours, minutes, and seconds to the equivalent number of frames or converts a number of hours, minutes, and seconds into time if you set the *tempo* argument to 1 (1 frame = 1 second).

- *hms*—String expression that specifies the time in the form `sHH:MM:SS.FFD`, where:

  s A character is used if the time is less than zero, or a space if the time is greater than or equal to zero.

  HH Hours.

  MM Minutes.

  SS Seconds.

  FF Indicates a fraction of a second if *fractionalSeconds* is TRUE or frames if *fractionalSeconds* is FALSE.

  D A d is used if *dropFrame* is TRUE, or a space if *dropFrame* is FALSE.

-
- *tempo*—Specifies the tempo in frames per second.
- *dropFrame*—Logical expression that determines whether the frame is a drop frame (`TRUE`) or not (`FALSE`). If the string *hms* ends in a *d*, the time is treated as a drop frame, regardless of the value of *dropFrame*.
- *fractionalSeconds*—Logical expression that determines the meaning of the numbers after the seconds; they can be either fractional seconds rounded to the nearest hundredth of a second (`TRUE`) or the number of residual frames (`FALSE`).

Example This statement determines the number of frames in a 1-minute, 30.1-second movie when the tempo is 30 frames per second. Neither the *dropFrame* nor *fractionalSeconds* arguments is used.

```
put HMStoFrames(" 00:01:30.10 ", 30, FALSE, FALSE)
-- 2710
```

Example This statement converts 600 seconds into minutes:

```
>> put framesToHMS(600, 1,0,0)
>> -- " 00:10:00.00 "
```

Example This statement converts an hour and a half into seconds:

```
>> put HMStoFrames("1:30:00", 1,0,0)
>> -- 5400
```

See also **framesToHMS()**

hold

Syntax

```
sprite(whichFlashSprite).hitTest(point)
```

```
hold sprite whichFlashSprite
```

Description

Flash command; stops a Flash movie sprite that is playing in the current frame, but any audio continues to play.

Example

This frame script holds the Flash movie sprites playing in channels 5 through 10 while allowing the audio for these channels to continue playing:

```
on enterFrame
  repeat with i = 5 to 10
    sprite(i).hold()
  end repeat
end
```

See also

movieRate, **pause (movie playback)**

hotSpot

Syntax

```
member(whichCursorCastMember).hotspot
the hotspot of member whichCursorCastMember
```

Description

Cursor cast member property; specifies the horizontal and vertical point location of the pixel that represents the hotspot within the animated color cursor cast member `whichCursorCastMember`. Director uses this point to track the cursor's position on the screen (for example, when it returns the values for the Lingo functions `mouseH` and `mouseV`) and to determine where a rollover (signaled by the Lingo message `mouseEnter`) occurs.

The upper left corner of a cursor is point(0,0), which is the default `hotSpot` value. Trying to set a point outside the bounds of the cursor produces an error. For example, setting the hotspot of a 16-by-16-pixel cursor to point(16,16) produces an error (because the starting point is 0,0, not 1,1).

This property can be tested and set.

Example

This handler sets the hotspot of a 32-by-32-pixel cursor (whose cast member number is stored in the variable `cursorNum`) to the middle of the cursor:

```
on startMovie
  member(cursorNum).hotSpot = point(16,16)
end
```

hotSpotEnterCallback

Syntax

```
sprite(whichQTVRSprite).hotSpotEnterCallback
the hotSpotEnterCallback of sprite whichQTVRSprite
```

Description

QuickTime VR sprite property; contains the name of the handler that runs when the cursor enters a QuickTime VR hot spot that is visible on the Stage. The QuickTime VR sprite receives the message first. The message has two arguments: the `me` parameter and the ID of the hot spot that the cursor entered.

To clear the callback, set this property to 0.

To avoid a performance penalty, set a callback property only when necessary.

This property can be tested and set.

See also

hotSpotExitCallback, **nodeEnterCallback**, **nodeExitCallback**, **triggerCallback**

hotSpotExitCallback

Syntax
```
sprite(whichQTVRSprite).hotSpotExitCallback
the hotSpotExitCallback of sprite whichQTVRSprite
```

Description
QuickTime VR sprite property; contains the name of the handler that runs when the cursor leaves a QuickTime VR hot spot that is visible on the Stage. The QuickTime VR sprite receives the message first. The message has two arguments: the `me` parameter and the ID of the hot spot that the cursor entered.

To clear the callback, set this property to 0.

To avoid a performance penalty, set a callback property only when necessary.

This property can be tested and set.

See also
nodeEnterCallback, **nodeExitCallback**, **triggerCallback**

HTML

Syntax `member(whichMember).HTML`

Description Cast member property; accesses text and tags that control the layout of the text within an HTML-formatted text cast member.

This property can be tested and set.

Example This statement displays in the message window the HTML formatting information embedded in the text cast member Home Page:

```
put member("Home Page").HTML
```

See also **importFileInto**, **RTF**

hyperlink

Syntax `chunkExpression.hyperlink`

Description Text cast member property; returns the hyperlink string for the specified chunk expression in the text cast member.

This property can be both tested and set.

When retrieving this property, the link containing the first character of *`chunkExpression`* is used.

Hyperlinks may not overlap. Setting a hyperlink over an existing link, even partially over it), replaces the initial link with the new one.

Setting a hyperlink to an empty string removes it.

Example This handler creates a hyperlink in the first word of text cast member “MacroLink”. The text is linked to Macromedia’s web site.

```
on startMovie
  member("MacroLink").word[1].hyperlink = \
  "http://www.macromedia.com"
end
```

See also **hyperlinkRange**, **hyperlinkState**

on hyperlinkClicked

Syntax

```
on hyperlinkClicked me, data, range
  statement(s)
end
```

Description

System message and event handler; used to determine when a hyperlink is actually clicked.

This event handler has the following parameters:

- `me`—Used in a behavior to identify the sprite instance
- `data`—The hyperlink data itself; the string entered in the Text Inspector when editing the text cast member
- `range`—The character range of the hyperlink in the text (It's possible to get the text of the range itself by using the syntax member Ref.char[range[1]..range[2]]

This handler should be attached to a sprite as a behavior script. Avoid placing this handler in a cast member script.

Example

This behavior shows a link examining the hyperlink that was clicked, jump to a URL if needed, then output the text of the link itself to the message window:

```
property spriteNum

on hyperlinkClicked me, data, range
  if data starts "http://" then
    goToNetPage(data)
  end if
  currentMember = sprite(spriteNum).member
  anchorString = currentMember.char[range[1]..range[2]]
  put "The hyperlink on"&&anchorString&&"was just clicked."
end
```

hyperlinkRange

Syntax `*chunkExpression*.hyperlinkRange`

Description Text cast member property; returns the range of the hyperlink that contains the first character of the chunk expression.

This property can be tested but not set.

Like `hyperLink` and `hyperLinkState`, the returned range of the link contains the first character of `*chunkExpression*`.

See also **hyperlink**, **hyperlinkState**

hyperlinks

Syntax `chunkExpression.hyperlinks`

Description Text cast member property; returns a linear list containing all the hyperlink ranges for the specified chunk of a text cast member. Each range is given as a linear list with two elements, one for the starting character of the link and one for the ending character.

Example This statement returns all the links for the text cast member Glossary to the message window:

```
put member("Glossary").hyperlinks
-- [[3, 8], [10, 16], [41, 54]]
```

hyperlinkState

Syntax `textChunk.hyperlinkState`

Description Text cast member property; contains the current state of the hyperlink. Possible values for the state are: `#normal`, `#active`, and `#visited`.

This property can be tested and set.

Like `hyperLink` and `hyperLinkRange`, the returned range of the link contains the first character of *`chunkExpression`*.

Example This handler checks to see if the hyperlink clicked is a web address. If it is, the state of the hyperlink text state is set to #visited, and the movie branches to the web address.

```
property spriteNum
on hyperlinkClicked me, data, range
  if data starts "http://" then
    currentMember = sprite(spriteNum).member
    currentMember.word[4].hyperlinkState = #visited
    goToNetPage(data)
  end if
end
```

See also **hyperlink**, **hyperlinkRange**

on idle

Syntax

```
on idle
  statement(s)
end
```

Description

System message and event handler; contains statements that run whenever the movie has no other events to handle and is a useful location for Lingo statements that you want to execute as frequently as possible, such as statements that update values in global variables and displays current movie conditions.

Because statements in `on idle` handlers run frequently, it is good practice to avoid placing Lingo that takes a long time to process in an `on idle` handler.

It is often preferable to put `on idle` handlers in frame scripts instead of movie scripts to take advantage of the `on idle` handler only when appropriate.

Director can load cast members from an internal or external cast during an `idle` event. However, it cannot load linked cast members during an `idle` event.

The `idle` message is only sent to frame scripts and movie scripts.

Example

This handler updates the time being displayed in the movie whenever there are no other events to handle:

```
on idle
  member("Time").text = the short time
end idle
```

See also

idleHandlerPeriod

idleHandlerPeriod

Syntax `the idleHandlerPeriod`

Description Movie property; determines the maximum number of ticks that passes until the movie sends an `idle` message. The default value is 1, which tells the movie to send `idle` handler messages no more than 60 times per second.

When the playback head enters a frame, Director starts a timer, repaints the appropriate sprites on the Stage, and issues an `enterFrame` event. Then, if the amount of time set for the tempo has elapsed, Director generates an `exitFrame` event and goes to the next specified frame; if the amount of time set for this frame hasn't elapsed, Director waits until the time runs out and periodically generates an `idle` message. The amount of time between `idle` events is determined by `idleHandlerPeriod`.

Possible settings for `idleHandlerPeriod` are:

- 0—As many idle events as possible
- 1—Up to 60 per second
- 2—Up to 30 per second
- 3—Up to 20 per second
- *n*—Up to 60/*n* per second

The number of `idle` events per frame also depends on the frame rate of the movie and other activity, including whether Lingo scripts are executing. If the tempo is 60 frames per second (fps) and the `idleHandlerPeriod` value is 1, one `idle` event per frame occurs. If the tempo is 20 fps, three `idle` events per frame occur. Idle time results from Director doesn't have a current task to perform and cannot generate any events.

In contrast, if the `idleHandlerPeriod` property is set to 0 and the tempo is very low, thousands of `idle` events can be generated.

The default value for this property is 1, which differs from previous versions in which it defaulted to 0.

Example The following statement causes the movie to send an `idle` message a maximum of once per second:

```
the idleHandlerPeriod = 60
```

See also **idleHandlerPeriod**, **idleLoadDone()**,**idleLoadMode**, **idleLoadTag**, **idleReadChunkSize**

idleLoadDone()

Syntax `idleLoadDone(loadTag)`

Description Function; reports whether all cast members with the given tag have been loaded (`TRUE`) or are still waiting to be loaded (`FALSE`).

Example This statement checks whether all cast members whose load tag is 20 have been loaded and then plays the movie Kiosk if they are:

```
if idleLoadDone(20) then play movie("on idle")"
```

See also **idleHandlerPeriod**, **idleLoadMode**, **idleLoadPeriod**, **idleLoadTag**, **idleReadChunkSize**

idleLoadMode

Syntax `the idleLoadMode`

Description System property; determines when the `preLoad` and `preLoadMember` commands try to load cast members during idle periods according to the following values:

- 0—Does not perform idle loading
- 1—Performs idle loading when there is free time between frames
- 2—Performs idle loading during `idle` events
- 3—Performs idle loading as frequently as possible

The `idleLoadMode` system property performs no function and works only in conjunction with the `preLoad` and `preLoadMember` commands.

Cast members that were loaded using idle loading remain compressed until the movie uses them. When the movie plays back, it may have noticeable pauses while it decompresses the cast members.

Example This statement causes the movie to try as frequently as possible to load cast members designated for preloading by the `preLoad` and `preLoadMember` commands:

```
the idleLoadMode = 3
```

See also **idleHandlerPeriod**, **idleLoadDone()**,**idleLoadPeriod**, **idleLoadTag**, **idleReadChunkSize**

idleLoadPeriod

Syntax `the idleLoadPeriod`

Description System property; determines the number of ticks that Director waits before trying to load cast members waiting to be loaded. The default value for `idleLoadPeriod` is 0, which instructs Director to service the load queue as frequently as possible.

Example This statement instructs Director to try loading every 1/2 second (30 ticks) any cast members waiting to be loaded:

```
set the idleLoadPeriod = 30
```

See also **idleLoadDone()**,**idleLoadMode**, **idleLoadTag**, **idleReadChunkSize**

idleLoadTag

Syntax `the idleLoadTag`

Description System property; identifies or tags with a number the cast members that have been queued for loading when the computer is idle. The `idleLoadTag` is a convenience that identifies the cast members in a group that you want to preload.

The property can be tested and set using any number that you choose.

Example This statement makes the number 10 the idle load tag:

```
the idleLoadTag = 10
```

See also **idleHandlerPeriod**, **idleLoadDone()**,**idleLoadMode**, **idleLoadPeriod**, **idleReadChunkSize**

idleReadChunkSize

Syntax `the idleReadChunkSize`

Description System property; determines the maximum number of bytes that Director can load when it attempts to load cast members from the load queue. The default value is 32K.

This property can be tested and set.

Example This statement specifies that 500K is the maximum number of bytes that Director can load in one attempt at loading cast members in the load queue:

```
the idleReadChunkSize = 500 * 1024
```

See also **idleHandlerPeriod**, **idleLoadDone()**,**idleLoadMode**, **idleLoadPeriod**, **idleLoadTag**

if

Syntax

```
if logicalExpression then statement
```

```
if logicalExpression then statement
else statement
end if
```

```
if logicalExpression then
     statement(s)
end if
```

```
if logicalExpression then
     statement(s)
else
     statement(s)
end if
```

```
if logicalExpression1 then
     statement(s)
else if logicalExpression2 then
     statement(s)
else if logicalExpression3 then
     statement(s)
end if
```

```
if logicalExpression1 then
     statement(s)
else logicalExpression2
end if
```

Description

Keyword; `if...then` structure that evaluates the logical expression specified by *`logicalExpression`*.

- If the condition is `TRUE`, Lingo executes the statement(s) that follow `then`.
- If the condition is `FALSE`, Lingo executes the statement(s) following `else`. If no statements follow `else`, Lingo exits the `if...then` structure.
- All parts of the condition must be evaluated; execution does not stop at the first condition that is met or not met. Thus, faster code may be created by nesting `if...then` statements on separate lines instead of placing them all on the first line to be evaluated.

When the condition is a property, Lingo automatically checks whether the property is `TRUE`. You don't need to explicitly add the phrase `= TRUE` after the property.

The `else` portion of the statement is optional. To use more than one *`then- statement`* or *`else-statement`*, you must end with the form `end if`.

The `else` portion always corresponds to the previous `if` statement; thus, sometimes you must include an `else nothing` statement to associate an `else` keyword with the proper `if` keyword.

Note: A quick way to determine in the script window if a script is paired properly is to press Tab. This forces Director to check the open Script window and show the indentation for the contents. Any mismatches will be immediately apparent.

Example

This statement checks whether the carriage return was pressed and then continues if it was:

```
if the key = RETURN then go the frame + 1
```

Example This handler checks whether the Command and Q keys were pressed simultaneously and, if so, executes the subsequent statements:

```
on keyDown
  if (the commandDown) and (the key = "q") then
    cleanUp
    quit
  end if
end keyDown
```

Example Compare the following two constructions and the performance results. The first construction evaluates both conditions, and so must determine the time measurement, which may take a while. The second construction evaluates the first condition; the second condition is checked only if the first condition is `TRUE`.

```
spriteUnderCursor = rollOver()
if (spriteUnderCursor > 25) AND MeasureTimeSinceIStarted() then
  alert "You found the hidden treasure!"
end if
```

The alternate, and faster construction would be:

```
spriteUnderCursor = rollOver()
if (spriteUnderCursor > 25) then
  if MeasureTimeSinceIStarted() then
    alert "You found the hidden treasure!"
  end if
end if
```

See also **case**

ilk()

Syntax `ilk(object)`

`ilk(object, type)`

Description Function; indicates the type of an object.

- The syntax `ilk(object)` returns a value indicating the type of an object. If the object is a list, `ilk(object)` returns `#list`; if the object is a property list, `ilk(object)` returns `#propList`.
- The syntax `ilk(object, type)` compares the object represented by `object` to the specified type. If the object is of the specified type, the `ilk()` function returns `TRUE`. If the object is not of the specified type, the `ilk()` function returns `FALSE`.

The following table shows the return value for each type of object recognized by `ilk()`:

Type of Object	ilk(Object) returns	ilk(Object, Type) returns 1 only if Type =	Example
linear list	#list	#list or #linearlist	ilk ([1,2,3])
property list	#proplist	#list or #proplist	ilk ([#his: 1234, #hers: 7890])
integer	#integer	#integer or #number	ilk (333)
float	#float	#float or #number	ilk (123.456)
string	#string	#string	ilk ("asdf")
rect	#rect	#rect or #list	ilk (sprite(1).rect)
point	#point	#point or #list	ilk (sprite(1).loc)
color	#color	#color	ilk (sprite(1).color)
date	#date	#date	ilk (the systemdate)
symbol	#symbol	#symbol	ilk (#hello)
void	#void	#void	ilk (void)
picture	#picture	#picture	ilk (member (2).picture)
parent script instance	#instance	#object	ilk (new (script "blahblah"))
xtra instance	#instance	#object	ilk (new (xtra "fileio"))
member	#member	#member	ilk (member 1)
xtra	#xtra	#object	ilk (xtra "fileio")
script	#script	#object	ilk (script "blahblah")
castlib	#castlib	#object	ilk (castlib 1)
sprite	#sprite	#object	ilk (sprite 1)
sound	#sound	#object	ilk (sound "yaddayadda")

window	#window	#object	ilk (the stage)
media	#media	#object	ilk (member (2).media)

Example The following `ilk` statement identifies the type of the object named Bids.

```
Bids = [:]
put ilk( Bids )
-- #proplist
```

Example The following `ilk` statement tests whether the variable Total is a list and displays the result in the Message window:

```
Total = 2+2
put ilk( Total, #list )
-- 0
```

In this case, since the variable Total is not a list, the Message window displays 0, which is the numeric equivalent of FALSE.

Example The following example tests a variable named myVariable and verifies that it is a date object before displaying it in the Message window:

```
myVariable = the systemDate
if ilk(myVariable, #date) then put myVariable
-- date( 1999, 2, 19 )
```

image

Syntax

```
whichMember.image
(the stage).image
window(windowName).image
```

Description

This property refers to the image object of a bitmap or text cast member, of the Stage, or of a window. You can get or set a cast member's image, but you can only get the image of the Stage or a window.

Setting a cast member's image property immediately changes the contents of the member. However, when you get the image of a member or window, Director creates a reference to the image of the specified member or window. If you make changes to the image, the contents of the cast member or window change immediately.

If you plan to make a lot of changes to an item's image property, it is faster to copy the item's image property into a new image object using the `duplicate()` function, apply your changes to the new image object, and then set the original item's image to the new image object. For non-bitmap members, it is always faster to use the `duplicate()` function.

Example

This statement puts the image of cast member originalFlower into cast member newFlower.

```
member("newFlower").image = member("originalFlower").image
```

Example

These statements place a reference to the image of the stage into the variable `myImage` and then put that image into cast member flower.

```
myImage=(the stage).image
member("flower").image = myImage
```

See also

setPixel(), **draw()**, **image()**, **fill()**, **duplicate() (image function)**, **copyPixels()**

image()

Syntax `image(width, height, bitDepth {, alphaDepth} {, paletteSymbolOrMember})`

Description Function; creates and returns a new image object of the dimensions specified by *`width`*, *`height`*, and *`bitDepth`*, with optional *`alphaDepth`* and *`paletteObject`* values.

The `bitDepth` can be 1, 2, 4, 8, 16, or 32. The `alphaDepth`, if given, is used only for 32-bit images and must be 0 or 8. The `paletteObject`, if given, is used only for 2-, 4-, and 8-bit images and can be either a palette symbol, such as `#grayscale`, or a palette cast member. If no palette is specified, the movie's default palette is used.

Image objects created with `image()` are independent and do not refer to any cast member or window.

To see an example of `image()` used in a completed movie, see the Imaging movie in the Learning\Lingo Examples folder inside the Director application folder.

Example These statements create a 200 x 200 pixel, 8-bit image object and fills the image object with red.

```
redSquare = image(200, 200, 8)
redSquare.fill(0, 0, 200, 200, rgb(255, 0, 0))
```

See also **palette**, **image**, **duplicate() (image function)**, **fill()**

imageCompression

Syntax

```
member(whichMember).imageCompression
the imageCompression of member whichMember
```

Description

This bitmap cast member property indicates the type of compression that Director will apply to the member when saving the movie in Shockwave format. This property can be tested and set, and has no effect at runtime. Its value can be any one of these symbols:

Value	Meaning
#movieSetting	Use the compression settings of the movie, as stored in the movieImageCompression property. This is the default value for image formats not restricted to standard compression (see below).
#standard	Use Director's standard internal compression format.
#jpeg	Use JPEG compression. See imageQuality.

You normally set this property in the Property Inspector's Bitmap tab. However, if you want to set this property for a large number of images at once, you can set the property with a Lingo routine.

If a member doesn't support JPEG compression because it is 8-bit or lower, or if the image is linked from an external file, only `#standard` compression can be used. Image formats that do not support JPEG compression include GIF and 8-bit or lower images.

Example

This statement displays the imageCompression of member Sunrise in the message window:

```
put member("Sunrise").imageCompression
-- #movieSetting
```

See also

imageQuality, **movieImageCompression**, **movieImageQuality**

imageEnabled

Syntax

```
sprite(whichVectorOrFlashSprite).imageEnabled
the imageEnabled of sprite whichVectorOrFlashSprite
member(whichVectorOrFlashMember).imageEnabled
the imageEnabled of member whichVectorOrFlashMember
```

Description

Cast member property and sprite property; controls whether a Flash movie or vector shape's graphics are visible (`TRUE`, default) or invisible (`FALSE`).

This property can be tested and set.

Example

This `beginSprite` script sets up a linked Flash movie sprite to hide its graphics when it first appears on the Stage and begins to stream into memory and saves its sprite number in a global variable called `gStreamingSprite` for use in a frame script later in the Score:

```
global gStreamingSprite

on beginSprite me
   gStreamingSprite = me.spriteNum
   sprite(gStreamingSprite).imageEnabled = FALSE
end
```

In a later frame of the movie, this frame script checks to see if the Flash movie sprite specified by the global variable `gStreamingSprite` has finished streaming into memory. If it has not, the script keeps the playback head looping in the current frame until 100% of the movie has streamed into memory. It then sets the `imageEnabled` property to `TRUE` so that the graphics appear and lets the playback head continue to the next frame in the Score.

```
global gStreamingSprite

on exitFrame me
  if sprite(gStreamingSprite).member.percentStreamed < 100 then
    go to frame
  else
    sprite(gStreamingSprite).imageEnabled = TRUE
    updatestage
  end if
end
```

imageQuality

Syntax

```
member(whichMember).imageQuality
the imageQuality of member whichMember
```

Description

This bitmap cast member property indicates the level of compression to use when the member's `imageCompression` property is set to `#jpeg`. The range of acceptable values is 0–100. Zero yields the lowest image quality and highest compression; 100 yields the highest image quality and lowest compression.

This property is settable only during authoring and only affects cast members when saving a movie in Shockwave format. The compressed image can be previewed via the Otimize in Fireworks button in the Property Inspector's Bitmap tab or the Preview in Browser command in the File menu.

If an image cast member's `imageCompression` property is set to `#MovieSetting`, the movie property `movieImageQuality` is used instead of `imageQuality`.

See also **imageCompression**, **movieImageCompression**, **movieImageQuality**

importFileInto

Syntax

```
importFileInto member whichCastMember, fileName
importFileInto member whichCastMember of castLib whichCast, fileName
importFileInto member whichCastMember, URL
```

Description

Command; replaces the content of the cast member specified by *whichCastMember* with the file specified by *fileName*.

The `importFileInto` command is useful in four situations:

- When finishing developing a movie, use it to embed media that you have kept linked and external so it can be edited during the project.
- When generating Score from Lingo during movie creation, use it to assign content to new cast members that you created.
- When downloading files from the Internet, use it to download the file at a specific URL and set the file name of linked media. (However, to import a file from a URL, it's usually more efficient and minimizes downloading to use the `preloadNetThing` command to download the file to a local disk first and then import the file from the local disk.)
- Use it to import both RTF and HTML documents into text cast members with formatting and links intact.

Use of the `importFileInto` command in projectors can quickly consume available memory, so it's best to reuse the same members for imported data if possible.

Also, the `importFileInto` command doesn't work with the Director player for Java. To change the content of a bitmap or sound cast member in a movie playing back as an applet, make the cast members linked cast members and change the cast member's `fileName` property.

Note: In Shockwave, you must issue a `preloadNetThing` and wait for a successful completion of the download before using importFileInto with the file. In Director and projectors, `importFileInto` automatically downloads the file for you.

Example

This handler assigns a URL that contains a GIF file to the variable `tempURL` and then uses the `importFileInto` command to import the file at the URL into a new bitmap cast member:

```
on exitFrame
   tempURL = "http://www.dukeOfUrl.com/crown.gif"
  importFileInto new(#bitmap), tempUrl
end
```

Example

This statement replaces the content of the sound cast member Memory with the sound file Wind:

```
importFileInto member "Memory", "Wind.wav"
```

Example

These statements download an external file from a URL to the Director application folder and then import that file into the sound cast member Norma Desmond Speaks:

```
downLoadNetThing http://www.cbDeMille.com/Talkies.AIF, the \
applicationPath&"Talkies.AIF" \
importFileInto(member "Norma Desmond Speaks", the
applicationPath&"Talkies.AIF")
```

See also

downloadNetThing, **fileName (cast member property)**, **preloadNetThing()**

in

See also **number (characters)**, **number (items)**, **number (lines)**, **number (words)**

INF

Description Lingo return value; indicates that a specified Lingo expression evaluates as an infinite number.

See also **NAN**

inflate

Syntax

```
rectangle.Inflate(widthChange , heightChange)
inflate (rectangle, widthChange, heightChange)
```

Description

Command; changes the dimensions of the rectangle specified by *rectangle* relative to the center of the rectangle, either horizontally (*widthChange*) or vertically (*heightChange*).

The total change in each direction is twice the number you specify. For example, replacing *widthChange* with 15 increases the rectangle's width by 30 pixels. A value less than 0 for the horizontal or vertical dimension reduces the rectangle's size.

Example

This statement increases the rectangle's width by 4 pixels and the height by 2 pixels:

```
rect(10, 10, 20, 20).inflate(2, 1)
-- rect (8, 9, 22, 21)
```

Example

This statement decreases the rectangle's height and width by 20 pixels:

```
inflate (rect(0, 0, 100, 100), -10, -10)
-- rect (10, 10, 90, 90)
```

ink

Syntax `sprite(whichSprite).ink`

`the ink of sprite whichSprite`

Description Sprite property; determines the ink effect applied to the sprite specified by *whichSprite*, as follows:

0—Copy	32—Blend
1—Transparent	33—Add pin
2—Reverse	34—Add
3—Ghost	35—Subtract pin
4—Not copy	36—Background transparent
5—Not transparent	37—Lightest
6—Not reverse	38—Subtract
7—Not ghost	39—Darkest
8—Matte	40—Lighten
9—Mask	41—Darken

Description

For a movie that plays back as an applet, valid values for the `ink` sprite property vary for different sprites, as follows:

- For bitmap sprites, the `ink` sprite property can be 0 (Copy), 8 (Matte), 32 (Blend), or 36 (Background transparent).
- For vector shape, Flash, and shape sprites, the `ink` sprite property can be 0, 8, or 36.
- For field sprites, the `ink` sprite property can be 0 or 36. The player treats Blend and Matte inks as Background transparent.

In the case of 36 (background transparent), you select a sprite in the score and select a transparency color from the background color box in the Tools window. You can also do this by setting the `backColor` property.

If you set this property within a script while the playback head is not moving, be sure to use the `updateStage` command to redraw the Stage. If you change several sprite properties—or several sprites—use only one `updateStage` command at the end of all the changes.

This property can be tested and set.

Example This statement changes the variable currentInk to the value for the ink effect of sprite (3):

```
currentInk = sprite(3).ink
```

Example This statement gives sprite (i + 1) a matte ink effect by setting the ink effect of the sprite property to 8, which specifies matte ink:

```
sprite(i + 1).ink = 8
```

See also **backColor**, **foreColor**

inlineImeEnabled

Syntax `the inlineImeEnabled`

Description Global property; determines whether Director's Inline IME feature is turned on. When `TRUE`, this property allows the user to enter double-byte characters directly into Director's Text, Field, Script and Message windows on Japanese systems.

This property can be tested and set. The default value is detemined by the Enable Inline IME setting in Director's General Preferences.

insertFrame

Syntax `insertFrame`

Description Command; duplicates the current frame and its content. The duplicate frame is inserted after the current frame and then becomes the current frame.

This command can be used only during a Score recording session and performs the same function as the `duplicateFrame` command.

Example This handler generates a frame that has the transition cast member Fog assigned in the transition channel followed by a set of empty frames. The argument `numberOfFrames` sets the number of frames.

```
on animBall numberOfFrames
  beginRecording
    the frameTransition = member ("Fog").number
    go the frame + 1
    repeat with i = 0 to numberOfFrames
      insertFrame
    end repeat
  endRecording
end
```

inside()

Syntax `point.inside(`*`rectangle`*`)`

inside(point, rectangle)

Description Function; indicates whether the point specified by *`point`* is within the rectangle specified by *`rectangle`* (`TRUE`), or outside the rectangle (`FALSE`).

Example This statement indicates whether the point Center is within the rectangle Zone and displays the result in the Message window:

```
put Center.inside(Zone)
```

See also **map()**,**mouseH**, **mouseV**, **point()**

installMenu

Syntax `installMenu whichCastMember`

Description Command; installs the menu defined in the field cast member specified by *`whichCastMember`*. These custom menus appear only while the movie is playing. To remove the custom menus, use the `installMenu` command with no argument or with 0 as the argument. This command doesn't work with hierarchical menus.

For an explanation of how menu items are defined in a field cast member, see the `menu` keyword.

Avoid changing menus many times because doing so affects system resources.

In Windows, if the menu is longer than the screen, only part of the menu appears; on the Macintosh, menus longer than the screen can scroll.

Note: Menus are not available in Shockwave.

Example This statement installs the menu defined in field cast member 37:

```
installMenu 37
```

Example This statement installs the menu defined in the field cast member named Menubar:

```
installMenu member "Menubar"
```

Example This statement disables menus that were installed by the `installMenu` command:

```
installMenu 0
```

See also **menu**

integer()

Syntax
```
(numericExpression).integer
integer(numericExpression)
```

Description Function; rounds the value of *numericExpression* to the nearest whole integer.

You can force an integer to be a string by using the `string()` function.

Example This statement rounds off the number 3.75 to the nearest whole integer:

```
put integer(3.75)
-- 4
```

Example This statement rounds off the value in parentheses. This provides a usable value for the `locH` sprite property, which requires an integer:

```
sprite(1).locH = integer(0.333 * stageWidth)
```

See also **float()**,**string()**

integerP()

Syntax

```
expression.integerP
(numericExpression).integerP
integerP(expression)
```

Description Function; indicates whether the expression specified by *expression* can be evaluated to an integer (`1 or TRUE`) or not (`0 or FALSE`). *P* in `integerP` stands for *predicate*.

Example This statement checks whether the number 3 can be evaluated to an integer and then displays 1 (`TRUE`) in the Message window:

```
put(3).integerP
-- 1
```

Example This statement checks whether the number 3 can be evaluated to an integer. Because 3 is surrounded by quotation marks, it cannot be evaluated to an integer, so 0 `(FALSE)` is displayed in the Message window:

```
put("3").integerP
-- 0
```

Example This statement checks whether the numerical value of the string in field cast member Entry is an integer and if it isn't, displays an alert:

```
if field("Entry").value.integerP = FALSE then alert "Please enter
an integer."
```

See also **floatP()**,**integer()**,**ilk()**,**objectP()**,**stringP()**,**symbolP()**

interface()

Syntax `xtra("XtraName").interface()`

`interface(xtra "XtraName")`

Description Function; returns a Return-delimited string that describes the Xtra and lists its methods. This function replaces the now obsolete `mMessageList` function.

Example This statement displays the output from the function used in the QuickTime Asset Xtra in the Message window:

```
put Xtra("QuickTimeSupport").interface()
```

intersect()

Syntax `rectangle1. Intersect(`*`rectangle2`*`)`

`intersect(rectangle1, rectangle2)`

Description Function; determines the rectangle formed where *`rectangle1`* and *`rectangle2`* intersect.

Example This statement assigns the variable `newRectangle` to the rectangle formed where rectangle toolKit intersects rectangle Ramp:

```
newRectangle = toolKit.intersect(Ramp)
```

See also **map()**,**rect()**,**union()**

interval

Syntax

```
member(whichCursorCastMember).interval
```

```
the interval of member whichCursorCastMember
```

Description

Cursor cast member property; specifies the interval, in milliseconds (ms), between each frame of the animated color cursor cast member *whichCursorCastMember*. The default interval is 100 ms.

The cursor interval is independent of the frame rate set for the movie using the tempo channel or the `puppetTempo` Lingo command.

This property can be tested and set.

Example

In this sprite script, when the animated color cursor stored in the cast member named Butterfly enters the sprite, the interval is set to 50 ms to speed up the animation. When the cursor leaves the sprite, the interval is reset to 100 ms to slow down the animation.

```
on mouseEnter
  member("Butterfly").interval = 50
end
```

```
on mouseLeave
  member("Butterfly").interval = 100
end
```

into

This code fragment occurs in a number of Lingo constructs, such as `put...into`.

invertMask

Syntax

```
member(whichQuickTimeMember).invertMask
```

```
the invertMask of member whichQuickTimeMember
```

Description

QuickTime cast member property; determines whether Director draws QuickTime movies in the white pixels of the movie's mask (`TRUE`) or in the black pixels (`FALSE`, default).

This property can be tested and set.

Example

This handler reverses the current setting of the `invertMask` property of a QuickTime movie named Starburst:

```
on toggleMask
  member("Starburst").intertMask = not
member("Starburst").intertMask
end
```

See also

mask

isBusy()

Syntax `sound(channelNum).isBusy()`

Description Function; returns `TRUE` if sound channel *channelNum* is currently playing or pausing a sound, and `FALSE` if it hasn't started playing any of its queued sounds or has been stopped.

To see an example of `isBusy()` used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This Lingo statement checks whether a sound is playing in sound channel 1. If there is, the text of member field is set to "You should hear music now." Otherwise, the text reads "The music has ended."

```
if sound(1).isBusy() then
  member("field").text = "You should hear music now."
else
  member("field").text = "The music has ended."
end if
```

See also **pause() (sound playback)**, **playNext()**,**queue()**,**status**, **stop() (sound)**

on isOKToAttach

Syntax `on isOKToAttach me, spriteType, spriteNum`

Description Built-in handler; you can add this handler to a behavior in order to check the type of sprite the behavior is being attached to and prevent the behavior from being attached to inappropriate sprite types.

When the behavior is attached to a sprite, the handler executes and Director passes to it the type of the sprite and its sprite number. The `me` argument contains a reference to the behavior that is being attached to the sprite.

This handler runs before the `on getPropertyDescriptionList` handler.

The Lingo author can check for two types of sprites. `#graphic` includes all graphic cast members, such as shapes, bitmaps, digital video, text, and so on. `#script` indicates the behavior was attached to the script channel. In this case, the *`spriteNum`* is 1.

For each of these sprite types, the handler must return `TRUE` or `FALSE`. A value of `TRUE` indicates that the behavior can be attached to the sprite. A value of `FALSE` prevents the behavior from being attached to the sprite.

If the behavior contains no `on isOKToAttach` handler, then the behavior can be attached to any sprite or frame.

This handler will only be called during the initial attachment of the behavior to the sprite or script channel and will not be called again when any other changes are made to the sprite.

Example This statement checks the sprite type the behavior is being attached to and returns `TRUE` for any graphic sprite except a shape and `FALSE` for the script channel:

```
on isOKToAttach me, spriteType, spriteNum
  case spriteType of
    #graphic: -- any graphic sprite type
      return sprite(spriteNum).member.type <> #shape
      -- works for everything but shape cast members
    #script: --the frame script channel
      return FALSE -- doesn't work as a frame script
  end case
end
```

isPastCuePoint()

Syntax

```
sprite(spriteNum).isPastCuePoint(cuePointID)
isPastCuePoint(sprite(spriteNum), cuePointID)
sound(channelNum).isPastCuePoint(cuePointID)
```

isPastCuePoint(sound(*channelNum*), *cuePointID*)

Description

Function; determines whether a sprite or sound channel has passed a specified cue point in its media. This function can be used with sound (WAV, AIFF, SND, SWA, AU), QuickTime, or Xtra files that support cue points.

Replace `spriteNum` or `channelNum` with a sprite channel or a sound channel. Shockwave Audio (SWA) sounds can appear as sprites in sprite channels, but they play sound in a sound channel. It is recommended that you refer to SWA sound sprites by their sprite channel number rather than their sound channel number.

Replace `cuePointID` with a reference for a cue point:

- If `cuePointID` is an integer, `isPastCuePoint` returns 1 if the cue point has been passed and 0 if it hasn't been passed.
- If `cuePointID` is a name, `isPastCuePoint` returns the number of cue points passed that have that name.

If the value specified for `cuePointID` doesn't exist in the sprite or sound, the function returns 0.

The number returned by `isPastCuePoint` is based on the absolute position of the sprite in its media. For example, if a sound passes cue point Main and then loops and passes Main again, `isPastCuePoint` returns 1 instead of 2.

When the result of `isPastCuePoint` is treated as a Boolean operator, the function returns `TRUE` if any cue points identified by `cuePointID` have passed and `FALSE` if no cue points are passed.

Example

This statement plays a sound until the third time the cue point Chorus End is passed:

```
if (isPastCuePoint(sound 1, "Chorus End")=3) then
  puppetSound 0
end if
```

Example

This displays information in cast member "field 2" about the music playing in sound channel 1. If the music is not yet past cue point "climax", the text of "field 2" is "This is the beginning of the piece." Otherwise, the text reads "This is the end of the piece."

```
if not sound(1).isPastCuePoint("climax") then
    member("field 2").text = "This is the beginning of the
piece."
  else
    member("field 2").text = "This is the end of the piece."
end if
```

isVRMovie

Syntax

```
member(whichCastMember).isVRMovie
isVRMovie of member whichCastMember
sprite(whichSprite).isVRMovie
isVRMovie of sprite whichSprite
```

Description

QuickTime cast member and sprite property; indicates whether a cast member or sprite is a QuickTime VR movie that has not yet been downloaded (`TRUE`), or whether the cast member or sprite isn't a QuickTime VR movie (`FALSE`).

Testing for this property in anything other than an asset whose type is #quickTimeMedia produces an error message.

This property can be tested but not set.

Example

This handler checks to see if the member of a sprite is a QuickTime movie. If it is, the handler further checks to see if it is a QTVR movie. An alert is posted in any case.

```
on checkForVR theSprite
  if sprite(theSprite).member.type = #quickTimeMedia then
    if sprite(theSprite).isVRMovie then
      alert "This is a QTVR asset."
    else
      alert "This is not a QTVR asset."
    end if
  else
    alert "This is not a QuickTime asset."
  end if
end
```

item...of

Syntax

```
textMemberExpression.item[whichItem]
item whichItem of fieldOrStringVariable
textMemberExpression.item[firstItem..lastItem]
item firstItem to lastItem of fieldOrStringVariable
```

Description

Keyword; specifies an item or range of items in a chunk expression. An item in this case is any sequence of characters delimited by the current delimiter as determined by the `itemDelimiter` property.

The terms *`whichItem`*, *`firstItem`*, and *`lastItem`* must be integers or integer expressions that refer to the position of items in the chunk.

Chunk expressions refer to any character, word, item, or line in any source of strings. Sources of strings include field and text cast members and variables that hold strings.

When the number that specifies the last item is greater than the item's position in the chunk expression, the actual last item is specified instead.

Example

This statement looks for the third item in the chunk expression that consists of names of colors and then displays the result in the Message window:

```
put "red, yellow, blue green, orange".item[3]
-- "blue green"
```

The result is the entire chunk "blue green" because this is the entire chunk between the commas."

Example

This statement looks for the third through fifth items in the chunk expression. Because there are only four items in the chunk expression, only the third item is used and fourth items are returned. The result appears in the Message window.

```
put "red, yellow, blue green, orange".item[3..5]"
-- " blue green, orange"
put item 5 of "red, yellow, blue green, orange"
-- ""
```

Example

This statement inserts the item Desk as the fourth item in the second line of the field cast member All Bids:

```
member("All Bids").line[2].item[4] = "Desk"
```

See also

char...of, **itemDelimiter**, **number (items)**, **word...of**

itemDelimiter

Syntax `the itemDelimiter`

Description System property; indicates the special character used to separate items.

You can use the `itemDelimiter` to parse file names by setting `itemDelimiter` to a backslash (\) in Windows or a colon (:) on the Macintosh. Restore the `itemDelimiter` character to a comma (,) for normal operation.

This function can be tested and set.

Example This handler finds the last component in a Macintosh pathname. The handler first records the current delimiter and then changes the delimiter to a colon (:). When a colon is the delimiter, Lingo can use `the last item of` to determine the last item in the chunk that makes up a Macintosh pathname. Before exiting, the delimiter is reset to its original value.

```
on getLastComponent pathName
  save = the itemDelimiter
  the itemDelimiter = ":"
  f = the last item of pathName
  the itemDelimiter = save
  return f
end
```

kerning

Syntax `member(whichTextMember).kerning`

Description Text cast member property; this property specifies whether the text is automatically kerned when the contents of the text cast member are changed.

When set to `TRUE`, kerning is automatic; when set to `FALSE`, kerning is not done.

This property defaults to `TRUE`.

See also **kerningThreshold**

kerningThreshold

Syntax `member(whichTextMember).kerningThreshold`

Description Text cast member property; this setting controls the size at which automatic kerning takes place in a text cast member. This has an effect only when the kerning property of the text cast member is set to TRUE.

The setting itself is an integer indicating the font point size at which kerning takes place.

This property defaults to 14 points.

See also **kerning**

key()

Syntax `the key`

Description Function; indicates the last key that was pressed. This value is the American National Standards Institute (ANSI) value assigned to the key, not the numerical value.

You can use `the key` in handlers that perform certain actions when the user presses specific keys as shortcuts and other forms of interactivity. When used in a primary event handler, the actions you specify are the first to be executed.

Note: The value of the key isn't updated if the user presses a key while Lingo is in a repeat loop.

Use the sample movie Keyboard Lingo to test which characters correspond to different keys on different keyboards.

Example These statements cause the movie to return to the main menu marker when the user presses the Return key. Because the `keyDownScript` property is set to `checkKey`, the `on prepareMovie` handler makes the `on checkKey` handler the first event handler executed when a key is pressed. The `on checkKey` handler checks whether the Return key is pressed and if it is, navigates to the main menu marker.

```
on prepareMovie
  the keyDownScript = "checkKey"
end prepareMovie
```

```
on checkKey
  if the key = RETURN then go to frame "Main Menu"
end
```

Example This `on keyDown` handler checks whether the last key pressed is the Enter key and if it is, then calls the `on addNumbers` handler:

```
on keyDown
  if the key = RETURN then addNumbers
end keyDown
```

See also **commandDown**, **controlDown**, **keyCode()**,**optionDown**

keyboardFocusSprite

Syntax `set the keyboardFocusSprite = *textSpriteNum*`

Description System property; lets the user set the focus for keyboard input (without controlling the cursor's insertion point) on a particular text sprite currently on the screen. This is the equivalent to using the Tab key when the `AutoTab` property of the member is selected.

Setting `keyboardFocusSprite` to -1 returns keyboard focus control to the score, and setting it to 0 disables keyboard entry into any editable sprite.

See also **autoTab**, **editable**

keyCode()

Syntax `the keyCode`

Description Function; gives the numerical code for the last key pressed. This keyboard code is the key's numerical value, not the American National Standards Institute (ANSI) value.

Note: When a movie plays back as an applet, this function returns the values of only function and arrow keys.

You can use the `keyCode` function to detect when the user has pressed an arrow or function key, which cannot be specified by the `key` function.

Use the sample movie Keyboard Lingo to test which characters correspond to different keys on different keyboards.

This function can be tested but not set.

Example This handler uses the Message window to display the appropriate key code each time a key is pressed:

```
on enterFrame
  the keydownScript = "put the keyCode"
end
```

Example This statement checks whether the up arrow (whose key code is 126) was pressed and if it was, goes to the previous marker:

```
if the keyCode = 126 then go to marker(-1)
```

Example This handler checks whether one of the arrow keys was pressed and if one was, responds accordingly:

```
on keyDown
  case (the keyCode) of
    123: TurnLeft
    126: GoForward
    125: BackUp
    124: TurnRight
  end case
end
```

See also **commandDown**, **controlDown**, **key()**,**optionDown**

on keyDown

Syntax

```
on keyDown
  statement(s)
end
```

Description

System message and event handler; contains statements that run when a key is pressed.

When a key is pressed, Director searches these locations, in order, for an `on keyDown` handler: primary event handler, editable field sprite script, field cast member script, frame script, and movie script. For sprites and cast members, `on keyDown` handlers work only for editable text and field members. A `keyDown` event on a different type of cast member, such as a bitmap, has no effect. (If pressing a key should have the same response throughout the movie, set `keyDownScript`.)

Director stops searching when it reaches the first location that has an `on keyDown` handler, unless the handler includes the `pass` command to explicitly pass the `keyDown` message on to the next location.

The `on keyDown` event handler is a good place to put Lingo that implements keyboard shortcuts or other interface features that you want to occur when the user presses keys.

The Director player for Java responds to `keyDown` messages only if the movie has focus in the browser. The user must click in the applet before the applet can receive any keys that the user types.

When the movie plays back as an applet, an `on keyDown` handler always traps key presses, even if the handler is empty. If the user is typing in an editable field, an `on keyDown` handler attached to the field must include the `pass` command for the key to appear in the field.

Where you place an `on keyDown` handler can affect when it runs.

- To apply the handler to a specific editable field sprite, put the handler in a sprite script.
- To apply the handler to an editable field cast member in general, put the handler in a cast member script.
- To apply the handler to an entire frame, put the handler in a frame script.
- To apply the handler throughout the entire movie, put the handler in a movie script.

You can override an `on keyDown` handler by placing an alternative `on keyDown` handler in a location that Lingo checks before it gets to the handler you want to override. For example, you can override an `on keyDown` handler assigned to a cast member by placing an `on keyDown` handler in a sprite script.

Example

This handler checks whether the Return key was pressed and if it was, sends the playback head to another frame:

```
on keyDown
  if the key = RETURN then go to frame "AddSum"
end keyDown
```

See also

charToNum(),**keyDownScript**, **keyUpScript**, **key()**,**keyCode()**,**keyPressed()**

keyDownScript

Syntax `the keyDownScript`

Description System property; specifies the Lingo that is executed when a key is pressed. The Lingo is written as a string, surrounded by quotation marks, and can be a simple statement or a calling script for a handler.

When a key is pressed and the `keyDownScript` property is defined, Lingo executes the instructions specified for the `keyDownScript` property first. Unless the instructions include the `pass` command so that the `keyDown` message can be passed on to other objects in the movie, no other `on keyDown` handlers are executed.

Setting the `keyDownScript` property performs the same function as using the `when keyDown then` command that appeared in earlier versions of Director.

When the instructions you specify for the `keyDownScript` property are no longer appropriate, turn them off by using the statement `set the keyDownScript to EMPTY`.

Example This statement sets `keyDownScript` to `if the key = RETURN then go to the frame + 1`. When this statement is in effect, the movie always goes to the next frame whenever the user presses the Return key.

```
the keyDownScript = "if the key = RETURN then go to the frame + 1"
```

Example This statement sets `keyDownScript` to the custom handler *`myCustomHandler`*. A Lingo custom handler must be enclosed in quotation marks when used with the `keyDownScript` property.

```
the keyDownScript = "myCustomHandler"
```

See also **on keyDown**, **keyUpScript**, **mouseDownScript**, **mouseUpScript**

keyPressed()

Syntax

```
the keyPressed
keyPressed (keyCode)
keyPressed (asciiCharacterString)
```

Description

Function; returns the character assigned to the key that was last pressed if no argument is used. The result is in the form of a string. When no key has been pressed, `the keyPressed` is an empty string.

If an argument is used, either a keyCode or the ASCII string for the key being pressed may be used. In either of these cases, the return value is `TRUE` if that particular key is being pressed, or `FALSE` if not.

The Director player for Java doesn't support this property. As a result, a movie playing back as an applet has no way to detect which key the user pressed while Lingo is in a repeat loop.

The `keyPressed` property is updated when the user presses keys while Lingo is in a repeat loop. This is an advantage over the `key` function, which doesn't update when Lingo is in a repeat loop.

Use the sample movie Keyboard Lingo to test which characters correspond to different keys on different keyboards.

This property can be tested but not set.

Example

The following statement checks whether the user pressed the Enter key in Windows or the Return key on a Macintosh and runs the handler `updateData` if the key was pressed:

```
if the keyPressed = RETURN then updateData
```

Example

This statement uses the keyCode for the *a* key to test if it's down, and displays the result in the Message window:

```
if keyPressed(0) then put "Key is down"
```

Example

This statement uses the ASCII strings to test if the *a* and *b* keys are down, and displays the result in the Message window:

```
if keyPressed("a") and keyPressed ("b") then put "Keys are down"
```

See also

keyCode(),key()

on keyUp

Syntax

```
on keyUp
  statement(s)
end
```

Description

System message and event handler; contains statements that run when a key is released. The `on keyUp` handler is similar to the `on keyDown` handler, except this event occurs after a character appears if a field or text sprite is editable on the screen.

When a key is released, Lingo searches these locations, in order, for an `on keyUp` handler: primary event handler, editable field sprite script, field cast member script, frame script, and movie script. For sprites and cast members, `on keyUp` handlers work only for editable strings. A `keyUp` event on a different type of cast member, such as a bitmap, has no effect. If releasing a key should always have the same response throughout the movie, set `keyUpScript`.

Lingo stops searching when it reaches the first location that has an `on keyUp` handler, unless the handler includes the `pass` command to explicitly pass the `keyUp` message on to the next location.

The `on keyUp` event handler is a good place to put Lingo that implements keyboard shortcuts or other interface features that you want to occur when the user releases keys.

The Director player for Java responds to `keyUp` messages only if the movie has focus in the browser. The user must click in the applet before the applet can receive any keys that the user types.

When the movie plays back as an applet, an `on keyUp` handler always traps key presses, even if the handler is empty. If the user is typing in an editable field, an `on keyUp` handler attached to the field must include the `pass` command for the key to appear in the field.

Where you place an `on keyUp` handler can affect when it runs, as follows:

- To apply the handler to a specific editable field sprite, put it in a behavior.
- To apply the handler to an editable field cast member in general, put it in a cast member script.
- To apply the handler to an entire frame, put it in a frame script.
- To apply the handler throughout the entire movie, put it in a movie script.

You can override an `on keyUp` handler by placing an alternative `on keyUp` handler in a location that Lingo checks before it gets to the handler you want to override. For example, you can override an `on keyUp` handler assigned to a cast member by placing an `on keyUp` handler in a sprite script.

Example

This handler checks whether the Return key was released and if it was, sends the playback head to another frame:

```
on keyUp
  if the key = RETURN then go to frame "AddSum"
end keyUp
```

See also

on keyDown, **keyDownScript**, **keyUpScript**

keyUpScript

Syntax `the keyUpScript`

Description System property; specifies the Lingo that is executed when a key is released. The Lingo is written as a string, surrounded by quotation marks, and can be a simple statement or a calling script for a handler.

When a key is released and the `keyUpScript` property is defined, Lingo executes the instructions specified for the `keyUpScript` property first. Unless the instructions include the `pass` command so that the `keyUp` message can be passed on to other objects in the movie, no other `on keyUp` handlers are executed.

When the instructions you've specified for the `keyUpScript` property are no longer appropriate, turn them off by using the statement `set the keyUpScript to empty`.

Example This statement sets `keyUpScript` to `if the key = RETURN then go the frame + 1`. When this statement is in effect, the movie always goes to the next frame whenever the user presses the Return key.

```
the keyUpScript = "if the key = RETURN then go to the frame + 1"
```

Example This statement sets `keyUpScript` to the custom handler myCustomHandler. A Lingo custom handler must be enclosed in quotation marks when used with the `keyUpScript` property.

```
the keyUpScript = "myCustomHandler"
```

See also **on keyUp**

label()

Syntax `label(expression)`

Description Function; indicates the frame associated with the marker label specified by *`expression`*. The term *`expression`* should be a label in the current movie; if it's not, this function returns 0.

Example This statement sends the playback head to the tenth frame after the frame labeled Start:

```
go to label("Start") + 10
```

Example This statement assigns the frame number of the fourth item in the label list to the variable `whichFrame`:

```
whichFrame = label(the labelList.line[4])
```

See also **go**, **frameLabel**, **labelList**, **marker()**,**play**

labelList

Syntax
```
the labelList
```

Description
System property; lists the frame labels in the current movie as a Return- delimited string (not a list) containing one label per line. Labels are listed according to their order in the Score. (Because the entries are Return- delimited, the end of the string is an empty line after the last Return. Be sure to remove this empty line if necessary.)

Example
This statement makes a list of frame labels in the content of the field cast member Key Frames:

```
member("Key Frames").text = the labelList
```

Example
This handler determines the label that starts the current scene:

```
on findLastLabel
  aa = label(0)
  repeat with i = 1 to (the labelList.line.count - 1)
    if aa = label(the labelList.line[i]) then
      return the labelList.line[i]
    end if
  end repeat
end
```

See also
frameLabel, **label()**,**marker()**

last()

Syntax
```
the last chunk of ( chunkExpression )
the last chunk in (chunkExpression)
```

Description Function; identifies the last chunk specified by *chunk* in the chunk expression specified by *chunkExpression*.

Chunk expressions refer to any character, word, item, or line in a container of character. Supported containers are field cast members, variables that hold strings, and specified characters, words, items, lines, and ranges within containers.

Example This statement identifies the last word of the string “Macromedia, the multimedia company” and displays the result in the Message window:

```
put the last char of "Macromedia, the multimedia company" "
```

The result is the word *company*.

Example This statement identifies the last character of the string “Macromedia, the multimedia company” and displays the result in the Message window:

```
put last char("Macromedia, the multimedia company")
```

The result is the letter *y*.

See also **char...of**, **word...of**

lastChannel

Syntax
```
the lastChannel
```

Description
Movie property; the number of the last channel in the movie, as entered in the Movie Properties dialog box.

This property can be tested but not set.

To see an example of `lastChannel` used in a completed movie, see the QT and Flash movie in the Learning\Lingo Examples folder inside the Director application folder.

Example
This statement displays the number of the last channel of the movie in the Message window:

```
put the lastChannel
```

lastClick()

Syntax `the lastClick`

Description Function; returns the time in ticks (1 tick = 1/60 of a second) since the mouse button was last pressed.

This function can be tested but not set.

Example This statement checks whether 10 seconds have passed since the last mouse click and, if so, sends the playback head to the marker No Click:

```
if the lastClick > 10 * 60 then go to "No Click"
```

See also **lastEvent()**,**lastKey**, **lastRoll**, **startTimer**

lastEvent()

Syntax `the lastEvent`

Description Function; returns the time in ticks (1 tick = 1/60 of a second) since the last mouse click, rollover, or key press occurred.

Example This statement checks whether 10 seconds have passed since the last mouse click, rollover, or key press, and, if so sends the playback head to the marker Help:

```
if the lastEvent > 10 * 60 then go to "Help"
```

See also **lastClick()**,**lastKey**, **lastRoll**, **startTimer**

lastFrame

Syntax
```
the lastFrame
```

Description
Movie property; displays the number of the last frame in the movie.

This property can be tested but not set.

Example
This statement displays the number of the last frame of the movie in the Message window:

```
put the lastFrame
```

lastKey

Syntax `the lastKey`

Description System property; gives the time in ticks (1 tick = 1/60 of a second) since the last key was pressed.

Example This statement checks whether 10 seconds have passed since the last key was pressed and, if so, sends the playback head to the marker No Key:

```
if the lastKey > 10 * 60 then go to "No Key"
```

See also **lastClick()**,**lastEvent()**,**lastRoll**, **startTimer**

lastRoll

Syntax `the lastRoll`

Description System property; gives the time in ticks (1 tick = 1/60 of a second) since the mouse was last moved.

Example This statement checks whether 45 seconds have passed since the mouse was last moved and, if so, sends the playback head to the marker Attract Loop:

```
if the lastRoll > 45 * 60 then go to "Attract Loop"
```

See also **lastClick()**,**lastEvent()**,**lastKey**, **startTimer**

left

Syntax
```
sprite(whichSprite).left
the left of sprite whichSprite
```

Description
Sprite property; identifies the left horizontal coordinate of the bounding rectangle of the sprite specified by *whichSprite*.

Sprite coordinates are measured in pixels, starting with (0,0) at the upper left corner of the Stage.

When a movie plays back as an applet, this property's value is relative to the left edge of the applet.

This property can be tested and set.

Example
The following statement determines whether the sprite's left edge is to the left of the Stage's left edge. If the sprite's left edge is to the Stage's left edge, the script runs the handler `offLeftEdge`:

```
if sprite(3).left < 0 then offLeftEdge
```

Example
This statement measures the left horizontal coordinate of the sprite numbered (i + 1) and assigns the value to the variable named `vLowest`:

```
set vLowest = sprite (i + 1).left
```

See also
bottom, **height**, **locH**, **locV**, **right**, **top**, **width**

leftIndent

Syntax `chunkExpression.leftIndent`

Description Text cast member property; contains the number of pixels the left margin of *`chunkExpression`* is offset from the left side of the text cast member.

The value is an integer greater than or equal to 0.

This property can be tested and set.

Example This line indents the first line of text cast member “theStory” by ten pixels:

```
member("theStory").line[1].leftIndent = 10
```

See also **firstIndent**, **rightIndent**

length()

Syntax

```
string.length
length(string)
```

Description Function; returns the number of characters in the string specified by *string*, including spaces and control characters such as TAB and RETURN.

Example This statement displays the number of characters in the string “Macro”&“media”:

```
put ("Macro" & "media").length
-- 10
```

Example This statement checks whether the content of the field cast member File Name has more than 31 characters and if it does, displays an alert:

```
if member("File Name").text.length > 31 then
  alert "That file name is too long."
end if"
```

See also **chars()**,**offset() (string function)**

line...of

Syntax

```
textMemberExpression.line[whichLine]
line whichLine of fieldOrStringVariable
textMemberExpression.line[firstLine..lastLine]
line firstLine to lastLine of fieldOrStringVariable
```

Description

Keyword; specifies a line or a range of lines in a chunk expression. A line chunk is any sequence of characters delimited by carriage returns, not by line breaks caused by text wrapping.

The expressions *whichLine*, *firstLine*, and *lastLine* must be integers that specify a line in the chunk.

Chunk expressions refer to any character, word, item, or line in any source of characters. Sources of characters include field cast members and variables that hold strings.

Example

This statement assigns the first four lines of the variable `Action` to the field cast member To Do:

```
member("To Do").text = Action.line[1..4]
```

Example

This statement inserts the word `and` after the second word of the third line of the string assigned to the variable `Notes`:

```
put "and" after Notes.line[3].word[2]
```

See also

char...of, **item...of**, **word...of**, **and**, **number (words)**

lineCount

Syntax

```
member(whichCastMember).lineCount
the lineCount of member whichCastMember
```

Description

Cast member property; indicates the number of lines that appear in the field cast member on the Stage according to the way the string wraps, not the number of carriage returns in the string.

Example

This statement determines how many lines the field cast member Today's News has when it appears on the Stage and assigns the value to the variable `numberOfLines`:

```
numberOfLines = member("Today's News").lineCount
```

lineDirection

Syntax `member(whichCastMember).lineDirection`

Description Shape member property; this property contains a 0 or 1 indicating the slope of the line drawn.

If the line is inclined from left to right, the property is set to 1; and if it is declined from left to right, the property is set to 0.

This property can be tested and set.

Example This toggles the slope of the line in cast member “theLine”, producing a see- saw effect:

```
on seeSaw
  member("theLine").lineDirection = \
  not member("theLine").lineDirection
end
```

lineHeight() (function)

Syntax

```
member(whichCastMember).lineHeight(lineNumber)
lineHeight(member whichCastMember, lineNumber)
```

Description

Function; returns the height, in pixels, of a specific line in the specified field cast member.

Example

This statement determines the height, in pixels, of the first line in the field cast member Today's News and assigns the result to the variable `headline`:

```
headline = member("Today's News").lineHeight(1)
```

lineHeight (cast member property)

Syntax

```
member(whichCastMember).lineHeight
the lineHeight of member whichCastMember
```

Description

Cast member property; determines the line spacing used to display the specified field cast member. The parameter *whichCastMember* can be either a cast member name or number.

Setting the `lineHeight` member property temporarily overrides the system's setting until the movie closes. To use the desired line spacing throughout a movie, set the `lineHeight` member property in an `on prepareMovie` handler.

This property can be tested and set.

Example

This statement sets the variable `oldHeight` to the current `lineHeight` setting for the field cast member Rokujo Speaks:

```
oldHeight = member("Rokujo Speaks").lineHeight
```

See also

text; **alignment**, **font**, **fontSize**, **fontStyle**

linePosToLocV()

Syntax `member(whichCastMember).linePosToLocV(lineNumber )`

`linePosToLocV(member whichCastMember, lineNumber)`

Description Function; returns a specific line's distance, in pixels, from the top edge of the field cast member.

Example This statement measures the distance, in pixels, from the second line of the field cast member Today's News to the top of the field cast member and assigns the result to the variable `startOfString`:

```
startOfString = member("Today's News").linePosToLocV(2)
```

lineSize

Syntax

```
member(whichCastMember).lineSize
the lineSize of member whichCastMember
sprite whichSprite.lineSize
the lineSize of sprite whichSprite
```

Description

Shape cast member property; determines the thickness, in pixels, of the border of the specified shape cast member displayed on the Stage. For nonrectangular shapes, the border is the edge of the shape, not its bounding rectangle.

The `lineSize` setting `of` the sprite takes precedence over the `lineSize` setting `of` the member. If Lingo changes the member's `lineSize` setting while a sprite is on the Stage, the sprite's `lineSize` setting remains in effect until the sprite is finished.

For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet.

This property can be tested and set.

Example

This statement sets the thickness of the shape cast member Answer Box to 5 pixels:

```
member("Answer Box").lineSize = 5
```

Example

This statement displays the thickness of the border of sprite 4:

```
thickness = sprite(4).lineSize
```

Example

This statement sets the thickness of the border of sprite 4 to 3 pixels:

```
sprite(4).lineSize = 3
```

linkAs()

Syntax `castMember.linkAs()`

Description Script cast member function; opens a save dialog box, allowing you to save the contents of the script to an external file. The script cast member is then linked to that file.

Linked scripts are imported into the movie when you save it as a projector, Shockwave movie, or Java movie. This differs from other linked media, which remains external to the movie unless you explicitly import it.

Example This statement, typed in the Message window, opens a save dialog box to save the script Random Motion as an external file:

```
member("Random Motion").linkAs()
```

See also **importFileInto**, **linked**

linked

Syntax

```
member(whichMember).linked
the linked of member whichMember
```

Description

Cast member property; controls whether a script, Flash movie, or animated GIF file is stored in an external file (`TRUE`, default), or inside the Director cast (`FALSE`). When the data is stored externally, the cast member's `pathName` property must point to the location where the movie file can be found.

This property can be tested and set for script, Flash, and GIF members. It may be tested for all member types.

Example

This converts Flash cast member "Homebodies" from a linked member to an internally stored member.

```
member("homeBodies").linked = 0
```

See also

fileName (cast member property), **pathName (cast member property)**

list()

Syntax `list(value1, value2, value3...)`

Description Function and data type; defines a linear list made up of the values specified by *value1*, *value2*, *value3*.... This is an alternative to using square brackets (`[ ]`) to create a list.

The maximum length of a single line of executable Lingo is 256 characters. You can't create a very large list using this command. If you have a large amount of data that you want to put in a list, enclose the data in square brackets and put the data into a field. You can then assign the field to a variable. The variable's content is a list of the data.

Example This statement sets the variable named designers equal to a linear list that contains the names Gee, Kayne, and Ohashi:

```
designers = list("Gee", "Kayne", "Ohashi")
```

The result is the list ["Gee", "Kayne", "Ohashi"].

See also **integer()**, l**integerP()**,**value()**

listP()

Syntax `listP(item)`

Description Function; indicates whether the item specified by *item* is a list, rectangle, or point (1 or TRUE) or not (0 or FALSE).

Example This statement checks whether the list in the variable `designers` is a list, rectangle, or point, and displays the result in the Message window:

```
put listP(designers)
```

The result is 1, which is the numerical equivalent of `TRUE`.

See also **ilk()**,**objectP()**

loaded

Syntax
```
member(whichCastMember).loaded
the loaded of member whichCastMember
```

Description Cast member property; specifies whether the cast member specified by *whichCastMember* is loaded into memory (TRUE) or not (FALSE).

Different cast member types have slightly different behaviors for loading:

- Shape and script cast members are always loaded into memory.
- Movie cast members are never unloaded.
- Digital video cast members can be preloaded and unloaded independent of whether they are being used. (A digital video cast member plays faster from memory than from disk.)

This property can be tested but not set.

Example This statement checks whether cast member Demo Movie is loaded in memory and if it isn't, goes to an alternative movie:

```
if member("Demo Movie").loaded = FALSE then go to
movie("Waiting")"
```

See also **preLoad (command)**, **ramNeeded()**,**size**, **unLoad**

loc

Syntax

```
sprite whichSprite.loc
the loc of sprite whichSprite
```

Description

Sprite property; determines the Stage coordinates of the specified sprite's registration point. The value is given as a point.

This `property` can be tested and set.

To see an example of `loc` used in a completed movie, see the Imaging movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement checks the Stage coordinates of sprite 6. The result is the point (50, 100):

```
put sprite(6).loc
-- point(50, 100)
```

See also

bottom, **height**, **left**, **locV**, **right**, **top**, **width**

locH

Syntax
```
sprite(whichSprite).locH
the locH of sprite whichSprite
```

Description Sprite property; indicates the horizontal position of the specified sprite's registration point. Sprite coordinates are relative to the upper left corner of the Stage.

This property can be tested and set. To make the value last beyond the current sprite, make the sprite a puppet.

Example This statement checks whether the horizontal position of sprite 9's registration point is to the right of the right edge of the stage and moves the sprite to the left edge of the Stage if it is:

```
if sprite(9).locH > (the stageRight - the stageLeft) then
  sprite(9).locH = 0
end if
```

Example This statement puts sprite 15 at the same horizontal location as the mouse click:

```
sprite(15).locH = the mouseH
```

See also **bottom**, **height**, **left**, **locV**, **point()**,**right**, **top**, **width**, **updateStage**

locToCharPos()

Syntax

```
member(whichCastMember). locToCharPos(location )
locToCharPos(member whichCastMember, location)
```

Description

Function; returns a number that identifies which character in the specified field cast member is closest to the point within the field specified by *location*. The value for *location* is a point relative to the upper left corner of the field cast member.

The value 1 corresponds to the first character in the string, the value 2 corresponds to the second character in the string, and so on.

Example

This statement determines which character is closest to the point 100 pixels to the right and 100 pixels below the upper left corner of the field cast member Today's News. The statement then assigns the result to the variable `PageDesign`.

```
pageDesign = member("Today's News").locToCharPos(point(100, 100))
```

Example

This handler tells which character is under the cursor when the user clicks the mouse over the field sprite Information:

```
on mouseDown
  put member("Information").locToCharPos(the clickLoc - \
  (sprite(the clickOn).loc))
end
```

locV

Syntax

```
sprite(whichSprite).locV
```

```
the locV of sprite whichSprite
```

Description

Sprite property; indicates the vertical position of the specified sprite's registration point. Sprite coordinates are relative to the upper left corner of the Stage.

This property can be tested and set. To make the value last beyond the current sprite, make the sprite a puppet.

Example

This statement checks whether the vertical position of sprite 9's registration point is below the bottom of the Stage and moves the sprite to the top of the Stage if it is:

```
if sprite(9).locV > (the stageBottom - the stageTop) then
  sprite(9).locV = 0
end if
```

Example

This statement puts sprite 15 at the same vertical location as the mouse click:

```
sprite(15).locV = the mouseV
```

See also

bottom, **height**, **left**, **locH**, **right**, **top**, **width**, **updateStage**

locVToLinePos()

Syntax `member(whichCastMember). locVToLinePos(locV )`

locVToLinePos(member *whichCastMember*, *locV*)

Description Function; returns the number of the line of characters that appears at the vertical position specified by `locV`. The `locV` value is the number of pixels from the top of the field cast member, not the part of the field cast member that currently appears on the Stage.

Example This statement determines which line of characters appears 150 pixels from the top of the field cast member Today's News and assigns the result to the variable `pageBreak`:

```
pageBreak = member("Today's News").locVToLinePos(150)
```

locZ of sprite

Syntax `sprite(whichSprite).locZ`

Description Sprite property; specifies the dynamic Z-order of a sprite, to control layering without having to manipulate sprite channels and properties.

This property can be tested and set.

This property can have an integer value from negative 2 billion to positive 2 billion. Larger numbers cause the sprite to appear in front of sprites with smaller numbers. If two sprites have the same `locZ` value, the channel number then takes precedence for deciding the final display order of those two sprites. This means sprites in lower numbered channels appear behind sprites in higher numbered channels even when the `locZ` values are equal.

By default, each sprite has a `locZ` value equal to it's own channel number.

Layer-dependent operations such as hit detection and mouse events obey sprites' `locZ` values, so changing a sprite's `locZ` value can make the sprite partially or completely obscured by other sprites and the user may be unable to click on the sprite.

Other Director functions do not follow the `locZ` ordering of sprites. Generated events still begin with channel 1 and increase consecutively from there, regardless of the sprite's Z-order.

Example This handler uses a global variable called gHighestSprite which has been initialized in the startMovie handler to the number of sprites used. When the sprite is clicked, its locZ is set to gHighestSprite + 1, which moves the sprite to the foreground on the stage. Then gHighestSprite is incremented by 1 to prepare for the next mouseUp call.

```
on mouseUp me
  global gHighestSprite
  sprite(me.spriteNum).locZ = gHighestSprite + 1
  gHighestSprite = gHighestSprite + 1
end
```

See also **locH**, **locV**

log()

Syntax `log(number)`

Description Math function; calculates the natural logarithm of the number specified by *`number`*, which must be a decimal number greater than 0.

Example This statement assigns the natural logarithm of 10.5 to the variable `Answer`.

```
Answer = log(10.5)
```

Example This statement calculates the natural logarithm of the square root of the value `Number` and then assigns the result to the variable `Answer`:

```
Answer = log(Number.sqrt)
```

long

See **date() (system clock)**, **time()**functions

loop (keyword)

Syntax `loop`

Description Keyword; refers to the marker. The `loop` keyword with the `go to` command is equivalent to the statement `go to the marker`.

Example This handler loops the movie between the previous marker and the current frame:

```
on exitFrame
  go loop
end exitFrame
```

loop (cast member property)

Syntax
```
member(whichCastMember).loop
the loop of member whichCastMember
```

Description
Cast member property; determines whether the specified digital video, sound, or Flash movie cast member is set to loop (`TRUE`) or not (`FALSE`).

Example
This statement sets the QuickTime movie cast member Demo to loop:

```
member("Demo").loop = 1
```

loop (Flash property)

Syntax

```
sprite(whichFlashSprite).loop
the loop of sprite whichFlashSprite
member (whichFlashMember).loop
the loop of member whichFlashMember
```

Description

Flash sprite and member property; controls whether a Flash movie plays in a a continuous loop (`TRUE`) or plays once and then stops (`FALSE`).

The property can be both tested and set.

Example

This frame script checks the download status of a linked Flash cast member called NetFlash using the `percentStreamed` property. While NetFlash is downloading, the movie loops in the current frame. When NetFlash finishes downloading, the movie advances to the next frame and the `loop` property of the Flash movie in channel 6 is set to `FALSE` so that it will continue playing through to the end and then stop (imagine that this sprite has been looping while NetFlash was downloading).

```
on exitFrame
  if member("NetFlash").percentStreamed = 100 then
    sprite(6).loop = FALSE
    go the frame + 1
  end if
  go the frame
end
```

loopBounds

Syntax
```
sprite(whichQuickTimeSprite).loopBounds
the loopBounds of sprite whichQuickTimeSprite
```

Description QuickTime sprite property; sets the internal loop points for a QuickTime cast member or sprite. The loop points are specified as a Director list: [*startTime*, *endTime*].

The *startTime* and *endTime* parameters must meet these requirements:

- Both parameters must be integers that specify times in Director ticks.
- The values must range from 0 to the duration of the QuickTime cast member.
- The starting time must be less than the ending time.

If any of these requirements is not met, the QuickTime movie loops through its entire duration.

The `loopBounds` property has no effect if the movie's `loop` property is set to `FALSE`. If the `loop` property is set to `TRUE` while the movie is playing, the movie continues to play. Director uses these rules to decide how to loop the movie:

- If the ending time specified by `loopBounds` is reached, the movie loops back to the starting time.
- If the end of the movie is reached, the movie loops back to the start of the movie.

If the `loop` property is turned off while the movie is playing, the movie continues to play. Director stops when it reaches the end of the movie.

This property can be tested and set. The default setting is [0,0].

Example This sprite script sets the starting and ending times for looping within a QuickTime sprite. Notice that the times are set by specifying seconds, which are then converted to ticks by multiplying by 60.

```
on beginSprite me
  sprite(me.spriteNum).loopBounds = [(16 * 60),(32 * 60)]
end
```

loopCount

Syntax

```
sound(channelNum).loopCount
the loopCount of sound channelNum
```

Description

Cast member property; the total number of times the current sound in sound channel *channelNum* is set to loop. The default is 1 for sounds that are simply queued with no internal loop.

You can loop a portion of a sound by passing the parameters `loopStartTime`, `loopEndTime`, and `loopCount` with a `queue()` or `setPlayList()` command. These are the only methods for setting this property.

If `loopCount` is set to 0, the loop will repeat forever. If the sound cast member's `loop` property is set to `TRUE`, the `loopCount` will return 0.

Example

This handler queues and plays two sounds in sound channel 2. The first sound, cast member introMusic, loops five times between 8 seconds and 8.9 seconds. The second sound, cast member creditsMusic, loops three times. However, no `#loopStartTime` and `#loopEndTime` are specified, so these values default to the `#startTime` and `#endTime`, respectively.

```
on playMusic
  sound(2).queue([#member:member("introMusic"), #startTime:3000,\
  #loopCount:5,#loopStartTime:8000, #loopEndTime:8900])
  sound(2).queue([#member:member("creditsMusic"),
#startTime:3000,\
  #endTime:8000, #loopCount:3])
  sound(2).play()
end
```

Example

This handler displays an alert indicating how many times the loop in the cast member of sound 2 plays. If no loop has been set in the current sound of sound channel 2, `sound(2).loopCount` returns 1.

```
on showLoopCount
  alert "The current sound's loop plays" && sound(2).loopCount &&
"times."
end
```

See also

breakLoop(), **setPlaylist()**, **loopEndTime**, **loopsRemaining**, **loopStartTime**, **queue()**

loopEndTime

Syntax
```
sound(channelNum).loopEndTime
the loopEndTime of sound channelNum
```

Description Sound property; the end time, in milliseconds, of the loop set in the current sound playing in sound channel *channelNum*. Its value is a floating-point number, allowing you to measure and control sound playback to fractions of a millisecond.

This property can only be set when passed as a property in a `queue()` or `setPlaylist()` command.

Example This handler plays sound cast member introMusic in sound channel 2. Playback loops five times between the 8 seconds point and the 8.9 second point in the sound.

```
on playMusic
  sound(2).play([#member:member("introMusic"), #startTime:3000,\
  #loopCount:5,#loopStartTime:8000, #loopEndTime:8900])
end
```

Example This handler causes the text field TimWords to read "Help me, I'm stuck!" when the `currentTime` of sound channel 2 is between its `loopStartTime` and `loopEndTime`.

```
on idle
  if sound(2).currentTime > sound(2).loopStartTime and \
  sound(2).currentTime < sound(2).loopEndTime then
       member("TimWords").text = "Help me, I'm stuck!"
  else
       member("TimWords").text = "What's this sticky stuff?"
  end if
end
```

See also **breakLoop()**,**getPlaylist()**,**loopCount**, **loopsRemaining**, **loopStartTime**, **queue()**

loopsRemaining

Syntax
```
sound(channelNum).loopsRemaining
the loopsRemaining of sound(channelNum)
```

Description Read-only property; the number of times left to play a loop in the current sound playing in sound channel *channelNum*. If the sound had no loop specified when it was queued, this property is 0. If this property is tested immediately after a sound starts playing, it returns one less than the number of loops defined with the `#loopCount` property in the `queue()` or `setPlayList()` command.

See also **breakLoop()**,**loopCount**, **loopEndTime**, **loopStartTime**, **queue()**

loopStartTime

Syntax

```
sound(channelNum).loopStartTime
the loopStartTime of sound(channelNum)
```

Description

Cast member property; the start time, in milliseconds, of the loop for the current sound being played by *soundObject*. Its value is a floating-point number, allowing you to measure and control sound playback to fractions of a millisecond. The default is the `startTime` of the sound if no loop has been defined.

This property can only be set when passed as a property in a `queue()` or `setPlaylist()` command.

Example

This handler plays sound cast member introMusic in sound channel 2. Playback loops five times between two points 8 seconds and 8.9 seconds into the sound.

```
on playMusic
  sound(2).play([#member:member("introMusic"), #startTime:3000,\
  #loopCount:5,#loopStartTime:8000, #loopEndTime:8900])
end
```

Example

This handler causes the text field TimWords to read "Help me, I'm stuck!" when the `currentTime` of sound channel 2 is between its `loopStartTime` and `loopEndTime`:

```
on idle
  if sound(2).currentTime > sound(2).loopStartTime and \
  sound(2).currentTime < sound(2).loopEndTime then
    member("TimWords").text = "Help me, I'm stuck!"
  else
    member("TimWords").text = "What's this sticky stuff?"
  end if
end
```

See also

breakLoop(), **setPlaylist()**, **loopCount**, **loopEndTime**, **loopsRemaining**, **queue()**

map()

Syntax

```
map(targetRect, sourceRect, destinationRect)
map(targetPoint, sourceRect, destinationRect)
```

Description

Function; positions and sizes a rectangle or point based on the relationship of a source rectangle to a target rectangle.

The relationship of the `targetRect` to the `sourceRect` governs the relationship of the result of the function to the `destinationRect`.

Example

In this behavior, all of the sprites have already been set to draggable. Sprite 2b contains a small bitmap. Sprite 1s is a rectangular shape sprite large enough to easily contain sprite 2b. Sprite 4b is a larger version of the bitmap in sprite 2b. Sprite 3s is a larger version of the shape in sprite 1s. Moving sprite 2b or sprite 1s will cause sprite 4b to move. When you drag sprite 2b, its movements are mirrored by sprite 4b. When you drag sprite 1s, sprite 4b moves in the opposite direction. Resizing sprite 2b or sprite 1s will also produce interesting results.

```
on exitFrame
  sprite(4b).rect = map(sprite(2b).rect, sprite(1s).rect,
sprite(3s).rect)
  go the frame
end
```

To see a demonstration of map(), see the **Lingo map()** movie.

mapMemberToStage()

Syntax `sprite(whichSpriteNumber). mapMemberToStage(whichPointInMember)`

`mapMemberToStage(sprite whichSpriteNumber, whichPointInMember)`

Description Function; uses the specified sprite and point to return an equivalent point inside the dimensions of the Stage. This properly accounts for the current transformations to the sprite using `quad`, or the rectangle if not transformed.

This is useful for determining if a particular area of a cast member has been clicked, even if there have been major transformations to the sprite on the Stage.

If the specified point on the Stage is not within the sprite, a `VOID` is returned.

See also **map()**,**mapStageToMember()**

mapStageToMember()

Syntax
```
sprite(whichSpriteNumber). mapStageToMember(whichPointOnStage)
mapStageToMember(sprite whichSpriteNumber, whichPointOnStage)
```

Description

Function; uses the specified sprite and point to return an equivalent point inside the dimensions of the cast member. This properly accounts for any current transformations to the sprite using `quad`, or the rectangle if not transformed.

This is useful for determining if a particular area on a cast member has been clicked even if there have been major transformations to the sprite on the Stage.

If the specified point on the Stage is not within the sprite, this function returns `VOID`.

See also **map()**,**mapMemberToStage()**

margin

Syntax

```
member(whichCastMember).margin
the margin of member whichCastMember
```

Description

Field cast member property; determines the size, in pixels, of the margin inside the field box.

Example

The following statement sets the margin inside the box for the field cast member Today's News to 15 pixels:

```
member("Today's News").margin = 15
```

marker()

Syntax
```
marker(integerExpression)
marker("string")
```

Description Function; returns the frame number of markers before or after the current frame. This function is useful for implementing a Next or Previous button or for setting up an animation loop.

The argument *integerExpression* can evaluate to any positive or negative integer or 0. For example:

- `marker(2)`—Returns the frame number of the second marker after the current frame.
- `marker(1)`—Returns the frame number of the first marker after the current frame.
- `marker(0)`—Returns the frame number of the current frame if the current frame is marked, or the frame number of the previous marker if the current frame is not marked.
- `marker(-1)`—Returns the frame number of the first marker before the marker(0).
- `marker(-2)`—Returns the frame number of the second marker before the marker(0).

If the argument for `marker` is a string, `marker` returns the frame number of the first frame whose marker label matches the string.

Example This statement sends the playback head to the beginning of the current frame if the current frame has a marker, otherwise it sends the playback head to the previous marker:

```
go to marker(0)
```

Example This statement sets the variable `nextMarker` equal to the next marker in the Score:

```
nextMarker = marker(1)
```

See also **go**, **frame() (function)**, **frameLabel**, **label()**,**labelList**

the markerList

Syntax
```
the markerList
```

Description
Global property; contains a Lingo property list of the markers in the Score. The list is of the format:

```
frameNumber: "markerName"
```

This property can be tested but not set.

Example
This statement displays the list of markers in the Message window:

```
put the markerlist
-- [1: "Opening Credits", 15: "Main Menu", 26: "Closing Credits"]
```

See also
marker()

mask

Syntax

```
member(whichQuickTimeMember).mask
the mask of member whichQuickTimeMember
```

Description

Cast member property; specifies a black-and-white (1-bit) cast member to be used as a mask for media rendered direct to Stage with media appearing in the areas where the mask's pixels are black. The `mask` property lets you benefit from the performance advantages of a direct-to-Stage digital video while playing a QuickTime movie in a nonrectangular area. The `mask` property has no effect on non–direct-to-Stage cast members.

Director always aligns the registration point of the mask cast member with the upper left of the QuickTime movie sprite. Be sure to reset the registration point of a bitmap to the upper left corner, as it defaults to the center. The registration point of the QuickTime member cannot be reset from the upper left corner. The mask cast member can't be moved and is not affected by the `center` and `crop` properties of its associated cast member.

For best results, set a QuickTime cast member's mask property before any of its sprites appear on the Stage in the `on beginSprite` event handler. Setting or changing the `mask` property while the cast member is on the Stage can have unpredictable results (for example, the mask may appear as a freeze frame of the digital video at the moment the `mask` property took effect).

Masking is an advanced feature; you may need to experiment to achieve your goal.

This property can be tested and set. To remove a mask, set the `mask` property to 0.

Example

This frame script sets a mask for a QuickTime sprite before Director begins to draw the frame:

```
on prepareFrame
  member("Peeping Tom").mask = member("Keyhole")
end
```

See also

invertMask

max()

Syntax

```
list.max()
max(list)
max(value1, value2, value3, ...)
```

Description

Function; returns the highest value in the specified list or the highest of a given series of values.

The `max` function also works with ASCII characters, similar to the way < and > operators work with strings.

Example

This handler assigns the variable `Winner` the maximum value in the list `Bids`, which consists of [#Castle:600, #Schmitz:750, #Wang:230]. The result is then inserted into the content of the field cast member Congratulations.

```
on findWinner Bids
  Winner = Bids.max()
  member("Congratulations").text = \
  "You have won, with a bid of $" & Winner &"!"
end
```

maxInteger

Syntax

```
the maxInteger
```

Description

System property; returns the largest whole number that is supported by the system. On most personal computers, this is 2,147,483,647 (2 to the thirty-first power, minus 1).

This property can be useful for initializing boundary variables before a loop or for limit testing.

To use numbers larger than the range of addressable integers, use floating- point numbers instead. They aren't processed as quickly as integers, but they support a greater range of values.

Example

This statement generates a table, in the Message window, of the maximum decimal value that can be represented by a certain number of binary digits:

```
on showMaxValues
  b = 31
  v = the maxInteger
  repeat while v > 0
    put b && "-" && v
    b = b-1
    v = v/2
  end repeat
end
```

mci

Syntax `mci "string"`

Description Command; for Windows only, passes the strings specified by *`string`* to the Windows Media Control Interface (MCI) for control of multimedia extensions.

Note: Microsoft no longer recommends using the 16-bit MCI interface. Consider using third-party Xtras for this functionality instead.

Example This statement makes the command `play cdaudio from 200 to 600 track 7` play only when the movie plays back in Windows:

```
mci "play cdaudio from 200 to 600 track 7"
```

me

Syntax `me`

Description Special variable; used within parent scripts and behaviors to refer to the current object that is an instance of the parent script or the behavior or a variable that contains the memory address of the object.

The term has no predefined meaning in Lingo. The term `me` is used by convention.

To see an example of `me` used in a completed movie, see the Parent Scripts movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement sets the object `myBird1` to the script named Bird. The `me` keyword accepts the parameter script Bird and is used to return that parameter.

```
myBird1 = new(script "Bird")
```

This is the `on new` handler of the Bird script:

```
on new me
  return me
end
```

Example The following two sets of handlers make up a parent script. The first set uses `me` to refer to the child object. The second set uses the variable `myAddress` to refer to the child object. In all other respects, the parent scripts are the same.

This is the first set:

```
property myData
on new me, theData
   myData = theData
  return me
end
on stepFrame me
  ProcessData me
end
```

This is the second set:

```
property myData
on new myAddress, theData
  myData = theData
  return myAddress
end
on stepFrame myAddress
  ProcessData myAddress
end
```

See also **new()**,**ancestor**

media

Syntax

```
member(whichCastMember).media
the media of member whichCastMember
```

Description

Cast member property; identifies the specified cast member as a set of numbers. Because setting the `media` member property can use large amounts of memory, this property is best used during authoring only.

You can use the `media` member property to copy the content of one cast member into another cast member by setting the second cast member's `media` value to the `media` value for the first cast member.

For a film loop cast member, the `media` member property specifies a selection of frames and channels in the Score.

To swap media in a projector, it's more efficient to set the `member` sprite property.

Example

This statement copies the content of the cast member Sunrise into the cast member Dawn by setting the `media` member property value for Dawn to the `media` member property value for Sunrise:

```
member("Dawn").media = member("Sunrise").media
```

See also

type (cast member property), **media**

mediaReady

Syntax

```
member(whichCastMember).mediaReady
the mediaReady of member whichCastMember
sprite(whichSpriteNumber).mediaReady
the mediaReady of sprite whichSpriteNumber
```

Description

Cast member or sprite property; determines whether the contents of the sprite, the specified cast member, or a movie or cast file, or linked cast member is downloaded from the Internet and is available on the local disk (`TRUE`) or is not available (`FALSE`).

This property is useful only when streaming a movie or cast file. Movie streaming is activated by setting the Movie:Playback properties in the Modify menu to Play While Downloading Movie (default setting).

For a demonstration of the `mediaReady` member function, see the **mediaReady** movie.

This property can be tested but not set.

Example

This statement changes cast members when the desired cast member is downloaded and available locally:

```
if member("background").mediaReady = TRUE then
    sprite(2).memberNum = 10
  -- 10 is the number of cast member "background"
end if
```

See also

frameReady()

member (keyword)

Syntax

```
member whichCastMember
member whichCastMember of castLib whichCast
member(whichCastMember, whichCastLib)
```

Description

Keyword; indicates that the object specified by *whichCastMember* is a cast member. If *whichCastMember* is a string, it is used as the cast member name. If *whichCastMember* is an integer, it is used as the cast member number.

When playing back a movie as an applet, refer to cast members by number rather than by name to improve the applet's performance.

The `member` keyword is a specific reference to both a castLib and a member within it if used alone:

```
put sprite(12).member
-- (member 3 of castLib 2)
```

This property differs from the `memberNum` property of a sprite, which is always an integer designating position in a castLib but does not specify the castLib:

```
put sprite(12).memberNum
-- 3
```

The number of a member is also an absolute reference to a particular member in a particular castLib:

```
put sprite(12).member.number
-- 131075
```

Example

The following statement sets the `hilite` property of the button cast member named Enter Bid to `TRUE`:

```
member("Enter Bid").hilite = TRUE
```

Example

This statement puts the name of sound cast member 132 into the variable `soundName`:

```
put member(132, "Viva Las Vegas").name
```

Example

This statement checks the type of member Jefferson Portrait in the castLib Presidents:

```
memberType = member("Jefferson Portrait", "Presidents").type
```

Example

This statement determines whether cast member 9 has a name assigned:

```
if member(9).name = EMPTY then exit
```

Example

You can check for the existence of a member by testing for its number:

```
memberCheck = member("Epiphany").number
if memberCheck = -1 then alert "Sorry, that member doesn't exist"
```

Example

Alternatively, you can check for the existence of a member by testing for its type:

```
memberCheck = member("Epiphany").type
if memberCheck = #empty then alert "Sorry, that member doesn't
exist"
```

See also

memberNum

member (sound property)

Syntax

```
sound(channelNum).member
the member of sound(channelNum)
```

Description

This read-only property is the sound cast member currently playing in sound channel *channelNum*. This property returns null if no sound is being played.

Example

This statement displays the name of the member of the sound playing in sound channel 2 in the message window.

```
put sound(2).member
-- (member 4 of castLib 1)
```

See also

getPlaylist(),**queue()**

member (sprite property)

Syntax

```
sprite(whichSprite).member
the member of sprite whichSprite
```

Description

Sprite property; specifies a sprite's cast member and cast.

The `member` sprite property differs from the `memberNum` sprite property, which specifies only the sprite's number to identify its location in the cast but doesn't specify the cast itself. The `member` sprite property also differs from `mouseMember` and the obsolete `castNum` sprite properties, neither of which specifies the sprite's cast.

When assigning a sprite's `member` property, use one of the following formats:

- Specify the full member and cast description (`sprite(x).member = member(A, B)`.
- Specify the cast member name (sprite(x).member = member ("MELODY ").
- Specify the unique integer that includes all cast libraries and corresponds to the mouseMember function (`sprite(x).member = 132`).

If you use only the cast member name, Director finds the first cast member that has that name in all current casts. If the name is duplicated in two casts, only the first name is used.

To specify a cast member by number when there are multiple casts, use the `memberNum` sprite property, which changes the member's position in its cast without affecting the sprite's cast (`set the memberNum of sprite x to 132`).

You can determine the `memberNum` sprite property from the `member` sprite property by using the phrase `the number of the member of sprite x`. You can also retrieve other cast member properties by using phrases such as `the name of the member of sprite x` or `the rect of the member of sprite x`.

The cast member assigned to a sprite channel is only one of that sprite's properties; other properties vary by the type of media element in that channel in the Score. For example, if you replace a bitmap with an unfilled shape by setting the `member` sprite property, the shape sprite's `lineSize` sprite property doesn't automatically change, and you probably won't see the shape.

Similar sprite property mismatches can occur if you change the member of a field sprite to a video. Although you can change all sprite properties through the `type` sprite property, it's generally more useful and predictable to replace cast members with similar cast members. For example, replace bitmap sprites with bitmap cast members.

This property can be tested and set.

Example

This statement assigns cast member 3 of cast number 4 to sprite 15:

```
sprite(15).member = member(3, 4)
```

Example

The following handler uses the mouseMember function with the sprite.member property to find if the mouse is over a particular sprite:

```
on exitFrame
  MM = the mouseMember
  target = sprite(1).member
  if target = MM then put "above the hotspot"
  go the frame
end
```

See also

castLibNum, **memberNum**

memberNum

Syntax

```
sprite(whichSprite).memberNum
```

```
the memberNum of sprite whichSprite
```

Description

Sprite property; identifies the position of the cast member (but doesn't identify the castLib) associated with the specified sprite *whichSprite*. Its value is the cast member number only; the cast member's cast is not specified.

The `memberNum` property is useful for switching cast members assigned to a sprite so long as the cast members are within the same cast. To switch among cast members in different casts, use the `member` sprite property. For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet.

This property also is useful for exchanging cast members when a sprite is clicked to simulate the reversed image that appears when a standard button is clicked. You can also make some action in the movie depend on which cast member is assigned to a sprite.

When you set this property within a script while the playback head is not moving, be sure to use the `updateStage` command to redraw the Stage.

This property can be tested and set.

Example

The following statement switches the cast member assigned to sprite 3 to cast member number 35 in the same cast:

```
sprite(3).memberNum = 35
```

Example

This statement assigns the cast member Narrator to sprite 10 by setting `memberNum` sprite property to Narrator's cast number. Narrator is in the same cast as the sprite's current cast member.

```
sprite(10).memberNum = member("Narrator").number
```

Example

This handler swaps bitmaps when a button is clicked or rolled off. It assumes that the artwork for the Down button immediately follows the artwork for the Up button in the same cast.

```
on mouseDown
  upButton = sprite(the clickOn).memberNum
  downButton = upButton + 1
  repeat while the stillDown
    if rollover(the clickOn) then
      sprite(the clickOn).memberNum = downButton
    else
      sprite(the clickOn).memberNum = upButton
    end if
    updateStage
  end repeat
  if rollover (the clickOn) then put "The button was activated"
end
```

See also

castLib, **member (sprite property)**, **number (cast member property)**, **member (keyword)**

members

See **number of members**

memorySize

Syntax `the memorySize`

Description System property; returns the total amount of memory allocated to the program, whether in use or free memory. This property is useful for checking minimum memory requirements. The value is given in bytes.

In Windows, the value is the total physical memory available; on the Macintosh, the value is the entire partition assigned to the application.

Example This statement checks whether the computer allocates less than 500K of memory and if it does, displays an alert:

```
if the memorySize < 500 * 1024 then alert "There is not enough
memory to run this movie."
```

See also **freeBlock()**,**freeBytes()**,**ramNeeded()**,**size**

menu

Syntax

```
menu: menuName
itemName | script
itemName | script
...
```

or

```
menu: menuName
itemName | script
itemName | script
...
[more menus]
```

Description

Keyword; in conjunction with the `installMenu` command, specifies the actual content of custom menus. Field cast members contain menu definitions; refer to them by the cast member name or number.

The `menu` keyword is followed immediately by a colon, a space, and the name of the menu. In subsequent lines, specify the menu items for that menu. You can set a script to execute when the user chooses an item by placing the script after the vertical bar symbol (|). A new menu is defined by the subsequent occurrence of the `menu` keyword.

Note: Menus are not available in Shockwave.

On the Macintosh, you can use special characters to define custom menus. These special characters are case sensitive. For example, to make a menu item bold, the letter *B* must be uppercase.

Special symbols should follow the item name and precede the vertical bar symbol (|). You can also use more than one special character to define a menu item. Using <B<U, for example, sets the style to Bold and Underline.

Avoid special character formatting for cross-platform movies because not all Windows computers support it.

Symbol	Example	Description
@	menu: @	*On the Macintosh, creates the Apple symbol and enables Macintosh menu bar items when you define an Apple menu.
!Ã	!ÃEasy Select	*On the Macintosh, checks the menu with a check mark (Option+v).
<B	Bold<B	*On the Macintosh, sets the menu item's style to Bold.
<I	Italic<I	*On the Macintosh, sets the style to Italic.
<U	Underline<U	*On the Macintosh, sets the style to Underline.
<O	Outline<O	*On the Macintosh, sets the style to Outline.
<S	Shadow<S	*On the Macintosh, sets the style to Shadow.
|	Open/O | go to frame "Open"	Associates a script with the menu item.

/	Quit/Q	Defines a command-key equivalent.
(	Save(	Disables the menu item.
(-	(-	Creates a disabled line in the menu.

Note: * identifies formatting tags that work only on the Macintosh.

Example This is the text of a field cast member named CustomMenu2 which can be used to specify the content of a custom File menu. To install this menu, use "`installMenu member("CustomMenu2")`" while the movie is running. The Convert menu item runs the custom handler `convertThis`.

```
menu: File
Open/O | go to frame "Open"
Close/W | go to frame "Close"
Convert/C | convertThis
   (-
Quit/Q | go to frame "Quit"
```

See also **installMenu**, **name (menu property)**, **name (menu item property)**, **number (menu items)**, **checkMark**, **enabled**, **script**

milliseconds

Syntax `the milliseconds`

Description System property; returns the current time in milliseconds (1/1000 of a second). Counting begins from the time the computer is started.

Example This statement converts milliseconds to seconds and minutes by dividing the number of milliseconds by 1000 and dividing that result by 60, and then sets the variable `currentMinutes` to the result:

```
currentSeconds = milliseconds/1000
currentMinutes = currentseconds/60
```

The resolution accuracy of the count is machine and operating system dependent.

Example This handler counts the milliseconds and posts an alert if you've been working too long.

```
on idle
  if the milliseconds > 1000 * 60 * 60 * 4 then
    alert "Take a break"
  end if
end
```

See also **ticks**, **time()**,**timer**

min

Syntax

```
list.min
```

min(*list*)

```
min (a1, a2, a3...)
```

Description

Function; specifies the minimum value in the list specified by *list*.

Example

This handler assigns the variable `vLowest` the minimum value in the list bids, which consists of [#Castle:600, #Shields:750, #Wang:230]. The result is then inserted in the content of the field cast member Sorry:

```
on findLowest bids
  vLowest = bids.min()
  member("Sorry").text = \
  "We're sorry, your bid of $" & vLowest && "is not a winner!"
end
```

See also

max()

missingFonts

Syntax `member(textCastMember).missingFonts`

Description Text cast member property; this property contains a list of the names of the fonts that are referenced in the text, but not currently available on the system.

This allows the developer to determine during run time if a particular font is available or not.

This property can be tested but not set.

See also **substituteFont**

mod

Syntax `integerExpression1 mod integerExpression2`

Description Math operator; performs the arithmetic modulus operation on two integer expressions. In this operation, *integerExpression1* is divided by *integerExpression2*.

The resulting value of the entire expression is the integer remainder of the division. It always has the sign of *integerExpression1*.

This is an arithmetic operator with a precedence level of 4.

Example This statement divides 7 by 4 and then displays the remainder in the Message window:

```
put 7 mod 4
```

The result is 3.

Example This handler sets the ink effect of all odd-numbered sprites to `copy`, which is the ink effect specified by the number 0. First the handler checks whether the sprite in the variable `mySprite` is an odd-numbered sprite by dividing the sprite number by 2 and then checking whether the remainder is 1. If the remainder is 1, the result for an odd-numbered number, the handler sets the ink effect to `copy`.

```
on setInk
  repeat with mySprite = 1 to the lastChannel
    if (mySprite mod 2) = 1 then
      sprite(mySprite).ink = 0
    else
      sprite(mySprite).ink = 8
    end if
  end repeat
end setInk
```

Example This handler regularly cycles a sprite's cast member among a number of bitmaps:

```
on exitFrame
  global gCounter
  -- These are sample values for bitmap cast member numbers
  theBitmaps = [2,3,4,5,6,7]
  -- Specify which sprite channel is affected
  theChannel = 1
  -- This cycles through the list
  gCounter = 1 + (gCounter mod theBitmaps.count)
  sprite(theChannel).memberNum = theBitmaps[gCounter]
  go the frame
end
```

modal

Syntax

```
window "window".modal
the modal of window "window"
```

Description Window property; specifies whether movies can respond to events that occur outside the window specified by *window*.

- When the `modal` window property is `TRUE`, movies cannot respond to events outside the window.
- When the `modal` window property is `FALSE`, movies can respond to events outside the window.

  Setting the `modal` window property to `TRUE` lets you make a specific movie in a window the only movie that the user can interact with.

  Be aware that this property works even in the authoring environment. If you set the `modal` window property to `TRUE`, you will not be able to interact with Director's windows either.

  You can always close a window that is modal by using Control+Alt+period (Windows) or Command+period (Macintosh).

Example This statement lets movies respond to events outside of the Tool Panel window:

```
window("Tool Panel").modal = FALSE
```

modified

Syntax
```
member(whichCastMember).modified
the modified of member whichCastMember
```

Description Cast member property; indicates whether the cast member specified by *whichCastMember* has been modified since it was read from the movie file.

- When the `modified` member property is `TRUE` (`1`), the cast member has been modified since it was read from the movie file.
- When the `modified` member property is `FALSE` (`0`), the cast member has not been modified since it was read from the movie file.

The `modified` member function always returns `FALSE` for a cast member in a movie that plays back as an applet.

Example This statement tests whether the cast member Introduction has been modified since it was read from the movie file:

```
return member("Introduction").modified
```

modifiedBy

Syntax `member.modifiedBy`

the modifiedBy of *member*

Description Cast member property; records the name of the user who last edited the cast member, taken from the user name information provided during Director installation. You can change this information in Director's General Preferences dialog box.

This property can be tested but not set. It is useful for tracking and coordinating Director projects with more than one author, and may also be viewed in the Property Inspector's Member tab.

Example This statement displays the name of the person who last modified cast member 1:

```
put member(1).modifiedBy
  -- "Sam Sein"
```

See also **comments**, **creationDate**, **modifiedDate**

modifiedDate

Syntax `member.modifiedDate`

the modifiedDate of *member*

Description Cast member property; indicates the date and time that the cast member was last changed, using the system time on the authoring computer. This property is useful for tracking and coordinating Director projects.

This property can be tested but not set. It can also be viewed in the Property Inspector's Member tab and the Cast window list view.

Example This statement displays the date of the last change to cast member 1:

```
put member(1).modifiedDate
  -- date( 1999, 12, 8 )
```

See also **comments**, **creationDate**, **modifiedBy**

mostRecentCuePoint

Syntax

```
sprite(whichSprite).mostRecentCuePoint
the mostRecentCuePoint of sprite whichSprite
sound(channelNum).mostRecentCuePoint
the mostRecentCuePoint of sound channelNum
```

Description

Cast member, sound channel, and sprite property; for sound cast members, QuickTime digital video, and Xtras that support cue points, indicates the number that identifies the most recent cue point passed in the sprite or sound. The value is the cue point's ordinal number. If no cue points have been passed, the value is 0.

For sound channels, the return value is for the sound cast member currently playing in the sound channel.

Shockwave Audio (SWA) sounds can appear as sprites in sprite channels, but they play sound in a sound channel. It is recommended that you refer to SWA sound sprites by their sprite channel number rather than their sound channel number.

Example

This statement tells the Message window to display the number for the most recent cue point passed in the sprite in sprite channel 1:

```
put sprite(1).mostRecentCuePoint
```

Example

This statement returns the ordinal number of the most recently passed cue point in the currently playing sound in sound channel 2:

```
put sound(2).mostRecentCuePoint
```

See also

cuePointNames, **isPastCuePoint()**

motionQuality

Syntax `sprite(whichQTVRSprite).motionQuality`

motionQuality of sprite *whichQTVRSprite*

Description QuickTime VR sprite property; the codec quality used when the user clicks and drags the QuickTime VR sprite. The property's value can be `#minQuality`, `#maxQuality`, or `#normalQuality`.

This property can be tested and set.

Example This sets the `motionQuality` of sprite 1 to `#minQuality`.

```
sprite(1).motionQuality = #minQuality
```

mouseChar

Syntax `the mouseChar`

Description System property; used for field sprites, contains the number of the character that is under the cursor when the property is called. The count is from the beginning of the field. If the mouse is not over a field or is in the gutter of a field, the result is -1.

The value of the `mouseChar` property can change in a handler or repeat loop. If a handler or repeat loop uses this property multiple times, it's usually a good idea to call the property once and assign its value to a local variable.

Example This statement determines whether the cursor is over a field sprite and changes the content of the field cast member Instructions to "Please point to a character." when it is:

```
if the mouseChar = -1 then
  member("Instructions").text = "Please point to a character."
```

Example This statement assigns the character under the cursor in the specified field to the variable `currentChar`:

```
currentChar = member(the mouseMember).char[the mouseChar]
```

Example This handler highlights the character under the cursor when the sprite contains a text cast member:

```
property spriteNum
on mouseWithin me
  if sprite(spriteNum).member.type = #field then
    MC = the mousechar
    if MC < 1 then exit    -- if over a border, final line, etc
    hilite char MC of field sprite(spriteNum).member
    else alert "Sorry, 'hilite' and 'mouseChar' are for fields."
end
```

See also **mouseItem**, **mouseLine**, **mouseWord**, **char...of**, **number (characters)**

on mouseDown (event handler)

Syntax

```
on mouseDown
  statement(s)
end
```

Description

System message and event handler; contains statements that run when the mouse button is pressed.

When the mouse button is pressed, Lingo searches the following locations, in order, for an `on mouseDown` handler: primary event handler, sprite script, cast member script, frame script, and movie script. Lingo stops searching when it reaches the first location that has an `on mouseDown` handler, unless the handler includes the `pass` command to explicitly pass the `mouseDown` message on to the next location.

To have the same response throughout the movie when pressing the mouse button, set `mouseDownScript` or put a mouseDown handler in a Movie script.

The `on mouseDown` event handler is a good place to put Lingo that flashes images, triggers sound effects, or makes sprites move when the user presses the mouse button.

Where you place an `on mouseDown` handler can affect when it runs.

- To apply the handler to a specific sprite, put it in a sprite script.
- To apply the handler to a cast member in general, put it in a cast member script.
- To apply the handler to an entire frame, put it in a frame script.
- To apply the handler throughout the entire movie, put it in a movie script.

You can override an `on mouseDown` handler by placing an alternative `on mouseDown` handler in a location that Lingo checks before it gets to the handler you want to override. For example, you can override an `on mouseDown` handler assigned to a cast member by placing an `on mouseDown` handler in a sprite script.

If used in a behavior, this event is passed the sprite script or frame script reference `me`.

Example

This handler checks whether the user clicks anywhere on the Stage and sends the playback head to another frame if a click occurs:

```
on mouseDown
  if the clickOn = 0 then go to frame "AddSum"
end
```

Example

This handler, assigned to a sprite script, plays a sound when the sprite is clicked:

```
on mouseDown
  puppetSound "Crickets"
end
```

See also

clickOn, **mouseDownScript**, **mouseUpScript**

the mouseDown (system property)

Syntax
```
the mouseDown
```

Description
System property; indicates whether the mouse button is currently being pressed (`TRUE`) or not (`FALSE`).

The Director player for Java doesn't update the `mouseDown` property when Lingo is in a repeat loop. If you try to test `mouseDown` inside a repeat loop in an applet, the applet hangs.

Example
This handler causes the movie to beep until the user clicks the mouse:

```
on enterFrame
  repeat while the mouseDown = FALSE
    beep
  end repeat
end
```

Example
This statement instructs Lingo to exit the repeat loop or handler it is in when the user clicks the mouse:

```
if the mouseDown then exit
```

See also
mouseH, **the mouseUp (system property)**, **mouseV**, **on mouseDown (event handler)**, **on mouseUp (event handler)**

mouseDownScript

Syntax `the mouseDownScript`

Description System property; specifies the Lingo that is executed when the mouse button is pressed. The Lingo is written as a string, surrounded by quotation marks and can be a simple statement or a calling script for a handler. The default value is `EMPTY`, which means that the `mouseDownScript` property has no Lingo assigned to it.

When the mouse button is pressed and the `mouseDownScript` property is defined, Lingo executes the instructions specified for the `mouseDownScript` property first. No other `on mouseDown` handlers are executed, unless the instructions include the `pass` command so that the `mouseDown` message can be passed to other objects in the movie.

Setting the `mouseDownScript` property performs the same function as the `when keyDown then` command in earlier versions of Director.

To turn off the instructions you've specified for the `mouseDownScript` property, use the statement `set the mouseDownScript to empty`.

This property can be tested and set.

Example In this statement, when the user clicks the mouse button, the playback head always branches to the next marker in the movie:

```
the mouseDownScript = "go next"
```

Example In this statement, when the user clicks anywhere on the Stage, the computer beeps:

```
the mouseDownScript = "if the clickOn = 0 then beep"
```

Example This statement sets `mouseDownScript` to the custom handler *`myCustomHandler`*. A Lingo custom handler must be enclosed in quotation marks when used with the `mouseDownScript` property.

```
the mouseDownScript = "myCustomHandler"
```

See also **stopEvent**, **mouseUpScript**, **on mouseDown (event handler)**, **on mouseUp (event handler)**

on mouseEnter

Syntax

```
on mouseEnter
  statement(s)
end
```

Description

System message and event handler; contains statements that run when the mouse pointer first contacts the active area of the sprite. The mouse button does not have to be pressed.

If the sprite is a bitmap cast member with matte ink applied, the active area is the portion of the image that is displayed; otherwise, the active area is the sprite's bounding rectangle.

If used in a behavior, this event is passed the sprite script or frame script reference `me`.

Example

This statement is a simple button behavior that switches the bitmap of the button when the mouse rolls over and then off the button:

```
property spriteNum
on mouseEnter me
  -- Determine current cast member and switch to next in cast
  currentMember = sprite(spritenum).member.number
  sprite(spritenum).member = currentMember + 1
end
on mouseLeave me
  -- Determine current cast member and switch to previous in cast
  currentMember = sprite(spritenum).member.number
  sprite(spritenum).member = currentMember - 1
end
```

See also

on mouseLeave, **on mouseWithin**

mouseH

Syntax

```
the mouseH
```

```
mouseH()
```

Description

System property and function; indicates the horizontal position of the mouse pointer. The value of `mouseH` is the number of pixels the cursor is positioned from the left edge of the Stage.

The `mouseH` function is useful for moving sprites to the horizontal position of the mouse pointer and checking whether the pointer is within a region of the Stage. Using the `mouseH` and `mouseV` functions together, you can determine the cursor's exact location.

The Director player for Java doesn't update the `mouseH` function when Lingo is in a repeat loop.

This function can be tested but not set.

Example

This handler moves sprite 10 to the mouse pointer location and updates the Stage when the user clicks the mouse button:

```
on mouseDown
  sprite(10).locH = the mouseH
  sprite(10).locV = the mouseV
  updateStage
end
```

Example

This statement tests whether the cursor is more than 10 pixels to the right or left of a starting point and if it is, sets the variable `Far` to `TRUE`:

```
if abs(mouseH() - startH) > 10 then
  draggedEnough = TRUE
```

See also

locH, **locV**, **mouseV**, **mouseLoc**

mouseItem

Syntax `the mouseItem`

Description System property; contains the number of the item under the pointer when the function is called and the cursor is over a field sprite. (An item is any sequence of characters delimited by the current delimiter as set by `the itemDelimiter` property.) Counting starts at the beginning of the field. If the mouse is not over a field, the result is -1.

The value of the `mouseItem` property can change in a handler or repeat loop. If a handler or repeat loop relies on the initial value of `mouseItem` when the handler or repeat loop begins, call the property once and assign its value to a local variable.

Example This statement determines whether the cursor is over a field sprite and changes the content of the field cast member Instructions to "Please point to an item." when it is not:

```
if the mouseItem = -1 then
  member("Instructions") = "Please point to an item."
end if
```

Example This statement assigns the item under the cursor in the specified field to the variable `currentItem`:

```
currentItem = member(mouseMember).item(mouseItem)
```

Example This handler highlights the item under the cursor when the mouse button is pressed:

```
on mouseDown
  thisField = sprite(the clickOn).member
  if the mouseItem < 1 then exit
  lastItem = 0
  repeat while the stillDown
    MI = the mouseItem
    if MI < 1 then next repeat
    if MI <> lastItem then
      thisField.item[MI].hilite()
      lastItem = MI
    end if
  end repeat
end
```

See also **item...of**, **mouseChar**, **mouseLine**, **mouseWord**, **number (items)**, **itemDelimiter**

on mouseLeave

Syntax

```
on mouseLeave
  statement(s)
end
```

Description

System message and event handler; contains statements that run when the mouse leaves the active area of the sprite. The mouse button does not have to be pressed.

If the sprite is a bitmap cast member with the matte ink applied, the active area is the portion of the image that is displayed; otherwise, the active area is the sprite's bounding rectangle.

If used in a behavior, this event is passed the sprite script or frame script reference `me`.

Example

This statement shows a simple button behavior that switches the bitmap of the button when the mouse rolls over and then back off the button:

```
property spriteNum
on mouseEnter me
  -- Determine current cast member and switch to next in cast
  currentMember = sprite(spritenum).member.number
  sprite(spritenum).member = currentMember + 1
end
on mouseLeave me
  -- Determine current cast member and switch to previous in cast
  currentMember = sprite(spritenum).member.number
  sprite(spritenum).member = currentMember - 1
end
```

See also

on mouseEnter, **on mouseWithin**

mouseLevel

Syntax `sprite(whichQuickTimeSprite).mouseLevel`

`the mouseLevel of sprite` *`whichQuickTimeSprite`*

Description QuickTime sprite property; controls how Director passes mouse clicks on a QuickTime sprite to QuickTime. The ability to pass mouse clicks within the sprite's bounding rectangle can be useful for interactive media such as QuickTime VR. The `mouseLevel` sprite property can have these values:

- `#controller`—Passes clicks only on the movie controller to QuickTime. Director responds only to mouse clicks that occur outside the controller. This is the standard behavior for QuickTime sprites other than QuickTime VR.
- `#all`—Passes all mouse clicks within the sprite's bounding rectangle to QuickTime. No clicks pass to other Lingo handlers.
- `#none`—Does not pass any mouse clicks to QuickTime. Director responds to all mouse clicks.
- `#shared`—Passes all mouse clicks within a QuickTime VR sprite's bounding rectangle to QuickTime and then passes these events to Lingo handlers. This is the default value for QuickTime VR.

This property can be tested and set.

Example This frame script checks to see if the name of the QuickTime sprite in channel 5 contains the string "QTVR." If it does, this script sets `mouseLevel` to `#all`; otherwise, it sets `mouseLevel` to `#none`.

```
on prepareFrame
  if sprite(5).member.name contains "QTVR" then
    sprite(5).mouseLevel = #all
  else
    sprite(5).mouseLevel = #none
  end if
end
```

mouseLine

Syntax `the mouseLine`

Description System property; contains the number of the line under the pointer when the property is called and the cursor is over a field sprite. Counting starts at the beginning of the field; a line is defined by Return delimiter, not by the wrapping at the edge of the field. When the mouse is not over a field sprite, the result is -1.

The value of the `mouseLine` property can change in a handler or repeat loop. If a handler or repeat loop uses this property multiple times, it's usually a good idea to call the property once and assign its value to a local variable.

Example This statement determines whether the cursor is over a field sprite and changes the content of the field cast member Instructions to "Please point to a line." when it is not:

```
if the mouseLine = -1 then
  member("Instructions").text = "Please point to a line."
```

Example This statement assigns the contents of the line under the cursor in the specified field to the variable `currentLine`:

```
currentLine = member(the mouseMember).line[the mouseLine]
```

Example This handler highlights the line under the cursor when the mouse button is pressed:

```
on mouseDown
  thisField = sprite(the clickOn).member
  if the mouseLine < 1 then exit
  lastLine = 0
  repeat while the stillDown
    ML = the mouseLine
    if ML < 1 then next repeat
    if ML <> lastLine then
      thisField.line[ML].hilite()
      lastLine = ML
    end if
  end repeat
end
```

See also **mouseChar**, **mouseItem**, **mouseWord**, **line...of**, **number (lines)**

mouseLoc

Syntax `the mouseLoc`

Description Function; returns the current position of the mouse as a point().

The point location is given as two coordinates, with the horizontal location first, then the vertical location.

See also **mouseH**, **mouseV**

mouseMember

Syntax `the mouseMember`

Description System property; returns the cast member assigned to the sprite that is under the pointer when the property is called. When the pointer is not over a sprite, it returns the result `VOID`.

The `mouseMember` property replaces `mouseCast`, used in earlier versions of Director.

You can use this function to make a movie perform specific actions when the pointer rolls over a sprite and the sprite uses a certain cast member.

The value of the `mouseMember` property can change frequently. To use this property multiple times in a handler with a consistent value, assign the `mouseMember` value to a local variable when the handler starts and use the variable.

For casts other than cast 1, `mouseMember` returns a value that does not distinguish between the cast member and the cast number. To distinguish the cast member and the cast number, use the expression `member (the mouseMember);` if the user doesn't click a sprite, however, this expression generates a script error.

Example This statement checks whether the cast member Off Limits is the cast member assigned to the sprite under the cursor and displays an alert if it is. This example shows how you can specify an action based on the cast member assigned to the sprite.

```
if the mouseMember = member "Off Limits" then alert "Stay away
from there!"
```

Example This statement assigns the cast member of the sprite under the cursor to the variable `lastMember`:

```
lastMember = the mouseMember
```

See also **member (sprite property)**, **memberNum**, **clickLoc**, **mouseChar**, **mouseItem**, **mouseLine**, **mouseWord**, **rollOver()**,**number (cast member property)**

mouseOverButton

Syntax

```
sprite whichFlashSprite.mouseOverButton
the mouseOverButton of sprite whichFlashSprite
```

Description

Flash sprite property; indicates whether the mouse pointer is over a button in a Flash movie sprite specified by the *whichFlashSprite* parameter (TRUE), or whether the mouse pointer is outside the bounds of the sprite or the mouse pointer is within the bounds of the sprite but over a nonbutton object, such as the background (FALSE).

This property can be tested but not set.

Example

This frame script checks to see if the mouse pointer is over a navigation button in the Flash movie in sprite 3. If the mouse pointer is over the button, the script updates a text field with an appropriate message; otherwise, the script clears the message.

```
on enterFrame
  case sprite(3).mouseOverButton of
    TRUE:
      member("Message Line").text = "Click here to go to the next
page."
    FALSE:
      member("Message Line").text = " "
  end case
  updatestage
end
```

on mouseUp (event handler)

Syntax

```
on mouseUp
  statement(s)
end
```

Description

System message and event handler; contains statements that are activated when the mouse button is released.

When the mouse button is released, Lingo searches the following locations, in order, for an `on mouseUp` handler: primary event handler, sprite script, cast member script, frame script, and movie script. Lingo stops searching when it reaches the first location that has an `on mouseUp` handler, unless the handler includes the `pass` command to explicitly pass the `mouseUp` message on to the next location.

To create the same response throughout the movie when the user releases the mouse button, set `the mouseUpScript`.

An `on mouseUp` event handler is a good place to put Lingo that changes the appearance of objects—such as buttons—after they are clicked. You can do this by switching the cast member assigned to the sprite after the sprite is clicked and the mouse button is released.

Where you place an `on mouseUp` handler can affect when it runs, as follows:

- To apply the handler to a specific sprite, put it in a sprite script.
- To apply the handler to a cast member in general, put it in a cast member script.
- To apply the handler to an entire frame, put it in a frame script.
- To apply the handler throughout the entire movie, put it in a movie script.

You can override an `on mouseUp` handler by placing an alternative `on mouseUp` handler in a location that Lingo checks before it gets to the handler you want to override. For example, you can override an `on mouseUp` handler assigned to a cast member by placing an `on mouseUp` handler in a sprite script.

If used in a behavior, this event is passed the sprite script or frame script reference `me`.

Example

This handler, assigned to sprite 10, switches the cast member assigned to sprite 10 when the user releases the mouse button after clicking the sprite:

```
on mouseUp
  sprite(10).member = member "Dimmed"
end
```

See also

on mouseDown (event handler)

the mouseUp (system property)

Syntax `the mouseUp`

Description System property; indicates whether the mouse button is released (`TRUE`) or is being pressed (`FALSE`).

The Director player for Java doesn't update the `mouseUp` property when Lingo is in a repeat loop.

Example This handler causes the movie to run as long as the user presses the mouse button. The playback head stops when the user releases the mouse button.

```
on exitFrame me
  if the mouseUp then
    go the frame
  end if
end
```

Example This statement instructs Lingo to exit the repeat loop or handler it is in when the user releases the mouse button:

```
if the mouseUp then exit
```

See also **the mouseDown (system property)**, **mouseH**, **mouseV**, **the mouseUp (system property)**

on mouseUpOutside

Syntax

```
on mouseUpOutside me
  statement(s)
end
```

Description

System message and event handler; sent when the user presses the mouse button on a sprite but releases it (away from) the sprite.

Example

This statement plays a sound when the user clicks the mouse over a sprite and then releases it outside the bounding rectangle of the sprite:

```
on mouseUpOutside me
  puppetSound "Professor Long Hair"
end
```

See also

on mouseEnter, **on mouseLeave**, **on mouseWithin**

mouseUpScript

Syntax `the mouseUpScript`

Description System property; determines the Lingo that is executed when the mouse button is released. The Lingo is written as a string, surrounded by quotation marks, and can be a simple statement or a calling script for a handler.

When the mouse button is released and the `mouseUpScript` property is defined, Lingo executes the instructions specified for the `mouseUpScript` property first. Unless the instructions include the `pass` command so that the `mouseUp` message can be passed on to other objects in the movie, no other `on mouseUp` handlers are executed.

When the instructions you've specified for the `mouseUpScript` property are no longer appropriate, turn them off by using the statement `set the mouseUpScript to empty`.

Setting the `mouseUpScript` property accomplishes the same thing as using the `when mouseUp then` command that appeared in earlier versions of Director.

This property can be tested and set. The default value is `EMPTY`.

Example When this statement is in effect and the movie is paused, the movie always continues whenever the user releases the mouse button:

```
the mouseUpScript = "go to the frame +1"
```

Example With this statement, when the user releases the mouse button after clicking anywhere on the Stage, the movie beeps:

```
the mouseUpScript = "if the clickOn = 0 then beep"
```

Example This statement sets `mouseUpScript` to the custom handler *`myCustomHandler`*. A Lingo custom handler must be enclosed in quotation marks when used with the `mouseUpScript` property.

```
the mouseUpScript = "myCustomHandler"
```

See also **stopEvent**, **mouseDownScript**, **on mouseDown (event handler)**, **on mouseUp (event handler)**

mouseV

Syntax

```
the mouseV
```

```
mouseV()
```

Description

System property; indicates the vertical position of the mouse cursor, in pixels, from the top of the Stage. The value increases as the cursor moves down and decreases as the cursor moves up.

The `mouseV` property is useful for moving sprites to the vertical position of the mouse cursor and checking whether the cursor is within a region of the Stage. Using the `mouseH` and `mouseV` properties together, you can identify the cursor's exact location.

The Director player for Java doesn't update the `mouseV` property when Lingo is in a repeat loop.

This property can be tested but not set.

Example

This handler moves sprite 1 to the mouse pointer location and updates the Stage when the user clicks the mouse button:

```
on mouseDown
  sprite(1).locH = the mouseH
  sprite(1).locV = the mouseV
  updateStage
end
```

Example

This statement tests whether the pointer is more than 10 pixels above or below a starting point and if it is, sets the variable `vFar` to `TRUE`:

```
if abs(the mouseV - startV) > 10 then draggedEnough = TRUE
```

See also

mouseH, **locH**, **locV**, **mouseLoc**

on mouseWithin

Syntax

```
on mouseWithin
  statement(s)
end
```

Description

System message and event handler; contains statements that run when the mouse is within the active area of the sprite. The mouse button does not have to be pressed.

If the sprite is a bitmap cast member with the matte ink applied, the active area is the portion of the image that is displayed; otherwise, the sprite's bounding rectangle is the active area.

If used in a behavior, this event is passed the sprite script or frame script reference `me`.

Example

This statement displays the mouse location when the mouse is over a sprite:

```
on mouseWithin
  member("Display").text = string(the mouseH)
end mouseWithin
```

See also

on mouseEnter, **on mouseLeave**

mouseWord

Syntax `the mouseWord`

Description System property; contains the number of the word under the pointer when the property is called and when the pointer is over a field sprite. Counting starts from the beginning of the field. When the mouse is not over a field, the result is -1.

The value of the `mouseWord` property can change in a handler or repeat loop. If a handler or repeat loop uses this property multiple times, it's usually a good idea to call the function once and assign its value to a local variable.

Example This statement determines whether the pointer is over a field sprite and changes the content of the field cast member Instructions to "Please point to a word." when it is not:

```
if the mouseWord = -1 then
  member("Instructions").text = "Please point to a word."
else
  member("Instructions").text = "Thank you."
end if
```

Example This statement assigns the number of the word under the pointer in the specified field to the variable `currentWord`:

```
currentWord = member(the mouseMember).word[the mouseWord]
```

Example This handler highlights the word under the pointer when the mouse button is pressed:

```
on mouseDown
  thisField = sprite(the clickOn).member
  if the mouseWord < 1 then exit
  lastWord = 0
  repeat while the stillDown
    MW = the mouseWord
    if MW < 1 then next repeat
    if MW <> lastWord then
      thisField.word[MW].hilite()
            lastWord = MW
    end if
  end repeat
end
```

See also **mouseChar**, **mouseItem**, **mouseLine**, **number (words)**, **word...of**

moveableSprite

Syntax
```
sprite(whichSprite).moveableSprite
the moveableSprite of sprite whichSprite
```

Description Sprite property; indicates whether a sprite can be moved by the user (`TRUE`) or not (`FALSE`).

You can make a sprite moveable by using the Moveable option in the Score. However, to control whether a sprite is moveable and to turn this condition on and off as needed, use Lingo. For example, to let users drag sprites one at a time and then make the sprites unmoveable after they are positioned, turn the `moveableSprite` sprite property on and off at the appropriate times.

Note: For more customized control such as snapping back to the origin or animating while dragging, create a behavior to manage the additional functionality.

This property can be tested and set.

Example This handler makes sprites in channel 5 moveable:

```
on spriteMove
   sprite(5).moveableSprite = TRUE
end
```

Example This statement checks whether a sprite is moveable and if it isn't, displays a message:

```
if sprite(13).moveableSprite = FALSE then
     member("Notice").text = "You can't drag this item by using
the mouse."
```

See also **mouseLoc**

move member

Syntax

```
member(whichCastMember).move()
member(whichCastMember).move(member whichLocation)
move member whichCastMember {,member whichLocation}
```

Description

Command; moves the cast member specified by *whichCastMember* to the first empty location in the Cast window (if no parameter is set) or to the location specified by *whichLocation*.

For best results, use this command during authoring, not at run time, because the move is usually saved with the file. The actual location of a cast member does not affect most presentations during playback for an end user. To switch the content of a sprite or change the display during run time, set the member of the sprite.

Example

This statement moves cast member Shrine to the first empty location in the Cast window:

```
member("shrine").move()
```

Example

This statement moves cast member Shrine to location 20 in the Bitmaps Cast window:

```
member("Shrine").move(20, "Bitmaps")
```

moveToBack

Syntax
```
window("whichWindow ").MoveToBack()
moveToBack window whichWindow
```

Description Command; moves the window specified by *whichWindow* behind all other windows.

Example These statements move the first window in `windowList` behind all other windows:

```
myWindow = the windowList[1]
moveToBack myWindow
```

Example If you know the name of the window you want to move, use the syntax:

```
window("Demo Window").moveToBack()
```

moveToFront

Syntax
```
window("whichWindow ").MoveToFront()
moveToFront window whichWindow
```

Description Command; moves the window specified by *whichWindow* in front of all other windows.

Example These statements move the first window in `windowList` in front of all other windows:

```
myWindow = the windowList[1]
moveToFront myWindow
```

Example If the you know the name of the window you want to move to the front, use the syntax:

```
window("Demo Window").moveToFront()
```

moveVertex()

Syntax `member(memberRef). MoveVertex(vertexIndex, xChange, yChange)`

moveVertex(member memberRef, vertexIndex, xChange, yChange)

Description Function; moves the vertex of a vector shape cast member to another location.

The horizontal and vertical coordinates for the move are relative to the current position of the vertex point. The location of the vertex point is relative to the origin of the vector shape member.

Changing the location of a vertex affects the shape in the same way as dragging the vertex in an editor.

Example This statement shifts the first vertex point in the vector shape Archie 25 pixels to the right and 10 pixels down from its current position:

```
member("Archie").moveVertex(1, 25, 10)
```

See also **addVertex**, **deleteVertex()**,**moveVertexHandle()**,**originMode**, **vertexList**

moveVertexHandle()

Syntax `moveVertexHandle(member memberRef, vertexIndex, handleIndex, xChange, yChange)`

Description Function; moves the vertex handle of a vector shape cast member to another location.

The horizontal and vertical coordinates for the move are relative to the current position of the vertex handle. The location of the vertex handle is relative to the vertex point it controls.

Changing the location of a control handle affects the shape in the same way as dragging the vertex in the editor.

Example This statement shifts the first control handle of the second vertex point in the vector shape Archie 15 pixels to the right and 5 pixels up:

```
MoveVertexHandle(member "Archie", 2, 1, 15, -5)
```

See also **addVertex**, **deleteVertex()**,**originMode**, **vertexList**

on moveWindow

Syntax

```
on moveWindow
  statement(s)
end
```

Description

System message and event handler; contains statements that run when a window is moved, such as by dragging a movie to a new location on the Stage, and is a good place to put Lingo that you want executed every time a movie's window changes location.

Example

This handler displays a message in the Message window when the window a movie is playing in moves:

```
on moveWindow
  put "Just moved window containing"&&the movieName
end
```

See also

activeWindow, **movieName**, **windowList**

movie

This property is obsolete. Use **movieName**.

movieAboutInfo

Syntax `the movieAboutInfo`

Description Movie property; a string entered during authoring in the Movie Properties dialog box. This property is provided to allow for enhancements in future versions of Shockwave.

This property can be set but not tested.

See also **movieCopyrightInfo**

movieCopyrightInfo

Syntax `the movieCopyrightInfo`

Description Movie property; enters a string during authoring in the Movie Properties dialog box. This property is provided to allow for enhancements in future versions of Shockwave.

This property can be tested but not set.

Example This statement displays the copyright information in a text cast member:

```
member("Display").text = "Copyright"&&the movieCopyrightInfo
```

See also **movieAboutInfo**

movieFileFreeSize

Syntax `the movieFileFreeSize`

Description Movie property; returns the number of unused bytes in the current movie caused by changes to the cast members and castLibs within a movie. The `Save and Compact` and `Save As` commands rewrite the file to delete this free space.

When the movie has no unused space, the `movieFileFreeSize` function returns 0.

Example This statement displays the number of unused bytes that are in the current movie:

```
put the movieFileFreeSize
```

movieFileSize

Syntax
```
the movieFileSize
```

Description
Movie property; returns the number of bytes in the current movie saved on disk. This is the same number returned when selecting File Properties in Windows or Get Info in the Macintosh Finder.

Example
This statement displays the number of bytes in the current movie:

```
put the movieFileSize
```

movieFileVersion

Syntax `the movieFileVersion`

Description Movie property; indicates the version of Director in which the movie was last saved. The result is a string.

Example This statement displays the version of Director that last saved the current movie:

```
put the movieFileVersion
-- "800"
```

movieImageCompression

Syntax `the movieImageCompression`

Description Movie property; indicates the type of compression that Director applies to internal (non-linked) bitmap members when saving a movie in Shockwave format. This property can be set only during authoring and has no affect until the movie is saved in Shockwave format. Its value can be either of these symbols:

Value	Meaning
#standard	Use Director's standard internal compression format
#jpeg	Use JPEG compression (see imageCompression)

You normally set this property in Director's Publish Settings dialog box.

You can choose to override this setting for specific bitmap cast members by setting their `imageCompression` and `imageQuality` cast member properties.

See also **imageCompression**, **imageQuality**, **movieImageQuality**

movieImageQuality

Syntax `the movieImageQuality`

Description Movie property; indicates the level of compression to use when the `movieImageCompression` property is set to `#jpeg`. The range of acceptable values is 0–100. Zero yields the lowest image quality and highest compression; 100 yields the highest image quality and lowest compression.

You can only set this property during authoring and it has no affect until the movie is saved in Shockwave format.

Individual members may override this movie property by using the member property `imageCompression`.

See also **imageCompression**, **imageQuality**, **movieImageCompression**

movieName

Syntax `the movieName`

Description Movie property; indicates the simple name of the current movie.

In the Director authoring environment, a new movie that has not been saved has an empty string as this property.

Example This statement displays the name of the current movie in the Message window:

```
put the movieName
```

See also **moviePath**

moviePath

Syntax `the moviePath`

Description Movie property; indicates the pathname of the folder in which the current movie is located.

For pathnames that work on both Windows and Macintosh computers, use the `@` pathname operator.

To see an example of `moviePath` used in a completed movie, see the Read and Write Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement displays the pathname for the folder containing the current movie:

```
put the moviePath
```

Example This statement plays the sound file Crash.aif stored in the Sounds subfolder of the current movie's folder. Note the path delimiter used, indicating a Windows environment:

```
sound playFile 1, the moviePath&"Sounds/crash.aif"
```

See also **@ (pathname)**, **movieName**

movieRate

Syntax `sprite(whichSprite).movieRate`

the movieRate of sprite *whichSprite*

Description Digital video sprite property; controls the rate at which a digital video in a specific channel plays. The movie rate is a value specifying the playback of the digital video. A value of 1 specifies normal forward play, -1 specifies reverse, and 0 specifies stop. Higher and lower values are possible. For example, a value of 0.5 causes the digital video to play slower than normal. However, frames may be dropped when the `movieRate` sprite property exceeds 1. The severity of frame dropping depends on factors such as the performance of the computer the movie is playing on and whether the digital video sprite is stretched.

This property can be tested and set.

To see an example of `movieRate` used in a completed movie, see the QT and Flash movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement sets the rate for a digital video in sprite channel 9 to normal playback speed:

```
sprite(9).movieRate = 1
```

This statement causes the digital video in sprite channel 9 to play in reverse:

```
sprite(9).movieRate = -1
```

See also **duration**, **movieTime**

movieTime

Syntax
```
sprite(whichSprite).movieTime
the movieTime of sprite whichSprite
```

Description
Digital video sprite property; determines the current time of a digital video movie playing in the channel specified by *whichSprite*. The movieTime value is measured in ticks.

This property can be tested and set.

To see an example of movieTime used in a completed movie, see the QT and Flash movie in the Learning\Lingo Examples folder inside the Director application folder.

Example
This statement displays the current time of the QuickTime movie in channel 9 in the Message window:

```
put sprite(9).movieTime
```

Example
This statement sets the current time of the QuickTime movie in channel 9 to the value in the variable Poster:

```
sprite(9).movieTime = Poster
```

See also
duration

movieXtraList

Syntax `the movieXtraList`

Description Movie property; displays a linear property list of all Xtras in the Movies/Xtras dialog box that have been added to the movie.

- `#name`—Specifies the file name of the Xtra on the current platform. It is possible to have a list without a #name entry, such as when the Xtra exists only on one platform.
- `#packagefiles`—Set only when the Xtra is marked for downloading. The value of this property is another list containing a property list for each file in the download package for the current platform. The properties in this subproperty list are `#name` and `#version`, which contain the same information as found in the XtraList.

Two possible properties can appear in `movieXtraList`:

Example This statement displays output from `movieXtraList` in the Message window:

```
put the movieXtraList
-- [[#name: "Mix Services"], [#name: "Sound Import Export"],
[#name: "SWA Streaming  \ PPC Xtra"], [#name: "TextXtra PPC"],
[#name: "Font Xtra PPC"],[#name: "Flash Asset  \ Options PPC"],
[#name: "Font Asset PPC"], [#name: "Flash Asset PPC",  \
  #packagefiles: [[#fileName: "Flash Asset PPC", #version:
"1.0.3"]]]]
```

See also **xtraList**

multiSound

Syntax `the multiSound`

Description System property; specifies whether the system supports more than one sound channel and requires a Windows computer to have a multichannel sound card (`TRUE`) or not (`FALSE`).

Example This statement plays the sound file Music in sound channel 2 if the computer supports more than one sound channel:

```
if the multiSound then sound playFile 2, "Music.wav"
```

name (cast property)

Syntax

```
castLib (whichCast).name
the name of castLib whichCast
```

Description

Cast member property; returns the name of the specified cast.

This property can be tested and set.

Example

This code iterates through all the castLibs in a movie and displays their names in the Message window:

```
totalCastLibs = the number of castLibs
  repeat with currentCastLib = 1 to totalCastLibs
    put "Castlib"&&currentCastLib&&"is
named"&&castLib(currentCastLib).name
  end repeat
```

See also

&& (concatenator)

name (cast member property)

Syntax
```
member(whichCastMember).name
the name of member whichCastMember
```

Description
Cast member property; determines the name of the specified cast member. The argument *whichCastMember* is a string when is used as the cast member name or an integer when used as the cast member number.

The name is a descriptive string assigned by the developer. Setting this property is equivalent to entering a name in the Cast Member Properties dialog box.

This property can be tested and set.

Example
This statement changes the name of the cast member named On to Off:

```
member("On").name = "Off"
```

Example
This statement sets the name of cast member 15 to Background Sound:

```
member(15).name = "Background Sound"
```

Example
This statement sets the variable `itsName` to the name of the cast member that follows the cast member whose number is equal to the variable `i`:

```
itsName = member(i + 1).name
```

See also
number (cast member property)

name (menu property)

Syntax
```
the name of menu(whichMenu)
the name of menu whichMenu
```

Description Menu property; returns a string containing the name of the specified menu number.

This property can be tested but not set. Use the `installMenu` command to set up a custom menu bar.

Note: Menus are not available in Shockwave.

Example This statement assigns the name of menu number 1 to the variable `firstMenu`:

```
firstMenu = menu(1).name
```

Example The following handler returns a list of menu names, one per line:

```
on menuList
  theList = []
  repeat with i = 1 to the number of menus
    theList[i] = the name of menu i
  end repeat
  return theList
end menuList
```

See also **number (menus)**, **name (menu item property)**

name (menu item property)

Syntax

```
the name of menuItem(whichItem) of menu(whichMenu)
the name of menuItem whichItem of menu whichMenu
```

Description

Menu property; determines the text that appears in the menu item specified by *whichItem* in the menu specified by *whichMenu*. The *whichItem* argument is either a menu item name or a menu item number; *whichMenu* is either a menu name or a menu number.

This property can be tested and set.

Note: Menus are not available in Shockwave.

Example

This statement sets the variable `itemName` to the name of the eighth item in the Edit menu:

```
set itemName = the name of menuItem(8) of menu("Edit")
```

Example

This statement causes a specific file name to follow the word *Open* in the File menu:

```
the name of menuItem("Open") of menu("fileMenu") = "Open" &&
fileName
```

See also

name (menu property), **number (menu items)**

name (window property)

Syntax
```
window (whichWindow).name
the name of window whichWindow
```

Description
Window property; determines the name of the specified window in `windowList`. (The `title` window property determines the title that appears in a window's title bar.)

This property can be tested and set.

Example
This statement changes the name of the window Yesterday to Today:

```
window("Yesterday").name = "Today"
```

name (system property)

Syntax

```
xtra (whichXtra).name
the name of xtra whichXtra
```

Description

System property; indicates the name of the specified Lingo Xtra. Xtras that provide support services or other functions not available to Lingo will not support this property.

This property can be tested but not set.

Example

This statement displays the name of the first Xtra in the Message window:

```
put xtra(1).name
```

name (timeout property)

Syntax `timeoutObject.name`

Description This timeout property is the name of the timeout object as defined when the object is created. The `new()` command is used to create timeout objects.

Example This timeout handler opens an alert with the name of the timeout that sent the event:

```
on handleTimeout timeoutObject
  alert "Timeout:" && timeoutObject.name
end
```

See also **forget window**, **new()**,**period**, **persistent**, **target**, **time (timeout object property)**, **timeout()**,**timeoutHandler**, **timeoutList**

NAN

Description Lingo return value; indicates that a specified Lingo expression is not a number.

This statement attempts to display the square root of -1, which is not a number, in the Message window:

```
put (-1).sqrt
-- NAN
```

See also **INF**

netAbort

Syntax `netAbort(`*`URL`*`)`

`netAbort(`*`netID`*`)`

Description Command; cancels a network operation without waiting for a result.

Using a network ID is the most efficient way to stop a network operation. The ID is returned when you use a network function such as `getNetText()` or `postNetText()`.

In some cases, when a network ID is not available, you can use a URL to stop the transmission of data for that URL. The URL must be identical to that used to begin the network operation. If the data transmission is complete, this command has no effect.

Example This statement passes a network ID to `netAbort` to cancel a particular network operation:

```
on mouseUp
  netAbort(myNetID)
end
```

See also **getNetText()**,**postNetText**

netDone()

Syntax `netDone()`

`netDone(netID)`

Description Function; indicates whether a background loading operation (such as `getNetText`, `preloadNetThing`, `gotoNetMovie`, `gotoNetPage`, or `putNetText`) is finished or was terminated by a browser error (`TRUE`, default) or is still in progress (`FALSE`).

- Use `netDone()` to test the last network operation.
- Use `netDone(netID)` to test the network operation identified by `netID`.

The `netDone` function returns 0 when a background loading operation is in progress.

Example This handler uses the `netDone` function to test whether the last network operation has finished. If the operation is finished, text returned by `netTextResult` is displayed in the field cast member Display Text.

```
on exitFrame
  if netDone() = 1 then
    member("Display Text").text = netTextResult()
  end if
end
```

Example This handler uses a specific network ID as an argument for `netDone` to check the status of a specific network operation:

```
on exitFrame
  -- stay on this frame until the net operation is
  -- completed
  global mynetID
  if netDone(mynetID) = FALSE then
    go to the frame
  end if
end
```

See also **getNetText()**,**netTextResult()**,**gotoNetMovie**, **preloadNetThing()**

netError()

Syntax `netError()`

`netError(`*`netID`*`)`

Description Function; determines whether an error has occurred in a network operation and, if so, returns an error number corresponding to an error message. If the operation was successful, this function returns a code indicating that everything is okay. If no background loading operation has started, or if the operation is in progress, this function returns an empty string.

- Use `netError()` to test the last network operation.
- Use `netError(`*`netID`*`)` to test the network operation specified by `netID`.

Several possible error codes may be returned:

OK	Everything is okay.
4	Bad MOA class. The required network or nonnetwork Xtras are improperly installed or not installed at all.
5	Bad MOA Interface. See 4.
6	Bad URL or Bad MOA class. The required network or nonnetwork Xtras are improperly installed or not installed at all.
20	Internal error. Returned by `netError()` in the Netscape browser if the browser detected a network or internal error.
4146	Connection could not be established with the remote host.
4149	Data supplied by the server was in an unexpected format.
4150	Unexpected early closing of connection.
4154	Operation could not be completed due to timeout.
4155	Not enough memory available to complete the transaction.
4156	Protocol reply to request indicates an error in the reply.
4157	Transaction failed to be authenticated.
4159	Invalid URL.
4164	Could not create a socket.
4165	Requested object could not be found (URL may be incorrect).
4166	Generic proxy failure.
4167	Transfer was intentionally interrupted by client.
4242	Download stopped by `netAbort(url)`.
4836	Download stopped for an unknown reason, possibly a network error, or the download was abandoned.

When a movie plays back as an applet, this function always returns a string. The string either has a length of 0 or consists of text that describes an error. The string's content comes from Java and can vary on different operating systems or browsers. The text may

not be translated into the local language.

Example This statement passes a network ID to `netError` to check the error status of a particular network operation:

```
on exitFrame
  global mynetID
  if netError(mynetID)<>"OK" then beep
end
```

netLastModDate()

Syntax `netLastModDate()`

Description Function; returns the date last modified from the HTTP header for the specified item. The string is in Universal Time (GMT) format: *Ddd, nn Mmm yyyy hh:mm:ss* GMT (for example, Thu, 30 Jan 1997 12:00:00 AM GMT). There are variations where days or months are spelled completely. The string is always in English.

The `netLastModDate` function can be called only after `netDone` and `netError` report that the operation is complete and successful. After the next operation starts, the Director movie or projector discards the results of the previous operation to conserve memory.

The actual date string is pulled directly from the HTTP header in the form provided by the server. However, this string is not always provided, and in that case `netLastModDate` returns `EMPTY`.

When a movie plays back as an applet, this function's date format may differ from the date format that Shockwave uses; the date is in the format that the Java function `Date.asString` returns. The format may also vary on systems that use different languages.

Example These statements check the date of a file downloaded from the Internet:

```
if netDone() then
  theDate = netLastModDate()
  if theDate.char[6..11] <> "Jan 30" then
    alert "The file is outdated."
  end if
end if
```

See also **netDone()**,**netError()**

netMIME()

Syntax `netMIME()`

Description Function; provides the MIME type of the Internet file that the last network operation returned (the most recently downloaded HTTP or FTP item).

The `netMIME` function can be called only after `netDone` and `netError` report that the operation is complete and successful. After the next operation starts, the Director movie or projector discards the results of the previous operation to conserve memory.

Example This handler checks the MIME type of an item downloaded from the Internet and responds accordingly:

```
on checkNetOperation theURL
  if netDone (theURL) then
    set myMimeType = netMIME()
    case myMimeType of
      "image/jpeg": go frame "jpeg info"
      "image/gif": go frame "gif info"
      "application/x-director": goToNetMovie theURL
      "text/html": goToNetPage theURL
      otherwise: alert "Please choose a different item."
    end case
  else
    go the frame
  end if
end
```

See also **netDone()**,**netError()**,**getNetText()**,**postNetText**, **preloadNetThing()**

netPresent

Syntax

```
netPresent()

the netPresent
```

Description

System property; determines whether the Xtras needed to access the Internet are available but does not report whether an Internet connection is currently active.

If the Net Support Xtras are not available, `netPresent` will function properly, but `netPresent()` will cause a script error

Example

This statement sends an alert if the Xtras are not available:

```
if not (the netPresent) then
  alert "Sorry, the Network Support Xtras could not be found."
end if
```

netStatus

Syntax `netStatus` *`msgString`*

Description Command; displays the specified string in the status area of the browser window.

The `netStatus` command doesn't work in projectors.

Example This statement would place the string "This is a test" in the status area of the browser the movie is running in:

```
on exitFrame
  netStatus "This is a test"
end
```

netTextResult()

Syntax `netTextResult(netID)`

`netTextResult()`

Description Function; returns the text obtained by the specified network operation. If no net ID is specified, `netTextResult` returns the result of the last network operation.

If the specified network operation was `getNetText()`, the text is the text of the file on the network.

If the specified network operation was `postNetText`, the result is the server's response.

After the next operation starts, Director discards the results of the previous operation to conserve memory.

When a movie plays back as an applet, this function returns valid results for the last 10 requests. When a movie plays back as a Shockwave movie, this function returns valid results for only the most recent `getNetText()` operation.

Example This handler uses the `"netDone` and `netDone"` functions to test whether the last network operation finished successfully. If the operation is finished, text returned by `netTextResult` is displayed in the field cast member Display Text.

```
global gNetID
on exitFrame
  if (netDone(gNetID) = TRUE) and (netError(gNetID) = "OK") then
    member("Display Text").text = netTextResult()
    end if
end
```

See also **netDone(),netError(),postNetText**

netThrottleTicks

Syntax `the netThrottleTicks`

Description System property; in the Macintosh authoring environment, allows you to control the frequency of servicing to a network operation.

The default value is 15. The higher the value is set, the smoother the movie playback and animation is, but less time is spent servicing any network activity. A low setting allows more time to be spent on network operations, but will adversely affect playback and animation performance.

This property only affects the authoring environment and projectors on the Macintosh. It is ignored on Windows or Shockwave on the Mac.

new()

Syntax

```
new(type)
new(type, castLib whichCast)
new(type, member whichCastMember of castLib whichCast)
variableName = new(parentScript arg1, arg2, ...)
new(script parentScriptName, value1, value2, ...)
timout("name").new(timoutPeriod, #timeoutHandler, {, targetObject})
new(xtra "xtraName")
```

Description

Function; creates a new cast member, child object, timeout object, or Xtra instance and allows you to assign of individual property values to child objects.

The Director player for Java supports this function only for the creation of child objects. When a movie plays back as an applet, you can't use the `new` function to create cast members.

For cast members, the *`type`* parameter sets the cast member's type. Possible predefined values correspond to the existing cast member types: `#bitmap`, `#field`, and so on. The `new` function can also create Xtra cast member types, which can be identified by any name that the author chooses.

It's also possible to create a new color cursor cast member using the Custom Cursor Xtra. Use `new(#cursor)` and set the properties of the resulting cast member to make them available for use.

The optional *`whichCastMember`* and *`whichCast`* parameters specify the cast member slot and Cast window where the new cast member is stored. When no cast member slot is specified, the first empty slot is used. The `new` function returns the cast member slot.

When the argument for the `new` function is a parent script, the `new` function creates a child object. The parent script should include an `on new` handler that sets the child object's initial state or property values and returns the `me` reference to the child object.

The child object has all the handlers of the parent script. The child object also has the same property variable names that are declared in the parent script, but each child object has its own values for these properties.

Because a child object is a value, it can be assigned to variables, placed in lists, and passed as a parameter.

As with other variables, you can use the `put` command to display information about a child object in the Message window.

When `new()` is used to create a timeout object, the timoutPeriod sets the number of milliseconds between timeout events sent by the timeout object. The *`#timoutHandler`* is a symbol that identifies the handler that will be called when each timeout event occurrs. The *`targetObject`* identifies the name of the child object that contains the *`#timeoutHandler`*. If no *`targetObject`* is given, the *`#timeoutHandler`* is assumed to be in a movie script.

When a timeout object is created, it enables its *`targetObject`* to receive the system events `prepareMovie`, `startMovie`, `stopMovie`, `prepareFrame`, and `exitFrame`. To take advantage of this, the *`targetObject`* must contain handlers for these events. The events do not need to be passed in order for the rest of the movie to have access to them.

To see an example of `new()` used in a completed movie, see the Parent Scripts, and Read and Write Text movies in the Learning\Lingo Examples folder inside the Director application folder.

Example To create a new bitmap cast member in the first available slot, you use this syntax:

```
set newMember = new(#bitmap)
```

After the line has been executed, `newMember` will contain the member reference to the cast member just created:

```
put newMember
-- (member 1 of castLib 1)
```

Example This startMovie script creates a new Flash cast member using the `new` command, sets the newly created cast member's `linked` property so that the cast member's assets are stored in an external file, and then sets the cast member's `pathName` property to the location of a Flash movie on the World Wide Web:

```
on startMovie
  flashCastMember = new(#flash)
  member(flashCastMember).pathName =
"http://www.someURL.com/myFlash.swf"
end
```

Example When the movie starts, this handler creates a new animated color cursor cast member and stores its cast member number in a variable called `customCursor`. This variable is used to set the `castMemberList` property of the newly created cursor and to switch to the new cursor.

```
on startmovie
  customCursor = new(#cursor)
  member(customCursor).castMemberList = [member 1, member 2,
member 3]
  cursor (member(customCursor))
end
```

Example These statements from a parent script include the `on new` handler to create a child object. The parent script is a script cast member named Bird, which contains these handlers.

```
on new me, nameForBird
  return me
end
on fly me
  put "I am flying"
end
```

Example The first statement in the example creates a child object the above script in the preceding example, and places it in a variable named `myBird`. The second statement makes the bird fly by calling the fly handler in the Bird parent script:

```
myBird = script("Bird").new()
myBird.fly()
```

Example This statement uses a new Bird parent script, which contains the property variable `speed`:

```
property speed
on new me, initSpeed
  speed = initSpeed
  return me
end
```

```
on fly me
  put "I am flying at " & speed & "mph"
end
```

Example These statements create two child objects called myBird1 and myBird2. They are given different starting speeds: 15 and 25, respectively. When the fly handler is called for each child object, the speed of the object is displayed in the Message window.

```
myBird1 = script("Bird").new(15)
myBird2 = script("Bird").new(25)
myBird1.fly()
myBird2.fly()
```

This message appears in the Message window:

```
-- "I am flying at 15 mph"
-- "I am flying at 25 mph"
```

Example This statement creates a new timout object called intervalTimer that will send a timeout event to the `on minuteBeep` handler in the child object playerOne every 60 seconds:

```
timeout("intervalTimer").new(60000, #minuteBeep, playerOne)
```

See also **on stepFrame**, **actorList**, **ancestor**, **me**, **type (cast member property)**, **timeout()**

newCurve()

Syntax

```
vectorMember.newCurve(positionInVertexList)
newCurve(vectorMember, positionInVertexList)
```

Description

Function; adds a `#newCurve` symbol to the `vertexList` of *`vectorCastMember`*, which adds a new shape to the vector shape. The `#newCurve` symbol is added at *`positionInVertexList`*. You can break apart an existing shape by calling `newCurve()` with a position in the middle of a series of vertices.

Example

This Lingo adds a new curve to cast member 2 at the third position in the cast member's vertexList. The second line of the example replaces the contents of curve 2 with the contents of curve 3.

```
member(2).newCurve(3)
member(2).curve[2]=member(2).curve[3]
```

curve, **vertexList**

next

Syntax `next`

Description Keyword; refers to the next marker in the movie and is equivalent to the phrase `the marker (+ 1)`.

Example This statement sends the playback head to the next marker in the movie:

```
go next
```

Example This handler moves the movie to the next marker in the Score when the right arrow key is pressed and to the previous marker when the left arrow key is pressed:

```
on keyUp
  if the keyCode = 124 then go next
  if the keyCode = 123 then go previous
end keyUp
```

See also **loop (keyword)**, **go previous**

next repeat

Syntax `next repeat`

Description Keyword; sends Lingo to the next step in a repeat loop in a script. This function differs from that of the `exit repeat` keyword.

Example This repeat loop displays only odd numbers in the Message window:

```
repeat with i = 1 to 10
  if (i mod 2) = 0 then next repeat
  put i
end repeat
```

node

Syntax
```
sprite(whichQTVRSprite).node
the node of sprite whichQTVRSprite
```

Description
QuickTime VR sprite property; the current node ID displayed by the sprite.

This property can be tested and set.

nodeEnterCallback

Syntax

```
sprite(whichQTVRSprite).nodeEnterCallback
the nodeEnterCallback of sprite whichQTVRSprite
```

Description

QuickTime VR sprite property; contains the name of the handler that runs after the QuickTime VR movie switches to a new active node on the Stage.The message has two arguments: the `me` parameter and the ID of the node that is being displayed.

The QuickTime VR sprite receives the message first.

To clear the callback, set this property to 0.

To avoid a performance penalty, set a callback property only when necessary.

This property can be tested and set.

nodeExitCallback

Syntax

```
sprite(whichQTVRSprite).nodeExitCallback
the nodeExitCallback of sprite whichQTVRSprite
```

Description

QuickTime VR sprite property; contains the name of the handler that runs when the QuickTime VR movie is about to switch to a new active node on the Stage. The message has three arguments: the `me` parameter, the ID of the node that the movie is about to leave, and the ID of the node that the movie is about to switch to.

The value that the handler returns determines whether the movie goes on to the next node. If the handler returns `#continue`, the QuickTime VR sprite continues with a normal node transition. If the handler returns #cancel, the transition doesn't occur and the movie stays in the original node.

Set this property to 0 to clear the callback.

The QuickTime VR sprite receives the message first.

To avoid a performance penalty, set a callback property only when necessary.

This property can be tested and set.

nodeType

Syntax
```
sprite(whichQTVRSprite).nodeType
nodeType of sprite whichQTVRSprite
```

Description
QuickTime VR sprite property; gives the type of node that is currently on the Stage for the specified sprite. Possible values are `#object`, `#panorama`, or `#unknown`. (`#unknown` is the value for a sprite that isn't a QuickTime VR sprite.)

This property can be tested but not set.

not

Syntax `not logicalExpression`

Description Operator; performs a logical negation on a logical expression. This is the equivalent of making a `TRUE` value `FALSE`, and making a `FALSE` value `TRUE`. It is useful when testing to see if a certain known condition is not the case.

This logical operator has a precedence level of 5.

Example This statement determines whether 1 is not less than 2:

```
put not (1 < 2)
```

Because 1 is less than 2, the result is 0, which indicates that the expression is `FALSE`.

Example This statement determines whether 1 is not greater than 2:

```
put not (1 > 2)
```

Because 1 is not greater than 2, the result is 1, which indicates that the expression is `TRUE`.

Example This handler sets `the checkMark` menu item property for Bold in the Style menu to the opposite of its current setting:

```
on resetMenuItem
  the checkMark of menuItem("Bold") of menu("Style") = \
  not (the checkMark of menuItem("Bold") of menu("Style"))
end resetMenuItem
```

See also **and**, **or**

nothing

Syntax `nothing`

Description Command; does nothing. This command is useful for making the logic of an `if...then` statement more obvious. A nested `if...then...else` statement that contains no explicit command for the `else` clause may require `else nothing`, so that Lingo does not interpret the `else` clause as part of the preceding `if` clause.

Example The nested `if...then...else` statement in this handler uses the `nothing` command to satisfy the statement's `else` clause:

```
on mouseDown
  if the clickOn = 1 then
    if sprite(1).moveableSprite = TRUE then
  member("Notice").text = "Drag the ball"
    else nothing
  else member("Notice").text = "Click again"
  end if
end
```

Example This handler instructs the movie to do nothing so long as the mouse button is being pressed:

```
on mouseDown
  repeat while the stillDown
    nothing
  end repeat
end mouseDown
```

See also **if**

nudge

Syntax

```
sprite(whichQTVRSprite).nudge(#direction )
nudge(sprite whichQTVRSprite, #direction)
```

Description

QuickTime VR command; nudges the view perspective of the specified QuickTime VR sprite in the direction specified by *#direction*. Possible values for `#direction` are `#down`, `#downLeft`, `#downRight`, `#left`, `#right`, `#up`, `#upLeft`, and `#upRight`. Nudging to the right causes the image of the sprite to move to the left.

The `nudge` command has no return value.

Example

This handler causes the perspective of the QTVR sprite to move to the left as long as the mouse is held down on the sprite.

```
on mouseDown me
  repeat while the stillDown
    sprite(1).nudge(#left)
  end repeat
end
```

number (cast property)

Syntax `the number of castLib whichCast`

Description Cast property; indicates the number of the specified cast. For example, 2 is the castLib number for Cast 2.

This property can be tested but not set.

Example This repeat loop uses the Message window to display the number of cast members that are in each of the movie's casts:

```
repeat with n = 1 to the number of castLibs
  put castLib(n).name && "contains" && the number of \
  members of castLib(n) && "cast members."
end repeat
```

number (cast member property)

Syntax

```
member(whichCastMember).number
the number of member whichCastMember
```

Description

Cast member property; indicates the cast number of the cast member specified by *whichCastMember*: either a name, if *whichCastMember* is a string, or a number, if *whichCastMember* is an integer.

The property is a unique identifier for the cast member that is a single integer describing its location in and position in the castLib.

This property can be tested but not set.

Note: When using the first syntax of member(*whichCastMember*).number, an error is generated if the cast member does not exist. When unsure of the existence of the member, use the alternate syntax to avoid the error.

Example

This statement assigns the cast number of the cast member Power Switch to the variable *whichCastMember*:

```
whichCastMember = member("Power Switch").number
```

Example

This statement assigns the cast member Red Balloon to sprite 1:

```
sprite(1).member = member("Red Balloon").number
```

Example

This verifies that a cast member actually exists before trying to switch the cast member in the sprite:

```
property spriteNum
on mouseUp me
  if (member("Mike's face").number > 0) then
    sprite(spriteNum).member = "Mike's face"
  end if
end
```

See also

member (sprite property), **memberNum**, **number of members**

number (characters)

Syntax `the number of chars in chunkExpression`

Description Chunk expression; returns a count of the characters in a chunk expression.

Chunk expressions are any character (including spaces and control characters such as tabs and carriage returns), word, item, or line in any container of characters. Containers include field cast members and variables that hold strings, and specified characters, words, items, lines, and ranges in containers.

Note: The `count()` function provides a more efficient alternative for determining the number of characters in a chunk expression.

Example This statement displays the number of characters in the string “Macromedia, the Multimedia Company” in the Message window:

```
put the number of chars in "Macromedia, the Multimedia Company"
```

The result is 34.

Example This statement sets the variable `charCounter` to the number of characters in the word i located in the string Names:

```
charCounter = the number of chars in member("Names").word[i]
```

You can accomplish the same thing with text cast members using the syntax:

```
charCounter = member("Names").word[i].char.count
```

See also **length(),char...of**, **count(),number (items)**, **number (lines)**, **number (words)**

number (items)

Syntax `the number of items in chunkExpression`

Description Chunk expression; returns a count of the items in a chunk expression. An item chunk is any sequence of characters delimited by commas.

Chunk expressions are any character, word, item, or line in any container of characters. Containers include fields (field cast members) and variables that hold strings, and specified characters, words, items, lines, and ranges in containers.

Note: The `count()` function provides a more efficient alternative for determining the number of items in a chunk expression.

Example This statement displays the number of items in the string "Macromedia, the Multimedia Company" in the Message window:

```
put the number of items in "Macromedia, the Multimedia Company"
```

The result is 2.

Example This statement sets the variable `itemCounter` to the number of items in the field Names:

```
itemCounter = the number of items in member("Names").text
```

You can accomplish the same thing with text cast members using the syntax:

```
itemCounter = member("Names").item.count
```

See also **item...of**, **count()**,**number (characters)**, **number (lines)**, **number (words)**

number (lines)

Syntax `the number of lines in chunkExpression`

Description Chunk expression; returns a count of the lines in a chunk expression. (Lines refers to lines delimited by carriage returns, not lines formed by line wrapping.)

Chunk expressions are any character, word, item, or line in any container of characters. Containers include field cast members and variables that hold strings, and specified characters, words, items, lines, and ranges in containers.

Note: The `count()` function provides a more efficient alternative for determining the number of lines in a chunk expression.

Example This statement displays the number of lines in the string “Macromedia, the Multimedia Company” in the Message window:

```
put the number of lines in "Macromedia, the Multimedia Company"
```

The result is 1.

Example This statement sets the variable `lineCounter` to the number of lines in the field Names:

```
lineCounter = the number of lines in member("Names").text
```

You can accomplish the same thing with text cast members with the syntax:

```
lineCounter = member("Names").line.count
```

See also **line...of**, **count()**,**number (characters)**, **number (items)**, **number (words)**

number (menus)

Syntax `the number of menus`

Description Menu property; indicates the number of menus installed in the current movie.

This menu property can be tested but not set. Use the `installMenu` command to set up a custom menu bar.

Note: Menus are not available in Shockwave

Example This statement determines whether any custom menus are installed in the movie and, if no menus are already installed, installs the menu Menubar:

```
if the number of menus = 0 then installMenu "Menubar"
```

Example This statement displays in the Message window the number of menus that are in the current movie:

```
put the number of menus
```

See also **installMenu**, **number (menu items)**

number (menu items)

Syntax `the number of menuItems of menu whichMenu`

Description Menu property; indicates the number of menu items in the custom menu specified by *whichMenu*. The *whichMenu* parameter can be a menu name or menu number.

This menu property can be tested but not set. Use the `installMenu` command to set up a custom menu bar.

Note: Menus are not available in Shockwave

Example This statement sets the variable `fileItems` to the number of menu items in the custom File menu:

```
fileItems = the number of menuItems of menu "File"
```

Example This statement sets the variable `itemCount` to the number of menu items in the custom menu whose menu number is equal to the variable `i`:

```
itemCount = the number of menuItems of menu i
```

See also **installMenu**, **number (menus)**

number (system property)

Syntax

```
the number of castLibs
```

Description

System property; returns the number of casts that are in the current movie.

This property can be tested but not set.

Example

This repeat loop uses the Message window to display the number of cast members that are in each of the movie's casts:

```
repeat with n = 1 to the number of castLibs
  put castLib(n).name && "contains" && the number of \
  members of castLib(n) && "cast members."
end repeat
```

number (words)

Syntax `the number of words in chunkExpression`

Description Chunk expression; returns the number of words in the chunk expression specified by *chunkExpression*.

Chunk expressions are any character, word, item, or line in any container of characters. Containers include field cast members and variables that hold strings, and specified characters, words, items, lines, and ranges in containers.

To accomplish this functionality with text cast members, see `count`.

Note: The `count()` function provides a more efficient alternative for determining the number of words in a chunk expression.

Example This statement displays in the Message window the number of words in the string "Macromedia, the multimedia company":

```
put the number of words in "Macromedia, the multimedia company"
```

The result is 4.

Example This handler reverses the order of words in the string specified by the argument `wordList`:

```
on reverse wordList
  theList = EMPTY
  repeat with i = 1 to the number of words in wordList
    put word i of wordList & " " before theList
  end repeat
  delete theList.char[thelist.char.count]
  return theList
end
```

See also **count()**,**number (characters)**, **number (items)**, **number (lines)**, **word...of**

number of members

Syntax `the number of members of castLib whichCast`

Description Cast member property; indicates the number of the last cast member in the specified cast.

This property can be tested but not set.

Example This statement displays in the Message window the type of each cast member in the cast Central Casting. The `number of members of castLib` property is used to determine how many times the loop repeats.

```
repeat with i = 1 to the number of members of castLib("Central
Casting")
    put "Cast member" && i && "is a" && member(i, "Central
Casting").type
  end repeat
```

number of xtras

Syntax `the number of xtras`

Description System property; returns the number of scripting Xtras available to the movie. The Xtras may be either those opened by the `openxlib` command or those present in the standard Xtras folder.

This property can be tested but not set.

Example This statement displays in the Message window the number of scripting Xtras that are available to the movie:

```
put the number of xtras
```

numChannels

Syntax

```
member(whichCastMember).numChannels
the numChannels of member whichCastMember
```

Description

Shockwave Audio (SWA) cast member property; returns the number of channels within the specified SWA streaming cast member. The value can be either 1 for monaural or 2 for stereo.

This property is available only after the SWA streaming cast member begins playing or after the file has been preloaded using the `preLoadBuffer` command.

This property can be tested but not set.

Example

This example assigns the number of sound channels of the SWA streaming cast member Duke Ellington to the field cast member Channel Display:

```
myVariable = member("Duke Ellington").numChannels
if myVariable = 1 then
  member("Channel Display").text = "Mono"
else
  member("Channel Display").text = "Stereo"
end if
```

numToChar()

Syntax `numToChar(integerExpression)`

Description Function; displays a string containing the single character whose ASCII number is the value of *integerExpression*. This function is useful for interpreting data from outside sources that are presented as numbers rather than as characters.

ASCII values up to 127 are standard on all computers. Values of 128 or greater refer to different characters on different computers.

Example This statement displays in the Message window the character whose ASCII number is 65:

```
put numToChar(65)
```

The result is the letter *A*.

Example This handler removes any nonalphabetic characters from any arbitrary string and returns only capital letters:

```
on ForceUppercase input
  output = EMPTY
  num = length(input)
  repeat with i = 1 to num
    theASCII = charToNum(input.char[i])
    if theASCII = min(max(96, theASCII), 123) then
      theASCII = theASCII - 32
      if theASCII = min(max(63, theASCII), 91) then
           put numToChar(theASCII) after output
      end if
    end if
  end repeat
  return output
end
```

See also **charToNum()**

obeyScoreRotation

Syntax `member(flashMember).obeyScoreRotation`

Description Flash cast member property; set to TRUE or FALSE to determine if a Flash movie sprite uses the rotation information from the Score, or the older rotation property of Flash assets.

This property is automatically set to FALSE for all movies created in Director prior to version 7 in order to preserve old functionality of using the member rotation property for all sprites containing that Flash member.

New assets created in version 7 or later will have this property automatically set to TRUE.

If set to TRUE, the rotation property of the member is ignored and the Score rotation settings are obeyed instead.

Example The following sprite script sets the obeyScoreRotation property of cast member "dalmation" to 1 (TRUE), then rotates the sprite which contains the cast member 180 degrees.

```
on mouseUp me
  member("dalmation").obeyScoreRotation = 1
  sprite(1).rotation = sprite(1).rotation + 180
end
```

This property can be tested and set.

See also **rotation**

objectP()

Syntax `objectP(expression)`

Description Function; indicates whether the expression specified by *`expression`* is an object produced by a parent script, Xtra, or window (`TRUE`) or not (`FALSE`).

The *P* in `objectP` stands for *predicate*.

It is good practice to use `objectP` to determine which items are already in use when you create objects by parent scripts or Xtra instances.

To see an example of `objectP()` used in a completed movie, see the Read and Write Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement checks whether the global variable `gDataBase` has an object assigned to it and, if not, assigns one. This check is commonly used when you perform initializations at the beginning of a movie or section that you don't want to repeat.

```
if objectP(gDataBase) then
  nothing
else
  gDataBase = script("Database Controller").new()
end if
```

See also **floatP()**,**ilk()**,**integerP()**,**stringP()**,**symbolP()**

of

The word `of` is part of many Lingo properties, such as `foreColor`, `number`, `name`, and so on.

offset() (string function)

Syntax `offset(stringExpression1, stringExpression2)`

Description Function; returns an integer indicating the position of the first character of *`stringExpression1`* in *`stringExpression2`*. This function returns 0 if *`stringExpression1`* is not found in *`stringExpression2`*. Lingo counts spaces as characters in both strings.

On the Macintosh, the string comparison is not sensitive to case or diacritical marks. For example, Lingo considers *a* and *Å* to be the same character on the Macintosh.

Example This statement displays in the Message window the beginning position of the string "media" within the string "Macromedia":

```
put offset("media","Macromedia")
```

The result is 6.

Example This statement displays in the Message window the beginning position of the string "Micro" within the string "Macromedia":

```
put offset("Micro", "Macromedia")
```

The result is 0, because "Macromedia" doesn't contain the string "Micro".

Example This handler finds all instances of the string represented by `stringToFind` within the string represented by `input` and replaces them with the string represented by `stringToInsert`.

```
on SearchAndReplace input, stringToFind, stringToInsert
  output = ""
  findLen = stringToFind.length - 1
  repeat while input contains stringToFind
    currOffset = offset(stringToFind, input)
    output = output & input.char [1..currOffset]
    delete the last char of output
    output = output & stringToInsert
    delete input.char [1.. (currOffset + findLen)]
  end repeat
  set output = output & input
  return output
end
```

See also **chars()**,**length()**,**contains**, **starts**

offset() (rectangle function)

Syntax
```
rectangle.offset(horizontalChange, verticalChange)
offset (rectangle, horizontalChange, verticalChange)
```

Description Function; yields a rectangle that is offset from the rectangle specified by *rectangle*. The horizontal offset is the value specified by *horizontalChange*; the vertical offset is the value specified by *verticalChange*.

- When *horizontalChange* is greater than 0, the offset is toward the right of the Stage; when *horizontalChange* is less than 0, the offset is toward the left of the Stage.
- When *verticalChange* is greater than 0, the offset is toward the top of the Stage; when *verticalChange* is less than 0, the offset is toward the bottom of the Stage.

The values for *verticalChange* and *horizontalChange* are in pixels.

Example This handler moves sprite 1 five pixels to the right and five pixels down.

```
on diagonalMove
  newRect=sprite(1).rect.offset(5, 5)
  sprite(1).rect=newRect
end
```

on

Syntax

```
on handlerName {argument1}, {arg2}, {arg3} ...
  statement(s)
end handlerName
```

Description

Keyword; indicates the beginning of a handler, a collection of Lingo statements that you can execute using the handler name. A handler can accept arguments as input values and returns a value as a function result.

Handlers can be defined in behaviors, movie scripts, and cast member scripts. A handler in a cast member script can be called only by other handlers in the same script. A handler in a movie script can be called from anywhere.

You can use the same handler in more than one movie by putting the handler's script in a shared cast.

open

Syntax `open {whichDocument with} whichApplication`

Description Command; launches the application specified by the string *`whichApplication`*. Use *`whichDocument`* to specify a document that the application opens when it is launched. When either is in a different folder than the current movie, you must specify the full pathname to the file or files.

The computer must have enough memory to run both Director and other applications at the same time.

This is a very simple command for opening an application or a document within an application. For more control, look at options available in third-party Xtras.

Example This statement checks whether the computer is a Macintosh and then if it is, opens the application SimpleText:

```
if the platform contains "Mac" then open "SimpleText"
```

Example This statement opens the SimpleText application, which is in the folder Applications on the drive HD, and the document named Storyboards:

```
open "Storyboards" with "HD:Applications:SimpleText"
```

See also **openXlib**

openResFile

This is obsolete. Use **recordFont**.

open window

Syntax `window(whichWindow).open()`

`open window whichWindow`

Description Window command; opens the window object or movie file specified by *whichWindow* and brings it to the front of the Stage. If no movie is assigned to the window, the Open File dialog box appears.

- If you replace *whichWindow* with a movie's file name, the window uses the file name as the window.
- If you replace *whichWindow* with a window name, the window takes that name. However, you must then assign a movie to the window by using `set the fileName of window`.

To open a window that uses a movie from a URL, it's a good idea to use the `downloadNetThing` command to download the movie's file to a local disk first and then use the file on the disk. This minimizes problems with waiting for the movie to download.

For local media, the movie is not loaded into memory until the `open movie` command is executed. This can create a noticeable delay if you don't use `preloadMovie` to load at least the first frame of the movie prior to issuing the `open window` command.

Note: Opening a movie in a window is currently not supported in playback using a browser.

Example This statement opens the window Control Panel and brings it to the front:

```
window("Control Panel").open()"
```

See also **close window**, **downloadNetThing**, **preLoadMovie**

on openWindow

Syntax

```
on openWindow
  statement(s)
end
```

Description

System message and event handler; contains statements that run when Director opens a window and is a good place to put Lingo that you want executed every time the movie's window opens.

Example

This handler plays the sound file Hurray when the window that the movie is playing in opens:

```
on openWindow
  puppetSound 2, "Hurray"
end
```

openXlib

Syntax `openXlib *whichFile*`

Description Command; opens the Xlibrary file specified by the string expression `*whichFile*`. If the file is not in the folder containing the current movie, `*whichFile*` must include the pathname.

It is good practice to close any file you have opened as soon as you are finished using it. The `openXlib` command has no effect on an open file.

The `openXlib` command doesn't support URLs as file references.

Xlibrary files contain Xtras. Unlike `openResFile`, `openXlib` makes these Xtras known to Director.

In Windows, the .dll extension is optional.

Note: This command is not supported in Shockwave.

Example This statement opens the Xlibrary file Video Disc Xlibrary:

```
openXlib "Video Disc Xlibrary"
```

Example This statement opens the Xlibrary file Xtras, which is in a different folder than the current movie:

```
openXlib "My Drive:New Stuff:Transporter Xtras"
```

See also **closeXlib**, **interface()**,**showXlib**

optionDown

Syntax `the optionDown`

Description System property; determines whether the user is pressing the Alt key (Windows) or the Option key (Macintosh) (`TRUE`) or not (`FALSE`).

In Windows, `optionDown` doesn't work in projectors if Alt is pressed without another nonmodifier key. Avoid using `optionDown` if you intend to distribute a movie as a Windows projector and need to detect only the modifier key press; use `controlDown` or `shiftDown` instead.

On the Macintosh, pressing the Option key changes the `key` value, so use `keyCode` instead.

For a movie playing back with the Director player for Java, this function returns `TRUE` only if a second key is pressed at the same time as the Alt or Option key. If the Alt or Option key is pressed by itself, `optionDown` returns `FALSE`.

The Director player for Java supports key combinations with the Alt or Option key. However, the browser receives the keys before the movie plays and responds to and intercepts any key combinations that are also browser keyboard shortcuts.

Example This handler checks whether the user is pressing the Alt or the Option key and if so, calls the handler named `doOptionKey`:

```
on keyDown
  if (the optionDown) then doOptionKey(key)
end keyDown
```

See also **controlDown**, **commandDown**, **key()**,**keyCode()**,**shiftDown**

or

Syntax `logicalExpression1 or logicalExpression2`

Description Operator; performs a logical OR operation on two or more logical expressions to determine whether any expression is `TRUE`.

This is a logical operator with a precedence level of 4.

Example This statement indicates in the Message window whether at least one of the expressions 1 < 2 and 1 > 2 is `TRUE`:

```
put (1 < 2) or (1 > 2)
```

Because the first expression is `TRUE`, the result is 1, which is the numerical equivalent of `TRUE`.

Example This statement checks whether the content of the field cast member named State is either AK or HI and displays an alert if it is:

```
if member("State").text = "AK" or member("State").text = "HI" then
  alert "You're off the map!"
end if"
```

See also **and**, **not**

organizationName

Syntax `the organizationName`

Description Movie property; contains the company name entered during installation of Director.

This property is available in the authoring environment only. It can be used in a movie in a window tool that is personalized to show the user's information.

Example This handler would be located in a movie script of a movie in a window (MIAW). It places the user's name and serial number into a display field when the window is opened:

```
on prepareMovie
  displayString = the userName
  put RETURN & the organizationName after displayString
  put RETURN & the serialNumber after displayString
  member("User Info").text = displayString
end
```

See also **serialNumber**, **userName**, **window**

originalFont

Syntax `member(whichFontMember).originalFont`

the originalFont of member *whichFontMember*

Description Font cast member property; returns the exact name of the original font that was imported when the given cast member was created.

Example This statement displays the name of the font that was imported when cast member 11 was created:

```
put member(11).originalFont
-- "Monaco"
```

See also **recordFont**, **bitmapSizes**, **characterSet**

originH

Syntax

```
sprite(whichVectorOrFlashSprite).originH
the originH of sprite whichVectorOrFlashSprite
member(whichVectorOrFlashMember).originH
the originH of member whichVectorOrFlashMember
```

Description

Cast member and sprite property; controls the horizontal coordinate of a Flash movie or vector shape's origin point, in pixels. The value can be a floating- point value.

The origin point is the coordinate in a Flash movie or vector shape around which scaling and rotation occurs. The origin point can be set with floating- point precision using the separate `originH` and `originV` properties, or it can be set with integer precision using the single `originPoint` property.

You can set the `originH` property only if the `originMode` property is set to `#point`.

This property can be tested and set. The default value is 0.

Note: This property must be set to the default value if the scaleMode property is set to #autoSize, or the sprite does not display correctly.

Example

This sprite script uses the `originMode` property to set up a Flash movie sprite so it's origin point can be set to a specific point. It then sets the horizontal and vertical origin points.

```
on beginSprite me
  sprite(spriteNum of me).originMode = #point
  sprite(spriteNum of me).originH = 100
  sprite(spriteNum of me).originV = 80
end
```

See also

originV, **originMode**, **originPoint**, **scaleMode**

originMode

Syntax
```
sprite(whichFlashOrVectorShapeSprite).originMode
the originMode of sprite whichFlashOrVectorShapeSprite
member(whichFlashOrVectorShapeMember).originMode
the originMode of member whichFlashOrVectorShapeMember
```

Description Cast member property and sprite property; sets the origin point around which scaling and rotation occurs, as follows:

- `#center` (default)—The origin point is at the center of the Flash movie.
- `#topleft`—The origin point is at the top left of the Flash movie.
- `#point`—The origin point is at a point specified by the `originPoint`, `originH`, and `originV` properties.

This property can be tested and set.

Note: This property must be set to the default value if the scaleMode property is set to #autoSize, or the sprite will not display correctly.

Example This sprite script uses the `originMode` property to set up a Flash movie sprite so its origin point can be set to a specific point. It then sets the horizontal and vertical origin points.

```
on beginSprite me
  sprite(spriteNum of me).originMode = #point
  sprite(spriteNum of me).originH = 100
  sprite(spriteNum of me).originV = 80
end
```

See also **originH**, **originV**, **originPoint**, **scaleMode**

originPoint

Syntax

```
sprite whichVectorOrFlashSprite.originPoint
the originPoint of sprite whichVectorOrFlashSprite
member(whichVectorOrFlashMember).originPoint
the originPoint of member whichVectorOrFlashMember
```

Description

Cast member and sprite property; controls the origin point around which scaling and rotation occurs of a Flash movie or vector shape.

The `originPoint` property is specified as a Director point value: for example, point(100,200). Setting a Flash movie or vector shape's origin point with the `originPoint` property is the same as setting the `originH` and `originV` properties separately. For example, setting the `originPoint` property to point(50,75) is the same as setting the `originH` property to 50 and the `originV` property to 75.

Director point values specified for the `originPoint` property are restricted to integers, whereas `originH` and `originV` can be specified with floating- point numbers. When you test the `originPoint` property, the point values are truncated to integers. As a rule of thumb, use the `originH` and `originV` properties for precision; use the `originPoint` property for speed and convenience.

You can set the `originPoint` property only if the `originMode` property is set to `#point`.

This property can be tested and set. The default value is 0.

Note: This property must be set to the default value if the scaleMode property is set to #autoSize, or the sprite will not display correctly.

Example

This sprite script uses the `originMode` property to set up a Flash movie sprite so its origin point can be set to a specific point. It then sets the origin points.

```
on beginSprite me
  sprite(me.spriteNum).scaleMode = #showAll
  sprite(me.spriteNum).originMode = #point
  sprite(me.spriteNum).originPoint = point(100, 80)
end
```

See also

originH, **originV**, **scaleMode**

originV

Syntax

```
sprite(whichVectorOrFlashSprite).originV
the originV of sprite whichVectorOrFlashSprite
member(whichVectorOrFlashMember).originV
the originV of member whichVectorOrFlashMember
```

Description

Cast member and sprite property; controls the vertical coordinate of a Flash movie or vector shape's origin point around which scaling and rotation occurs, in pixels. The value can be a floating-point value.

The origin point can be set with floating-point precision using the separate `originH` and `originV` properties, or it can be set with integer precision using the single `originPoint` property.

You can set the `originV` property only if the `originMode` property is set to `#point`.

This property can be tested and set. The default value is 0.

Note: This property must be set to the default value if the scaleMode property is set to #autoSize, or the sprite does not display correctly.

Example

This sprite script uses the `originMode` property to set up a Flash movie sprite so its origin point can be set to a specific point. It then sets the horizontal and vertical origin points.

```
on beginSprite me
  sprite(me.spriteNum).scaleMode = #showAll
  sprite(me.spriteNum).originMode = #point
  sprite(me.spriteNum).originH = 100
  sprite(me.spriteNum).originV = 80
end
```

See also

originH, **originPoint**, **scaleMode**

otherwise

Syntax `otherwise statement(s)`

Description Keyword; precedes instructions that Lingo performs when none of the earlier conditions in a `case` statement are met.

This keyword can be used to alert users of out-of-bound input or invalid type, and can be very helpful in debugging during development.

Example The following handler tests which key the user pressed most recently and responds accordingly:

- If the user pressed A, B, or C, the movie performs the corresponding action following the `of` keyword.
- If the user pressed any other key, the movie executes the statement that follows the `otherwise` keyword. In this case, the statement is a simple alert.

```
on keyDown
  case (the key) of
    "A": go to frame "Apple"
    "B", "C":
      puppetTransition 99
      go to frame "Oranges"
    otherwise:
      alert "That is not a valid key."
  end case
end keyDown
```

pageHeight

Syntax

```
member(whichCastMember).pageHeight
the pageHeight of member whichCastMember
```

Description

Field cast member property; returns the height, in pixels, of the area of the field cast member that is visible on the Stage.

This property can be tested but not set.

Example

This statement returns the height of the visible portion of the field cast member Today's News:

```
put member("Today's News").pageHeight"
```

palette

Syntax

```
member(whichCastMember).palette
the palette of member whichCastMember
```

Description

Cast member property; for bitmap cast members only, determines which palette is associated with the cast member specified by *whichCastMember*.

This property can be tested and set.

Example

This statement displays the palette assigned to the cast member Leaves in the Message window:

```
put member("Leaves").palette"
```

paletteMapping

Syntax `the paletteMapping`

Description Movie property; determines whether the movie remaps (`TRUE`) or does not remap (`FALSE`, default) palettes for cast members whose palettes are different from the current movie palette. Its effect is similar to that of the Remap Palettes When Needed check box in the Movie Properties dialog box.

To display different bitmaps with different palettes simultaneously, set `paletteMapping` to `TRUE`. Director looks at each onscreen cast member's reference palette (the palette assigned in its Cast Member Properties dialog box) and, if it is different from the current palette, finds the closest match for each pixel in the new palette.

The colors of the nonmatching bitmap will be close to the original colors.

Remapping consumes processor time, and it's usually better to adjust the bitmap's palette in advance.

Remapping can also produce undesirable results. If the palette changes in the middle of a sprite span, the bitmap immediately remaps to the new palette and appears in the wrong colors. However, if anything refreshes the screen—a transition or a sprite moving across the Stage—then the affected rectangle on the screen appears in remapped colors.

Example This statement tells the movie to remap the movie's palette whenever necessary:

```
set the paletteMapping = TRUE
```

paletteRef

Syntax

```
member(whichCastMember). paletteRef
the paletteRef
```

Description

Bitmap cast member property; determines the palette associated with a bitmap cast member. Built-in Director palettes are indicated by symbols (`#systemMac`, `#rainbow`, and so on). Palettes that are cast members are treated as cast member references. This behavior differs from that of the `palette` member property, which returns a positive number for cast palettes and negative numbers for built-in Director palettes.

This property can be tested and set.

Example

This statement assigns the Macintosh system palette to the bitmap cast member Shell:

```
member("Shell").paletteRef = #systemMac
```

pan (QTVR property)

Syntax `pan of sprite whichQTVRSprite`

Description QuickTime VR sprite property; the current pan of the QuickTime VR movie. The value is in degrees.

This property can be tested and set.

pan (sound property)

Syntax

```
sound(channelNum).pan
the pan of sound(channelNum)
```

Description

Property; indicates the left/right balance of the sound playing in sound channel *channelNum*.The range of values is from –100 to 100. -100 indicates only the left channel is heard. 100 indicate only the right channel is being heard. A value of 0 indicates even left/right balance, causing the sound source to appear to be centered. For mono sounds, `pan` affects which speaker (left or right) the sound plays through.

You can change the pan of a sound object at any time, but if the sound channel is currently performing a fade, the new pan setting doesn't take effect until the fade is complete.

To see an example of `pan (sound property)` used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This Lingo statement pans the sound in sound channel 2 from the left channel to the right channel:

```
repeat with x = -100 to 100
  sound(2).pan = x
end repeat
```

See also

fadeIn(),**fadeOut()**,**fadeTo()**,**volume (sound channel)**

paragraph

Syntax `chunkExpression.paragraph[whichParagraph]`

chunkExpression.paragraph[firstParagraph..lastParagr aph]

Description Text cast member property; this chunk expression allows access to different paragraphs within a text cast member.

The paragraph is delimited by a carriage return.

```
put member("AnimText").paragraph[3]
```

See also **line...of**

param()

Syntax `param(parameterPosition)`

Description Function; provides the value of a parameter passed to a handler. The expression *parameterPosition* represents the parameter's position in the arguments.

To avoid errors in a handler, this function can be used to determine the type of a particular paramater.

Example This handler accepts any number of arguments, adds all the numbers passed in as parameters, and then returns the sum:

```
on AddNumbers
  sum = 0
  repeat with currentParamNum = 1 to the paramCount
    sum = sum + param(currentParamNum)
  end repeat
  return sum
end
```

You would use it by passing in the values you wanted to add:

```
put AddNumbers(3, 4, 5, 6)
-- 18
put AddNumbers(5, 5)
-- 10
```

See also **getAt**, **param()**,**paramCount()**,**return (keyword)**

paramCount()

Syntax `the paramCount`

Description Function; indicates the number of parameters sent to the current handler.

Example This statement sets the variable `counter` to the number of parameters that were sent to the current handler:

```
set counter = the paramCount
```

pass

Syntax `pass`

Description Command; passes an event message to the next location in the message hierarchy and enables execution of more than one handler for a given event.

The Director player for Java supports this command only within `on keyDown` and `on keyUp` handlers attached to editable sprites.

The `pass` command branches to the next location as soon as the command runs. Any Lingo that follows the `pass` command in the handler does not run.

By default, an event message stops at the first location containing a handler for the event, usually at the sprite level.

If you include the `pass` command in a handler, the event is passed to other objects in the hierarchy even though the handler would otherwise intercept the event.

Example This handler checks the keypresses being entered, and allows them to pass through to the editable text sprite if they are valid characters:

```
on keyDown me
  legalCharacters = "1234567890"
  if legalCharacters contains the key then
    pass
  else
    beep
  end if
end
```

See also **stopEvent**

pasteClipBoardInto

Syntax `member(whichCastMember). pasteClipBoardInto()`

`pasteClipBoardInto member whichCastMember`

Description Command; pastes the contents of the Clipboard into the cast member specified by *whichCastMember* and erases the exiting cast member. For example, pasting a bitmap into a field cast member makes the bitmap the cast member and erases the field cast member.

You can paste any item that is in a format that Director can use as a cast member. When you copy a string from another application, the string's formatting is not retained.

The `pasteClipBoardInto` command provides a convenient way to copy objects from other movies and from other applications into the Cast window. Because copied cast members must be stored in RAM, avoid using this command during playback in low memory situations.

Note: When you use this command in Shockwave, or in the authoring environment and projectors with the safePlayer property set to TRUE, a warning dialog will allow the user to cancel the paste operation.

Example This statement pastes the Clipboard contents into the bitmap cast member Shrine:

```
member("shrine").pasteClipBoardInto()
```

See also **safePlayer**

pathName (cast member property)

Syntax

```
member(whichFlashMember).pathName
the pathName of member whichFlashMember
```

Description

Cast member property; controls the location of an external file that stores the assets of a Flash movie cast member are stored. You can link a Flash movie to any path on a local or network drive or to a URL.

Setting the path of an unlinked cast member converts it to a linked cast member.

This property can be tested and set. The `pathName` property of an unlinked member is an empty string.

This property is the same as the `fileName` property for other member types, and you can use `fileName` instead of `pathName`.

Example

This `startMovie` script creates a new Flash cast member using the `new` command, sets the newly created cast member's `linked` property so that the cast member's assets are stored in an external file, and then sets the cast member's `pathName` property to the location of a Flash movie on the World Wide Web:

```
on startMovie
  member(new(#flash)).pathName = \
"http://www.someURL.com/myFlash.swf"
end
```

See also

fileName (cast member property), **linked**

pathName (movie property)

This is obsolete. Use **moviePath**.

pattern

Syntax

```
member(whichCastMember).pattern
the pattern of member whichCastMember
```

Description

Cast member property; determines the pattern associated with the specified shape. Possible values are the numbers that correspond to the swatches in the Tools window's patterns palette. If the shape cast member is unfilled, the pattern is applied to the cast member's outer edge.

The Director player for Java can assign only the patterns for chips 1 and 15 in Director's patterns palette.

This property can be useful in Shockwave movies to change images by changing the tiling applied to a shape, allowing you to save memory required by larger bitmaps.

This property can be tested and set.

Example

The following statements make the shape cast member Target Area a filled shape and assign it pattern 1, which is a solid color:

```
member("Target Area").filled = TRUE
member("Target Area").pattern = 1
```

Example

This handler cycles through eight tiles, with each tile's number offset from the previous one, enabling you to create animation using smaller bitmaps:

```
on exitFrame
  currentPat = member("Background Shape").pattern
  nextPat = 57 + ((currentPat - 56) mod 8)
  member("Background Shape").pattern = nextPat
  go the frame
end
```

pause (movie playback)

This is obsolete. Use `go to the frame`.

pause() (sound playback)

Syntax

```
sound(channelNum).pause()
pause(sound(channelNum))
```

Description

This command suspends playback of the current sound in sound channel *channelNum*. A subsequent `play()` command will resume playback.

Example

This statement pauses playback of the sound cast member playing in sound channel 1.

```
sound(1).pause()
```

See also

breakLoop(), **isBusy()**, **play() (sound)**, **playNext()**, **queue()**, **rewind()**, **status**, **stop() (sound)**

pausedAtStart

Syntax

```
member(whichFlashOrDigitalVideoMember).pausedAtStart
the pausedAtStart of member whichFlashOrDigitalVideoMember
```

Description

Cast member property; controls whether the digital video or Flash movie plays when it appears on the Stage. If this property is `TRUE`, the digital video or Flash movie does not play when it appears. If this property is `FALSE`, it plays immediately when it appears.

For a digital video cast member, the property specifies whether the Paused at Start check box in the Digital Video Cast Member Properties dialog box is selected or not.

This property can be tested and set.

Example

This statement turns on the Paused at Start check box in the Digital Video Cast Member Info dialog box for the QuickTime movie Rotating Chair:

```
member("Rotating Chair").pausedAtStart = TRUE
```

See also

play

pause member

Syntax `member(whichCastMember). pause()`

pause member ("*whichCastMember*")

Description Command; pauses the streaming of a Shockwave Audio (SWA) streaming cast member. When the sound is paused, the `state` member property equals 4. The portion of the sound that has already been downloaded and is available will continue to play until the cache runs out.

Example This handler could be used for a Play or Pause button. If the sound is playing, the handler pauses the sound; otherwise, the handler plays the sound linked to the SWA streaming cast member soundSWA.

```
on mouseDown
  whatState = member("soundSWA").state
  if whatState = 3 then
    member("soundSWA").pause()
  else
    member("soundSWA").play()
  end if
end
```

See also **play member**, **stop member**

pause sprite

Syntax `sprite(whichGIFSpriteNumber). pause()`

pause(sprite whichGIFSpriteNumber)

Description Command; causes an animated GIF sprite to pause in its playback and remain on the current frame.

Example sprite(1).pause()

See also **resume sprite**, **rewind sprite**

pauseState

Syntax `the pauseState`

Description Movie property; determines whether the pause command is currently pausing the movie (`TRUE`) or not (`FALSE`).

Becuase the `pause` command is obsolete, this property is not commonly used.

Example This statement checks whether the movie is currently paused and, if it is paused, causes the movie to continue playing:

```
if the pauseState = TRUE then go the frame + 1
```

See also **pause (movie playback)**

percentPlayed

Syntax

```
member(whichCastMember).percentPlayed
the percentPlayed of member whichCastMember
```

Description

Shockwave Audio (SWA) cast member property; returns the percentage of the specified SWA file that has actually played.

This property can be tested only after the SWA sound starts playing or has been preloaded by means of the `preLoadBuffer` command. This property cannot be set.

Example

This handler displays the percentage of the SWA streaming cast member Frank Sinatra that has played and puts the value in the field cast member Percent Played:

```
on exitFrame
  whatState = member("Frank Sinatra").state
  if whatState > 1 AND whatState < 9 then
    member("Percent Played").text = /
string(member("Frank Sinatra").percentPlayed)
  end if
end
```

See also

percentStreamed

percentStreamed

Syntax

```
member(whichCastMember).percentStreamed
the percentStreamed of member whichCastMember
```

Description

Shockwave Audio (SWA) cast member property; for SWA streaming sounds, gets the percent of an SWA file already streamed from an HTTP or FTP server; or for Flash movie cast memebers, gets the percent of a Flash movie that has streamed into memory. This property is a value from 0 to 100%.

For SWA, this property differs from the `percentPlayed` property in that it includes the amount of the file that has been buffered but not yet played.

This property can be tested only after the SWA sound starts playing or has been preloaded by means of the `preLoadBuffer` command. This property cannot be set.

Example

This example displays the percentage of the SWA streaming cast member Ray Charles that has streamed and puts the value in a field:

```
on exitFrame
  whatState = member("Ray Charles").state
  if whatState > 1 AND whatState < 9 then
    member("Percent Streamed Displayer").text = \
string(member("Ray Charles").percentStreamed)
  end if
end
```

Example

This frame script keeps the playback head looping in the current frame so long as less than 60 percent of a Flash movie called Splash Screen has streamed into memory:

```
on exitFrame
  if member("Splash Screen").percentStreamed < 60 then
    go to the frame
  end if
 end
```

See also

percentPlayed

period

Syntax `timeoutObject.period`

Description Object property; the number of milliseconds between timeout events sent by the `timeOutObject` to the timeout handler.

This property can be tested and set.

Example This timeout handler decreases the timeout's `period` by one second each time it's invoked, until a minimum period of 2 seconds (2000 milliseconds) is reached:

```
on handleTimeout timeoutObject
  if timeoutObject.period > 2000 then
       timeoutObject.period = timeoutObject.period - 1000
  end if
end handleTimeout
```

See also **name (timeout property)**, **persistent**, **target**, **time (timeout object property)**, **timeout()**,**timeoutHandler**, **timeoutList**

persistent

Syntax `timeoutObject.persistent`

Description Object property; determines whether the given *`timeoutObject`* is removed from `the timeoutList` when the current movie stops playing. If `TRUE`, *`timeoutObject`* remains active. If `FALSE`, the timeout object is deleted when the movie stops playing. The default value is `FALSE`.

Setting this property to `TRUE` allows a timeout object to continue generating timeout events in other movies. This is useful when one movie branches to another with the `go to movie` command.

Example This `prepareMovie` handler creates a timeout object that will remain active after the declaring movie stops playing:

```
on prepareMovie
  -- Make a timeout object that sends an event every 60 minutes.
  timeout("reminder").new(1000 * 60 * 60, #handleReminder)
  timeout("reminder").persistent = TRUE
end
```

See also **name (timeout property)**, **period**, **target**, **time (timeout object property)**, **timeout()**,**timeoutHandler**, **timeoutList**

PI

Syntax `PI`

Description Constant; returns the value of pi (π), the ratio of a circle's circumference to its diameter, as a floating-point number. The value is rounded to the number of decimal places set by the `floatPrecision` property.

Example This statement uses the `PI` constant as part of an equation for calculating the area of a circle:

```
set vArea = PI*power(vRadius,2)
```

picture (cast member property)

Syntax

```
member(whichCastMember).picture
the picture of member whichCastMember
```

Description

Cast member property; determines which image is associated with a bitmap, text, or PICT cast member. To update changes to a cast member's registration point or update changes to an image after relinking it using the `fileName` property, use the following statement:

```
member(whichCastMember).picture = member(whichCastMember).picture
```

where you replace *whichCastMember* with the name or number of the affected cast member.

Because changes to cast members are stored in RAM, this property is best used during authoring. Avoid setting it in projectors.

The property can be tested and set.

Example

This statement sets the variable named `pictHolder` to the image in the cast member named Sunset:

```
pictHolder = member("Sunset").picture
```

See also

type (sprite property)

picture (window property)

Syntax

```
the stage.picture
the picture of the stage
window whichWindow.picture
the picture of window whichWindow
```

Description

Window property; this property provides a way to get a picture of the current contents of a window (either the Stage window or a movie in a window).

You can apply the the resulting bitmap data to an existing bitmap or use it to create a new one.

This property can be read and but not set.

Example

This statement grabs the current content of the Stage and places it into a bitmap cast member:

```
member("Stage image").picture = (the stage).picture
```

See also

media, **picture (cast member property)**

pictureP()

Syntax `pictureP(pictureValue)`

Description Function; reports whether the state of the `picture` member property for the specified cast member is `TRUE` (1) or `FALSE` (0).

Because `pictureP` doesn't directly check whether a picture is associated with a cast member, you must test for a picture by checking the cast member's `picture` member property.

Example The first statement assigns the value of the `picture` member property for the cast member Shrine, which is a bitmap, to the variable `pictureValue`. The second statement checks whether Shrine is a picture by checking the value assigned to `pictureValue`.

```
set pictureValue to the picture of member "Shrine"
put pictureP(pictureValue)
```

The result is 1, which is the numerical equivalent of `TRUE`.

platform

Syntax `the platform`

Description System property; indicates the platform type for which the projector was created.

This property can be tested but not set.

Possible values are the following:

Possible value	Corresponding platform
Macintosh,PowerPC	PowerPC Macintosh
Windows,32	Windows 95 or Windows NT

For forward compatibility and to allow for addition of values, it is better to test the platform by using `contains`.

When the movie plays back as a converted Java applet, this property's value indicates the browser and operating system in which the applet is playing. The property's value has the following syntax when the movie plays back as an applet:

```
Java javaVersion, browser, operatingSystem
```

The following are the possible values for this property's parameters:

- *javaVersion*: 1.0 or 1.1
- *browser*: IE, Netscape, or UnknownBrowser
- *operatingSystem*: Macintosh, Windows, or UnknownOS

For example, if an applet is playing in Microsoft Internet Explorer with Java 1.1 in Windows, `platform` has the value `Java 1.1, IE, Windows`.

Example This statement checks whether a projector was created for Windows 95 or Windows NT:

```
on exitFrame
  if the platform contains "Windows,32" then
    castLib("Win95 Art").name = "Interface"
  end if
end
```

See also **runMode**

play

Syntax

```
sprite(whichFlashSprite).play()
play [frame] whichFrame
play movie whichMovie
play frame whichFrame of movie whichMovie
play sprite whichFlashSprite
```

Description

Command; branches the playback head to the specified frame of the specified movie or starts a Flash movie sprite playing. For the former, the expression *whichFrame* can be either a string marker label or an integer frame number. The expression *whichMovie* must be a string that specifies a movie file. When the movie is in another folder, *whichMovie* must specify a path.

The `play` command is like the `go to` command, except that when the current sequence finishes playing, `play` automatically returns the playback head to the frame where `play` was called.

If `play` is issued from a frame script, the playback head returns to the next frame; if `play` is issued from a sprite script or handler, the playback head returns to the same frame. A play sequence ends when the playback head reaches the end of the movie or when the `play done` command is issued.

To play a movie from a URL, use `downloadNetThing` or `preloadNetThing()` to download the file to a local disk first, and then use `play` to play the movie on the local disk to minimize download time.

You can use the `play` command to play several movies from a single handler. The handler is suspended while each movie plays but resumes when each movie is finished. Contrast this with a series of `go` commands that, when called from a handler, play the first frame of each movie. The handler is not suspended while the movie plays but immediately continues executing.

When `play` is used to play a Flash movie sprite, the Flash movie plays from its current frame if it is stopped or from its first frame if it is already on the last frame.

Each `play` command needs a matching `play done` command to avoid using up memory if the original calling script isn't returned to. To avoid this memory consumption, you can use a global variable to record where the movie should return to.

Example

This statement moves the playback head to the marker named blink:

```
play "blink"
```

Example

This statement moves the playback head to the next marker:

```
play marker(1)
```

Example

This statement moves the playback head to a separate movie:

```
play movie "My Drive:More Movies:" & newMovie
```

Example

This frame script checks to see if the Flash movie sprite in channel 5 is playing, and if it is not, it starts the movie:

```
on enterFrame
  if not sprite(5).playing then
    sprite(5).play()
  end if
end
```

See also **downloadNetThing**

play() (sound)

Syntax

```
sound(channelNum).play()
```

sound(*channelNum*).play(member (*whichMember*))

```
sound(channelNum).play([#member: member(whichmember), {#startTime:
milliseconds, #endTime: milliseconds, #loopCount: numberOfLoops,
#loopStartTime: milliseconds, #loopEndTime: milliseconds,
#preloadTime: milliseconds}])
```

Description

This function begins playing any sounds queued in `soundObject`, or queues and begins playing the given member.

Sound members take some time to load into RAM before they can begin playback. It's recommended that you queue sounds with `queue()` before you want to begin playing them and then use the first form of this function. The second two forms do not take advantage of the pre-loading accomplished with the `queue()` command.

By using an optional property list, you can specify exact playback settings for a sound. These properties may be optionally set:

Property	Description
#member	The sound cast member to queue. This property must be provided; all others are optional.
#startTime	The time within the sound at which playback begins, in milliseconds. The default is the beginning of the sound. See startTime.
#endTime	The time within the sound at which playback ends, in milliseconds. The default is the end of the sound. See endTime.
#loopCount	The number of times to play a loop defined with #loopStartTime and #loopEndTime. The default is 1. See loopCount.
#loopStartTime	The time within the sound to begin a loop , in milliseconds. See loopStartTime.
#loopEndTime	The time within the sound to end a loop, in milliseconds. See loopEndTime.
#preloadTime	The amount of the sound to buffer before playback, in milliseconds. See preloadTime.

To see an example of `play() (sound)` used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement plays cast member introMusic in sound channel 1:

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

Example

This statement plays cast member creditsMusic in sound channel 2. Playback begins 4 seconds into the sound and ends 15 seconds into the sound. The section from 10.5 seconds to 14 seconds loops 6 times.

```
sound(2).play([#member: member("creditsMusic"), #startTime: 4000,
\
#endTime: 15000, #loopCount: 6, #loopStartTime: 10500,
#loopEndTime: 14000])
```

See also **setPlaylist()**, **isBusy()**, **pause() (sound playback)**, **playNext()**, **preloadTime**, **queue()**, **rewind()**, **stop() (sound)**

playBackMode

Syntax

```
sprite(whichFlashSprite).playBackMode
the playBackMode of sprite whichFlashSprite
member(whichGIFAnimMember).playBackMode
the playBackMode of member whichGIFAnimMember
```

Description Cast member and sprite property; controls the tempo of a Flash movie or animated GIF cast member with the following values:

- #normal (*default*)—Plays the Flash movie or GIF file as close to the original tempo as possible.
- `#lockStep`—Plays the Flash movie or GIF file frame for frame with the Director movie.
- `#fixed`—Plays the Flash movie or GIF file at the rate specified by the `fixedRate` property.

This property can be tested and set.

Example This sprite script sets the frame rate of a Flash movie sprite to match the frame rate of the Director movie:

```
property spriteNum
on beginSprite me
  sprite(spriteNum).playBackMode = #lockStep
end
```

See also **fixedRate**

play done

Syntax `play done`

Description Command; ends the sequence started with the most recent `play` command. The `play done` command returns the playback head to where the calling sequence initiated the `play` command. If `play` is issued from a frame script, the playback head returns to the next frame; if `play` is issued from a sprite script, the playback head returns to the same frame.

Each `play` command needs a matching `play done` command to avoid using up memory if the original calling script isn't returned to. To avoid this memory consumption, you can use a global variable to record where the movie should return to.

The `play done` command has no effect in a movie that is playing in a window.

Example This handler returns the playback head to the frame of the movie that was playing before the current movie started:

```
on exitFrame
  play done
end
```

See also **play**

playing

Syntax

```
sprite(whichFlashSprite).playing
the playing of sprite whichFlashSprite
```

Description

Flash sprite property; indicates whether a Flash movie is playing (`TRUE`) or stopped (`FALSE`).

This property can be tested but not set.

Example

This frame script checks to see if the Flash movie sprite in channel 5 is playing and, if it is not, starts the movie:

```
on enterFrame
  if not sprite(5).playing then
    sprite(5).play()
  end if
end
```

play member

Syntax

```
member(whichCastMember).play()
play member whichCastMember
```

Description

Command; begins playback of a Shockwave Audio (SWA) streaming cast member.

If the sound has not been preloaded by means of the `preLoadBuffer` command, the SWA sound preloads before playing begins. When the sound is playing, the `state` member property equals 3.

Be aware that Xtras to support this functionality must be included when playing back a streaming sound.

Example

This handler begins the playback of the cast member Big Band:

```
on mouseDown
  member("Big Band").play()
end
```

See also

pause member, **stop member**

playNext()

Syntax `sound(channelNum).playNext()`

`playNext(sound(channelNum))`

Description This command immediately interrupts playback of the current sound playing in the given sound channel and begins playing the next queued sound. If no more sounds are queued in the given channel, the sound simply stops playing.

Example This statement plays the next queued sound in sound channel 2.

```
sound(2).playNext()
```

See also **pause() (sound playback)**, **play() (sound)**, **stop() (sound)**

point()

Syntax `point(horizontal, vertical)`

Description Function and data type; yields a point that has the horizontal coordinate specified by *`horizontal`* and the vertical coordinate specified by *`vertical`*.

A point has a `locH` and a `locV` property. Point coordinates can be changed by arithmetic operations.

To see an example of `point()` used in a completed movie, see the Imaging and Vector Shapes movies in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement sets the variable `lastLocation` to the point (250, 400):

```
set lastLocation = point(250, 400)
```

Example This statement adds 5 pixels to the horizontal coordinate of the point assigned to the variable `myPoint`:

```
myPoint.locH = myPoint.locH + 5
```

Example These statements set a sprite's Stage coordinates to `mouseH` and `mouseV` plus 10 pixels. The two statements are equivalent.

```
sprite(the clickOn).loc = point(the mouseH, the mouseV) +
point(10, 10)
sprite(the clickOn).loc = the mouseLoc + 10
```

Example This handler moves a named sprite to the location that the user clicks:

```
end mouseDown
on mouseDown
  -- Set these variables as needed for your own movie
  theSprite = 1 -- Set the sprite that should move
  steps = 40 -- Set the number of steps to get there
  initialLoc = sprite(theSprite).loc
  delta = (the clickLoc - initialLoc) / steps
  repeat with i = 1 to steps
    sprite(theSprite).loc = initialLoc + (i * delta)
    updateStage
  end repeat
end mouseDown
```

See also **mouseLoc**, **flashToStage()**,**rect()**,**stageToFlash()**

pointInHyperlink()

Syntax `sprite(whichSpriteNumber).pointInHyperlink(point)`

`pointInHyperlink(sprite whichSpriteNumber, point)`

Description Text sprite function; returns a value (`TRUE` or `FALSE`) that indicates whether the specified point is within a hyperlink in the text sprite. Typically, the point used is the cursor position. This is useful for setting custom cursors.

See also **cursor (command)**, **mouseLoc**

pointToChar()

Syntax `pointToChar(sprite spriteNumber, pointToTranslate)`

Description Function; returns an integer representing the character position located within the text or field sprite *`spriteNumber`* at screen coordinate *`pointToTranslate`*, or returns -1 if the point is not within the text.

This function can be used to determine the character under the cursor.

Example This Lingo displays the number of the character being clicked, as well as the letter, in the Message window:

```
property spriteNum

 on mouseDown me
  pointClicked = the mouseLoc
  currentMember = sprite(spriteNum).member
  charNum = sprite(spriteNum).pointToChar(pointClicked)
  actualChar = currentMember.char[charNum]
  put "Clicked character" && charNum & ", the letter" &&
actualChar
end
```

See also **mouseLoc**, **pointToWord()**,**pointToItem()**,**pointToLine()**,**pointToParagraph()**

pointToItem()

Syntax

```
sprite(whichSpriteNumber).pointToItem(pointToTransla te)
pointToItem(sprite spriteNumber, pointToTranslate)
```

Description

Function; returns an integer representing the item position in the text or field sprite *spriteNumber* at screen coordinate *pointToTranslate*, or returns -1 if the point is not within the text. Items are separated by the `itemDelimiter` property, which is set to a comma by default.

This function can be used to determine the item under the cursor.

Example

This Lingo displays the number of the item being clicked, as well as the text of the item, in the Message window:

```
property spriteNum
on mouseDown me
  pointClicked = the mouseLoc
  currentMember = sprite(spriteNum).member
  itemNum = sprite(spriteNum).pointToItem(pointClicked)
  itemText = currentMember.item[itemNum]
  put "Clicked item" && itemNum & ", the text" && itemText
end
```

See also

itemDelimiter, **mouseLoc**, **pointToChar()**,**pointToWord()**,**pointToItem()**,**pointToLine()**,**pointToParagraph()**

pointToLine()

Syntax

```
sprite(whichSpriteNumber).pointToLine(pointToTransla te)
pointToLine(sprite spriteNumber, pointToTranslate)
```

Description

Function; returns an integer representing the line position in the text or field sprite *spriteNumber* at screen coordinate *pointToTranslate*, or returns -1 if the point is not within the text. Lines are separated by carriage returns in the text or field cast member.

This function can be used to determine the line under the cursor.

Example

This Lingo displays the number of the line being clicked, as well as the text of the line, in the Message window:

```
property spriteNum

 on mouseDown me
  pointClicked = the mouseLoc
  currentMember = sprite(spriteNum).member
  lineNum = sprite(spriteNum).pointToLine(pointClicked)
  lineText = currentMember.line[lineNum]
  put "Clicked line" && lineNum & ", the text" && lineText
end
```

See also

itemDelimiter, **mouseLoc**, **pointToChar()**,**pointToWord()**,**pointToItem()**,**pointToLine()**,**pointToParagraph()**

pointToParagraph()

Syntax

```
sprite(whichSpriteNumber).pointToParagraph(pointToTr anslate)
pointToParagraph(sprite spriteNumber, pointToTranslate)
```

Description

Function; returns an integer representing the paragraph number located within the text or field sprite *spriteNumber* at screen coordinate *pointToTranslate*, or returns -1 if the point is not within the text. Paragraphs are separated by carriage returns in a block of text.

This function can be used to determine the paragraph under the cursor.

Example

This Lingo displays the number of the paragraph being clicked, as well as the text of the paragraph, in the message window:

```
property spriteNum

 on mouseDown me
  pointClicked = the mouseLoc
  currentMember = sprite(spriteNum).member
  paragraphNum = sprite(spriteNum).pointToParagraph(pointClicked)
  paragraphText = currentMember.paragraph[paragraphNum]
  put "Clicked paragraph" && paragraphNum & ", the text" &&
paragraphText
end
```

See also

itemDelimiter, **mouseLoc**, **pointToChar()**,**pointToWord()**,**pointToItem()**,**pointToLine()**

pointToWord()

Syntax
```
sprite(whichSpriteNumber).pointToWord(pointToTransla te)
pointToWord(sprite spriteNumber, pointToTranslate)
```

Description Function; returns an integer representing the number of a word located within the text or field sprite *spriteNumber* at screen coordinate *pointToTranslate*, or returns -1 if the point is not within the text. Words are separated by spaces in a block of text.

This function can be used to determine the word under the cursor.

Example This Lingo displays the number of the word being clicked, as well as the text of the word, in the Message window:

```
property spriteNum

 on mouseDown me
  pointClicked = the mouseLoc
  currentMember = sprite(spriteNum).member
  wordNum = sprite(spriteNum).pointToWord(pointClicked)
  wordText = currentMember.word[wordNum]
  put "Clicked word" && wordNum & ", the text" && wordText
end
```

See also **itemDelimiter**, **mouseLoc**, **pointToChar()**,**pointToItem()**,**pointToLine()**,**pointToParagraph()**

posterFrame

Syntax

```
member(whichFlashMember).posterFrame
the posterFrame of member whichFlashMember
```

Description

Flash cast member property; controls which frame of a Flash movie cast member is used for its thumbnail image. This property specifies an integer corresponding to a frame number in the Flash movie.

This property can be tested and set. The default value is 1.

Example

This handler accepts a reference to a Flash movie cast member and a frame number as parameters, and it then sets the thumbnail of the specified movie to the specified frame number:

```
on resetThumbnail whichFlashMovie, whichFrame
  member(whichFlashMovie).posterFrame = whichFrame
end
```

postNetText

Syntax

```
postNetText(url, propertyList {,serverOSString}
{,serverCharSetString})
```

```
postNetText(url, postText {,serverOSString}
{,serverCharSetString})
```

Description

Command; sends a `POST` request to *`url`*, which is an HTTP URL, with *`postText`* as the data.

This command is similar to `getNetText()`. As with `getNetText()`, the server's response is returned by `netTextResult(`*`netID`*`)` once `netDone(`*`netID`*`)` becomes 1, and if `netError(`*`netID`*`)` is 0, or okay.

When a property list is used instead of a string, the information is sent in the same way a browser posts an HTML form, with `METHOD=POST`. This facilitates the construction and posting of form data within a Director title. Property names correspond to HTML form field names and property values to field values.

The property list can use either strings or symbols as the property names. If a symbol is used, it is automatically converted to a string without the # at the beginning. Similarly, a numeric value is converted to a string when used as the value of a property.

Note: If a program uses the alternate form—a string instead of property list—the string postText is sent to the server as an HTTP POST request using MIME type "text/plain." This will be convenient for some applications, but is not compatible with HTML forms posting.

The optional parameter *`serverOSString`* defaults to `UNIX` but may be set to `Windows` or `Mac` and translates any carriage returns in the *`postText`* argument into those used on the server to avoid confusion. For most applications, this setting is unnecessary because line breaks are usually not used in form responses.

The optional parameter *`serverCharSetString`* applies only if the user is running on a Shift-JIS (Japanese) system. Its possible settings are `"JIS"`, `"EUC"`, `"ASCII"`, and `"AUTO"`. Posted data is converted from Shift-JIS to the named character set. Returned data is handled exactly as by `getNetText()` (converted from the named character set to Shift-JIS). If you use `"AUTO"`, the posted data from the local character set is not translated; the results sent back by the server are translated as they are for `getNetText()`. `"ASCII"` is the default if *`serverCharSetString`* is omitted. `"ASCII"` provides no translation for posting or results.

The optional arguments may be omitted without regard to position.

This command also has an additional advantage over `getNetText()`: a `postNetText()` query can be arbitrarily long, whereas the `getNetText()` query is limited to the length of a URL (1K or 4K, depending on the browser).

Note: If you use `postNetText` to post data to a domain different from the one the movie is playing from, the movie will display a security alert when playing back in Shockwave.

To see an example of `postNetText` used in a completed movie, see the Forms and Post movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement omits the *`serverCharSetString`* parameter:

```
netID = postNetText("www.mydomain.com\database.cgi", "Bill Jones",
```

```
"Win")
```

Example This example generates a form from user-entry fields for first and last name, along with a Score. Note that both *serverOSString* and *serverCharSetString* have been omitted:

```
lastName = member("Last Name").text
firstName = member("First Name").text
totalScore = member("Current Score").text
infoList = ["FName":firstName, "LName":lastName,
"Score":totalScore]
netID = postNetText("www.mydomain.com\userbase.cgi", infoList)
```

See also **getNetText()**,**netTextResult()**,**netDone()**,**netError()**

power()

Syntax `power(base, exponent)`

Description Math function; calculates the value of the number specified by *`base`* to the exponent specified by *`exponent`*.

Example This statement sets the variable `vResult` to the value of 4 to the third power:

```
set vResult = power(4,3)
```

preLoad (command)

Syntax
```
preLoad
preLoad toFrameNum
preLoad fromFrame, toFrameNum
```

Description Command; preloads cast members in the specified frame or range of frames into memory and stops when memory is full or when all of the specified cast members have been preloaded, as follows:

- When used without arguments, the command preloads all cast members used from the current frame to the last frame of a movie.
- When used with one argument, *toFrame*, the command preloads all cast members used in the range of frames from the current frame to the frame *toFrameNum*, as specified by the frame number or label name.
- When used with two arguments, *fromFrame* and *toFrameNum*, preloads all cast members used in the range of frames from the frame *fromFrame* to the frame *toFrameNum*, as specified by the frame number or label name.

The `preLoad` command also returns the number of the last frame successfully loaded. To obtain this value, use the `result` function.

Example This statement preloads the cast members used from the current frame to the frame that has the next marker:

```
preLoad marker (1)
```

Example This statement preloads the cast members used from frame 10 to frame 50:

```
preLoad 10, 50
```

See also **preLoadMember**

preLoad (cast member property)

Syntax

```
member(whichCastMember).preLoad
```

```
the preLoad of member whichCastMember
```

Description

Cast member property; determines whether the digital video cast member specified by *whichCastMember* can be preloaded into memory (TRUE) or not (FALSE, default). The TRUE status has the same effect as selecting Enable Preload in the Digital Video Cast Member Properties dialog box.

For Flash movie cast members, this property controls whether a Flash movie must load entirely into RAM before the first frame of a sprite is displayed (TRUE), or whether the movie can stream into memory as it plays (FALSE, default). This property works only for linked Flash movies whose assets are stored in an external file; it has no effect on members whose assets are stored in the cast. The streamMode and bufferSize properties determine how the cast member is streamed into memory.

This property can be tested and set.

Example

This statement reports in the Message window whether the QuickTime movie Rotating Chair can be preloaded into memory:

```
put member("Rotating Chair").preload
```

Example

This startMovie handler sets up a Flash movie cast member for streaming and then sets its bufferSize property:

```
on startMovie
  member("Flash Demo").preLoad = FALSE
  member("Flash Demo").bufferSize = 65536
end
```

See also

bufferSize, **streamMode**

preLoadBuffer member

Syntax

```
member(whichCastMember).preLoadBuffer()
```

preLoadBuffer member *whichCastMember*

Description

Command; preloads part of a specified Shockwave Audio (SWA) file into memory. The amount preloaded is determined by the `preLoadTime` property. This command works only if the SWA cast member is stopped.

When the `preLoadBuffer` command succeeds, the `state` member property equals 2.

Most SWA cast member properties can be tested only after the `preLoadBuffer` command has completed successfully. These properties include: `cuePointNames`, `cuePointTimes`, `currentTime`, `duration`, `percentPlayed`, `percentStreamed`, `bitRate`, `sampleRate`, and `numChannels`.

Example

This statement loads the cast member Mel Torme into memory:

```
member("Mel Torme").preLoadBuffer()
```

See also

preLoadTime

preLoadEventAbort

Syntax `the preLoadEventAbort`

Description Movie property; specifies whether pressing keys or clicking the mouse can stop the preloading of cast members (`TRUE`) or not (`FALSE`, default).

This property can be tested and set. Setting this property affects the current movie.

Example This statement lets the user stop the preloading of cast members by pressing keys or clicking the mouse:

```
set the preLoadEventAbort = TRUE
```

See also **preLoad (command)**, **preLoadMember**

preLoadMember

Syntax

```
preLoadMember
member(whichCastMember).preLoad()
preLoadMember whichCastMember
member(fromCastmember).preLoad(toCastMember)
preLoadMember fromCastmember, toCastmember
```

Description

Command; preloads cast members and stops when memory is full or when all of the specified cast members have been preloaded. The `preLoadMember` command returns the cast member number of the last cast member successfully loaded. To obtain this value, use the `result` function.

When used without arguments, `preLoadMember` preloads all cast members in the movie.

When used with the *`whichCastMember`* argument, `preLoadMember` preloads just that cast member. If *`whichCastMember`* is an integer, only the first cast is referenced. If *`whichCastMember`* is a string, the first member with the string as its name will be used.

When used with the arguments *`fromCastmember`* and *`toCastmember`*, the `preLoadMember` command preloads all cast members in the range specified by the cast member numbers or names.

Example

This statement preloads cast member 20 of the first (internal) cast:

```
member(20).preLoad()
```

Example

This statement preloads cast member Shrine and the 10 cast members after it in the Cast window:

```
member("Shrine").preLoad(member("Shrine").number + 10)
```

Example

To preload a specific member of a specific cast, use the following syntax:

```
member("Mumia Abu Jamal", "Black Panthers").preLoad()
```

preLoadMode

Syntax `castLib(whichCast).preLoadMode`

the preLoadMode of castLib *whichCast*

Description Cast member property; determines the specified cast's preload mode and has the same effect as setting Load Cast in the Cast Properties dialog box. Possible values are the following:

- 0—Load cast when needed.
- 1—Load cast before frame 1.
- 2—Load cast after frame 1.

The default value for cast members is 0, when needed.

An `on prepareMovie` handler is usually a good place for Lingo that determines when cast members are loaded.

This property can be tested and set.

Example The following statement tells Director to load the members of the cast Buttons before the movie enters frame 1:

```
CastLib("Buttons").preLoadMode = 1
```

preLoadMovie

Syntax `preLoadMovie *whichMovie*`

Description Command; preloads the data and cast members associated with the first frame of the specified movie. Preloading a movie helps it start faster when it is started by a `go to movie` or `play movie` command.

To preload cast members from a URL, use `preloadNetThing() to` load the cast members directly into the cache, or use `downLoadNetThing` to load a movie on a local disk from which you can load the movie into memory and minimize downloading time.

Example This statement preloads the movie Introduction, which is located in the same folder as the current movie:

```
preLoadMovie "Introduction"
```

preloadNetThing()

Syntax `preloadNetThing (url)`

Description Function; preloads a file from the Internet to the local cache so it can be used later without a download delay. Replace *url* with the name of any valid Internet file, such as a Director movie, graphic, or FTP server location. The return value is a network ID that you can use to monitor the progress of the operation.

The Director player for Java doesn't support this command because Java's security model doesn't allow writing to the local disk.

The `preloadNetThing()` function downloads the file while the current movie continues playing. Use `netDone()` to find out whether downloading is finished.

After an item is downloaded, it can be displayed immediately because it is taken from the local cache rather than from the network.

Although many network operations can be active at a time, running more than four concurrent operations usually slows down performance unacceptably.

Neither the cache size nor the Check Documents option in a browser's preferences affects the behavior of the `preloadNetThing` function.

The `preloadNetThing()` function does not parse a Director file's links. Thus, even if a Director file is linked to casts and graphic files, `preloadNetThing()` downloads only the Director file. You still must preload other linked objects separately.

Example This statement uses `preloadNetThing()` and returns the network ID for the operation:

```
set mynetid =
preloadNetThing("http://www.yourserver.com/menupage/mymovie.dir")
```

After downloading is complete, you can navigate to the movie using the same URL. The movie will be played from the cache instead of the URL, since it's been loaded in the cache.

See also **netDone()**

preLoadRAM

Syntax `the preLoadRAM`

Description System property; specifies the amount of RAM that can be used for preloading a digital video. This property can be set and tested.

This property is useful for managing memory, limiting digital video cast members to a certain amount of memory, so that other types of cast members can still be preloaded. When `preLoadRAM` is `FALSE`, all available memory can be used for preloading digital video cast members.

However, it's not possible to reliably predict how much RAM a digital video will require once it is preloaded, because memory requirements are affected by the content of the movie, how much compression was performed, the number of keyframes, changing imagery, and so on.

It is usually safe to preload the first couple of seconds of a video and then continue streaming from that point on.

If you know the data rate of your movie, you can estimate the setting for `preLoadRAM`. For example, if your movie has a data rate of 300K per second, set `preLoadRAM` to 600K if you want to preload the first 2 seconds of the video file. This is only an estimate, but it works in most situations.

Example This statement sets `preLoadRAM` to 600K, to preload the first 2 seconds of a movie with a data rate of 300K per second:

```
set the preLoadRAM = 600
```

See also **loop (keyword)**, **next**

preLoadTime

Syntax

```
member(whichCastMember).preLoadTime
```

```
the preLoadTime of member whichCastMember
```

```
sound(channelNum).preLoadTime
```

Description

Cast member and sound channel property; for cast members, specifies the amount of the Shockwave Audio (SWA) streaming cast member to download, in seconds, before playback begins or when a `preLoadBuffer` command is used. The default value is 5 seconds.

This property can be set only when the SWA streaming cast member is stopped.

For sound channels, the value is for the given sound in the queue or the currently playing sound if none is specified.

Example

This handler sets the preload download time for the SWA streaming cast member Louis Armstrong to 6 seconds. The actual preload occurs when a `preLoadBuffer` or `play` command is issued.

```
on mouseDown
  member("Louis Armstrong").stop()
  ember("Louis Armstrong").preLoadTime = 6
end
```

Example

This statement returns the preLoadTime of the currently playing sound in sound channel 1:

```
put sound(1).preLoadTime
```

See also

preLoadBuffer member

on prepareFrame

Syntax

```
on prepareFrame
  statement(s)
end
```

Description

System message and event handler; contains statements that run immediately before the current frame is drawn.

Unlike `beginSprite` and `endSprite` events, a `prepareFrame` event is generated each time the playback head enters a frame.

The `on prepareFrame` handler is a useful place to change sprite properties before the sprite is drawn.

If used in a behavior, the `on prepareFrame` handler receives the reference `me`.

The `go`, `play`, and `updateStage` commands are disabled in an `on prepareFrame` handler.

Example

This handler sets the `locH` property of the sprite that the behavior is attached to:

```
on prepareFrame me
  sprite(me.spriteNum).locH  = the mouseH
end
```

See also

on enterFrame

on prepareMovie

Syntax

```
on prepareMovie
  statement(s)
end
```

Description System message and event handler; contains statements that run after the movie preloads cast members but before the movie does the following:

- Creates instances of behaviors attached to sprites in the first frame that plays.
- Prepares the first frame that plays, including drawing the frame, playing any sounds, and executing transitions and palette effects.

New global variables used for sprite behaviors in the first frame should be initialized in the `on prepareMovie` handler. Global variables already set by the previous movie do not need to be reset.

An `on prepareMovie` handler is a good place to put Lingo that creates global variables, initializes variables, plays a sound while the rest of the movie is loading into memory, or checks and adjusts computer conditions such as color depth.

The `go`, `play`, and `updateStage` commands are disabled in an `on prepareMovie` handler.

Example This handler creates a global variable when the movie starts:

```
on prepareMovie
  global currentScore
  set currentScore = 0
end
```

See also **on enterFrame**, **on startMovie**

previous

See **go previous**

printFrom

Syntax `printFrom fromFrame {,toFrame} {,reduction}`

Description Command; prints whatever is displayed on the Stage in each frame, whether or not the frame is selected, starting at the frame specified by *`fromFrame`*. Optionally, you can supply *`toFrame`* and a reduction value (100%, 50%, or 25%).

The frame being printed need not be currently displayed. This command always prints at 72 dots per inch (dpi), bitmaps everything on the screen (text will not be as smooth in some cases), prints in portrait (vertical) orientation, and ignores Page Setup settings. For more flexibility when printing from within Director, see PrintOMatic Lite Xtra, which is on the installation disk.

Example This statement prints what is on the Stage in frame 1:

```
printFrom 1
```

This statement prints what is on the Stage in every frame from the marker Intro to the marker Tale. The reduction is 50%.

```
printFrom label("Intro"), label("Tale"), 50
```

property

Syntax `property {property1}{, property2} {,property3} {...}`

Description Keyword; declares the properties specified by *property1*, *property2*, and so on as property variables.

Declare property variables at the beginning of the parent script or behavior script. You can access them from outside the parent script or behavior script by using the `the` operator.

Note: The `spriteNum` property is available to all behaviors and simply needs to be declared to be accessed.

You can refer to a property within a parent script or behavior script without using the `me` keyword. However, to refer to a property of a parent script's ancestor, use the form `me.property`.

For behaviors, properties defined in one behavior script are available to other behaviors attached to the same sprite.

You can directly manipulate a child object's property from outside the object's parent scripts through syntax similar to that for manipulating other properties. For example, this statement sets the `motionStyle` property of a child object:

```
set the motionStyle of myBouncingObject to #frenetic
```

Use the `count` function to determine the number of properties within the parent script of a child object. Retrieve the name of these properties by using `getPropAt`. Add properties to an object by using `setaProp()`.

To see an example of `property` used in a completed movie, see the Parent Scripts movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement lets each child object created from a single parent script have its own location and velocity setting:

```
property location, velocity
```

Example This parent script handler declares `pMySpriteNum` a property to make it available:

```
-- script Elder
property pMyChannel
on new me, whichSprite
  me.pMyChannel = whichSprite
  return me
end
```

The original behavior script sets up the ancestor and passes the `spriteNum` property to all behaviors:

```
property spriteNum
property ancestor
on beginSprite me
  set ancestor = new(script "Elder", spriteNum)
end
```

See also **me**, **ancestor**, **spriteNum**

proxyServer

Syntax
```
proxyServer serverType, "ipAddress", portNum
proxyServer()
```

Description Command; sets the values of an FTP or HTTP proxy server, as follows:

- *serverType*—#ftp or #http
- *ipAddress*—A string containing the IP address
- *portNum*—The integer value of the port number

If you use the syntax `proxyServer()`, this element returns the settings of an FTP or HTTP proxy server.

Example This statement sets up an HTTP proxy server at IP address 197.65.208.157 using port 5:

```
proxyServer #http,"197.65.208.157",5
```

Example This statement returns the port number of an HTTP proxy server:

```
put proxyServer(#http,#port)
```

If no server type is specified, the function returns 1.

Example This statement returns the IP address string of an HTTP proxy server:

```
put proxyServer(#http)
```

Example This statement turns off an FTP proxy server:

```
proxyServer #ftp,#stop
```

ptToHotSpotID()

Syntax `ptToHotSpotID(whichQTVRSprite, point)`

Description QuickTime VR function; returns the ID of the hotspot, if any, that is at the specified point. If there is no hotspot, the function returns 0.

puppet

Syntax
```
sprite(whichSprite).puppet
the puppet of sprite whichSprite
```

Description Sprite property; determines whether the sprite channel specified by *whichSprite* is a puppet under Lingo control (TRUE) or not (FALSE, default).

- If a sprite channel is a puppet, any changes that Lingo makes to the channel's sprite properties remain in effect after the playback head leaves the sprite.
- If a sprite channel is not a puppet, any changes that Lingo makes to a sprite last for the life of the current sprite only.

While the playback head is in the same sprite, setting the sprite channel's puppet sprite property to FALSE resets the sprite's properties to those set in the Score.

Making the sprite channel a puppet lets you control many sprite properties, such as member, locH, and width, from Lingo after the playback head exits from the sprite.

Setting the puppet sprite property is equivalent to using the puppetSprite command. For example, the following statements are equivalent: `set the puppet of sprite 1 to TRUE` and `puppetSprite 1, TRUE`.

This property can be tested and set.

Example This statement makes the sprite numbered i + 1 a puppet:

```
sprite(i + 1).puppet = TRUE
```

Example This statement records whether sprite 5 is a puppet by assigning the value of the puppet sprite property to the variable. When sprite 5 is a puppet, isPuppet is set to TRUE. When sprite 5 is not a puppet, isPuppet is set to FALSE.

```
isPuppet = sprite(5).puppet
```

See also **puppetSprite**

puppetPalette

Syntax `puppetPalette whichPalette {, speed} {,nFrames}`

Description Command; causes the palette channel to act as a puppet and lets Lingo override the palette setting in the palette channel of the Score and assign palettes to the movie.

The `puppetPalette` command sets the current palette to the palette cast member specified by *`whichPalette`*. If *`whichPalette`* evaluates to a string, it specifies the cast name of the palette. If *`whichPalette`* evaluates to an integer, it specifies the member number of the palette.

For best results, use the `puppetPalette` command before navigating to the frame on which the effect will occur so that Director can map to the desired palette before drawing the next frame.

You can fade in the palette by replacing *`speed`* with an integer from 1(slowest) to 60 (fastest). You can also fade in the palette over several frames by replacing *`nFrames`* with an integer for the number of frames.

A puppet palette remains in effect until you turn it off with the command `puppetPalette 0`. No subsequent palette changes in the Score are obeyed when the puppet palette is in effect.

Note: The browser controls the palette for the entire Web page. Thus, Shockwave and the Director player for Java always uses the browser's palette.

For the most reliable color when authoring a movie for playback as a Director player for Java, use the default palette for the authoring system.

Example This statement makes Rainbow the movie's palette:

```
puppetPalette "Rainbow"
```

Example This statement makes Grayscale the movie's palette. The transition to the Grayscale palette occurs over a time setting of 15 and between frames labeled Gray and Color.

```
puppetPalette "Grayscale", 15, label("Gray") - label("Color")
```

puppetSound

Syntax
```
puppetSound whichChannel, whichCastMember
puppetSound whichCastMember
puppetSound member whichCastMember
puppetSound 0
puppetSound whichChannel, 0
```

Description Command; makes the sound channel a puppet, plays the sound cast member specified by *whichCastMember*, and lets Lingo override any sounds assigned in the Score's sound channels.

Specify a sound channel by replacing *whichChannel* with a channel number.

The sound starts playing after the playback head moves or the `updateStage` command is executed. Using 0 as the cast number argument stops the sound from playing. It also returns control of the sound channel to the Score.

Puppet sounds can be useful for playing a sound while a different movie is being loaded into memory.

The Director player for Java supports the following versions of the `puppetSound` command:

- puppetSound whichChannel, whichCastMember, or puppetSound whichCastMember— Plays a sound.
- puppetSound 0 or puppetSound whichChannel, 0—Stops a sound.

Example This statement plays the sound Wind under control of Lingo:

```
puppetSound "Wind"
```

Example This statement turns off the sound playing in channel 2:

```
puppetSound 2, 0
```

See also **sound fadeIn**, **sound fadeOut**, **sound playFile**, **sound stop**

puppetSprite

Syntax `puppetSprite whichChannel, state`

Description Command; determines whether the sprite channel specified by *`whichSprite`* is a puppet and under Lingo control (`TRUE`) or not a puppet and under the control of the Score (`FALSE`).

While the playback head is in the same sprite, turning off the sprite channel's puppetting using the command `puppetSprite` *`whichSprite`*, `FALSE` resets the sprite's properties to those in the Score.

The sprite channel's initial properties are whatever the channel's settings are when the `puppetSprite` command is executed. You can use Lingo to change sprite properties as follows:

- If a sprite channel is a puppet, any changes that Lingo makes to the channel's sprite properties remain in effect after the playback head exits the sprite.
- If a sprite channel is not a puppet, any changes that Lingo makes to a sprite last for the life of the current sprite only.

The channel must contain a sprite when you use the `puppetSprite` command.

Making the sprite channel a puppet lets you control many sprite properties— such as `memberNum`, `locH`, and `width`—from Lingo after the playback head exits the sprite.

Use the command `puppetSprite` *`whichSprite`*, `FALSE` to return control to the Score when you finish controlling a sprite channel from Lingo and to avoid unpredictable results that may occur when the playback head is in frames that aren't intended to be puppets.

Note: Version 6 of Director introduced autopuppeting, which made it unnecessary to explicitly puppet a sprite under most circumstances. Explicit control is still useful if you want to retain complete control over a channel's contents even after a sprite span has finished playing.

Example This statement makes the sprite in channel 15 a puppet:

```
puppetSprite 15, TRUE
```

Example This statement removes the puppet condition from the sprite in the channel numbered i + 1:

```
puppetSprite i + 1, FALSE
```

See also **backColor**, **bottom**, **constraint**, **foreColor**, **height**, **ink**, **left**, **lineSize**, **locH**, **locV**, **memberNum**, **puppet**, **right**, **top**, **type (sprite property)**, and **width**; **cursor (command)** and **puppetSprite**

puppetTempo

Syntax `puppetTempo framesPerSecond`

Description Command; causes the tempo channel to act as a puppet and sets the tempo to the number of frames specified by *`framesPerSecond`*. When the tempo channel is a puppet, Lingo can override the tempo setting in the Score and change the tempo assigned to the movie.

It's unnecessary to turn off the puppet tempo condition to make subsequent tempo changes in the Score take effect.

Note: Although it is theoretically possible to achieve frame rates up to 30,000 frames per second (fps) with the puppetTempo command, you could do this only with little animation and a very powerful machine.

Example This statement sets the movie's tempo to 30 fps:

```
puppetTempo 30
```

Example This statement increases the movie's old tempo by 10 fps:

```
puppetTempo oldTempo + 10
```

puppetTransition

Syntax

```
puppetTransition member whichCastMember
puppetTransition whichTransition {,time} {, chunkSize} {, changeArea}
```

Description

Command; performs the specified transition between the current frame and the next frame.

To use an Xtra transition cast member, use `puppetTransition member` followed by the cast member's name or number.

To use a built-in Director transition, replace *`whichTransition`* with a value in the following table. Replace *`time`* with the number of quarter seconds used to complete the transition. The minimum value is 0; the maximum is 120 (30 seconds). Replace *`chunkSize`* with the number of pixels in each chunk of the transition. The minimum value is 1; the maximum is 128. Smaller chunk sizes yield smoother transitions but are slower.

Code	Transition	Code	Transition
01	Wipe right	27	Random rows
02	Wipe left	28	Random columns
03	Wipe down	29	Cover down
04	Wipe up	30	Cover down, left
05	Center out, horizontal	31	Cover down, right
06	Edges in, horizontal	32	Cover left
07	Center out, vertical	33	Cover right
08	Edges in, vertical	34	Cover up
09	Center out, square	35	Cover up, left
10	Edges in, square	36	Cover up, right
11	Push left	37	Venetian blinds
12	Push right	38	Checkerboard
13	Push down	39	Strips on bottom, build left
14	Push up	40	Strips on bottom, build right
15	Reveal up	41	Strips on left, build down
16	Reveal up, right	42	Strips on left, build up
17	Reveal right	43	Strips on right, build down
18	Reveal down, right	44	Strips on right, build up
19	Reveal down	45	Strips on top, build left
20	Reveal down, left	46	Strips on top, build right
21	Reveal left	47	Zoom open
22	Reveal up, left	48	Zoom close

23	Dissolve, pixels fast*	49	Vertical blinds
24	Dissolve, boxy rectangles	50	Dissolve, bits fast*
25	Dissolve, boxy squares	51	Dissolve, pixels*
26	Dissolve, patterns	52	Dissolve, bits*

Transitions marked with an asterisk (*) do not work on monitors set to 32 bits.

There is no direct relationship between a low time value and a fast transition. The actual speed of the transition depends on the relation of *chunkSize* and *time*. For example, if *chunkSize* is 1 pixel, the transition takes longer no matter how low the time value, because the computer has to do a lot of work. To make transitions occur faster, use a larger chunk size, not a shorter time.

Replace *changeArea* with a value that determines whether the transition occurs only in the changing area (TRUE) or over the entire Stage (FALSE, default). The *changeArea* variable is an area within which sprites have changed.

Example This statement performs a wipe right transition. Because no value is specified for *changeArea*, the transition occurs over the entire Stage, which is the default.

```
puppetTransition 1
```

This statement performs a wipe left transition that lasts 1 second, has a chunk size of 20, and occurs over the entire Stage:

```
puppetTransition 2, 4, 20, FALSE
```

purgePriority

Syntax `member(whichCastMember).purgePriority`

`the purgePriority of member whichCastMember`

Description Cast member property; specifies the purge priority of the cast member specified by *whichCastMember*.

Cast members' purge priorities determine the priority that Director follows to choose which cast members to delete from memory when memory is full. The higher the purge priority, the more likely that the cast member will be deleted. The following `purgePriority` settings are available:

- 0—Never
- 1—Last
- 2—Next
- 3—Normal

Normal, which is the default, lets Director purge cast members from memory at random. Next, Last, and Never allow some control over purging, but Last or Never may cause your movie to run out of memory if several cast members are set to these values.

Setting `purgePriority` for cast members is useful for managing memory when the size of the movie's cast exceeds the available memory. As a general rule, you can minimize pauses while the movie loads cast members and reduce the number of times Director reloads a cast member by assigning a low purge priority to cast members that are used frequently in the course of the movie.

Example This statement sets the purge priority of cast member Background to 3, which makes it one of the first cast members to be purged when memory is needed:

```
member("Background").purgePriority = 3
```

put

Syntax `put expression`

Description Command; evaluates the expression specified by *`expression`* and displays the result in the Message window. This command can be used as a debugging tool by tracking the values of variables as a movie plays.

The Director player for Java displays the message from the `put` command in the browser's Java console window. Access to the console window depends on the browser.

Example This statement displays the time in the Message window:

```
put the time
-- "9:10 AM"
```

Example This statement displays the value assigned to the variable `bid` in the Message window:

```
put bid
-- "Johnson"
```

See also **put...after**, **put...before**, **put...into**

put...after

Syntax `put expression after chunkExpression`

Description Command; evaluates a Lingo expression, converts the value to a string, and inserts the resulting string after a specified chunk in a container, without replacing the container's contents. (If *`chunkExpression`* specifies a nonexistent target chunk, the string value is inserted as appropriate into the container.)

Chunk expressions refer to any character, word, item, or line in any container. Containers include field cast members; text cast members; variables that hold strings; and specified characters, words, items, lines, and ranges within containers.

Example This statement adds the string "fox dog cat" after the contents of the field cast member Animal List:

```
put "fox dog cat" after member "Animal List"
```

The same can be accomplished using this statement:

```
put "fox dog cat" after member("Animal List").line[1]
```

See also **char...of**, **item...of**, **line...of**, **paragraph**, **word...of**; **put...before**, **put...into**

put...before

Syntax `put expression before chunkExpression`

Description Command; evaluates a Lingo expression, converts the value to a string, and inserts the resulting string before a specified chunk in a container, without replacing the container's contents. (If *`chunkExpression`* specifies a nonexistent target chunk, the string value is inserted as appropriate into the container.)

Chunk expressions refer to any character, word, item, or line in any container. Containers include field cast members; text cast members; variables that hold strings; and specified characters, words, items, lines, and ranges in containers.

Example This statement sets the variable `animalList` to the string "fox dog cat" and then inserts the word *elk* before the second word of the list:

```
put "fox dog cat" into animalList
put "elk " before word 2 of animalList
```

The result is the string "fox elk dog cat".

The same can be accomplished using this syntax:

```
put "fox dog cat" into animalList
put "elk " before animalList.word[2]
```

See also **char...of**, **item...of**, **line...of**, **paragraph**, **word...of**; **put...after**, **put...into**

put...into

Syntax `put expression into chunkExpression`

Description Command; evaluates a Lingo expression, converts the value to a string, and uses the resulting string to replace a specified chunk in a container. (If *`chunkExpression`* specifies a nonexistent target chunk, the string value is inserted as appropriate into the container.)

Chunk expressions refer to any character, word, item, or line in any container. Containers include field cast members; text cast members; variables that hold strings; and specified characters, words, items, lines, and ranges in containers.

When a movie plays back as an applet, the `put...into` command replaces all text within a container, not chunks of text.

To assign values to variables, use the `set` command.

Example This statement changes the second line of the field cast member Review Comments to "Reviewed by Agnes Gooch":

```
put "Reviewed by Agnes Gooch" into line 2 of member "Review
Comments"
```

The same can be accomplished with a text cast member using this syntax:

```
put "Reviewed by Agnes Gooch" into member("Review
Comments").line[2]
```

See also **char...of**, **item...of**, **line...of**, **paragraph**, **word...of**, **put...after**, **put...before**, **set...to, set...=**

qtRegisterAccessKey

Syntax `qtRegisterAccessKey(categoryString, keyString)`

Description Command; allows registration of a key for encrypted QuickTime media.

The key is an application-level key, not a system-level key. After the application unregisters the key or shuts down, the media will no longer be accessible.

Note: For security reasons, there is no way to display a listing of all registered keys.

See also **qtUnRegisterAccessKey**

qtUnRegisterAccessKey

Syntax `qtUnRegisterAccessKey(categoryString, keyString)`

Description Command; allows the key for encrypted QuickTime media to be unregistered.

The key is an application-level key, not a system-level key. After the application unregisters the key, only movies encrypted with this key continue to play. Other media will no longer be accessible.

See also **qtRegisterAccessKey**

quad

Syntax `sprite(whichSpriteNumber).quad`

Description Sprite property; contains a list of four points, which are floating point values that describe the corner points of a sprite on the Stage. The points are organized in the following order: upper left, upper right, lower right, and lower left.

The points themselves can be manipulated to create perspective and other image distortions.

After you manipulate the quad of a sprite, you can reset it to the Score values by turning off the puppet of the sprite with `puppetSprite` *`whichSpriteNumber`*`, FALSE`. When the quad of a sprite is disabled, you cannot rotate or skew the sprite.

To see an example of `quad` used in a completed movie, see the Quad movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement displays a typical list describing a sprite:

```
put sprite(1).quad
-- [point(153.0000, 127.0000), point(231.0000, 127.0000),  \
  point(231.0000, 242.0000), point(153.0000, 242.0000)]
```

Example When modifying this sprite property, be aware that you must reset the list of points after changing any of the values. This is because when you set a variable to the value of a property, you are placing a copy of the list, not the list itself, in the variable. To effect a change, use syntax like this:

```
-- Get the current property contents
currQuadList = sprite(5).quad
-- Add 50 pixels to the horizontal and vertical positions of the
first point in the list
currQuadList[1] = currQuadList[1] + point(50, 50)
-- Reset the actual property to the newly computed position
sprite(5).quad = currQuadList
```

See also **rotation**, **skew**

quality

Syntax

```
sprite(whichFlashSprite).quality
the quality of sprite whichFlashSprite
member(whichFlashMember).quality
the quality of member whichFlashMember
```

Description

Flash cast member and sprite property; controls whether Director uses anti- aliasing to render a Flash movie sprite, producing high-quality rendering but possibly slower movie playback. The `quality` property can have these values:

- `#autoHigh`—Director starts by rendering the sprite with anti-aliasing. If the actual frame rate falls below the movie's specified frame rate, Director turns off anti-aliasing. This setting gives precedence to playback speed over visual quality.
- `#autoLow`—Director starts by rendering the movie without anti-aliasing. If the Flash player determines that the computer processor can handle it, anti- aliasing is turned on. This setting gives precedence to visual quality whenever possible.
- `#high` (default)—The movie always plays with anti-aliasing.
- `#low`—The movie always plays without anti-aliasing.

The `quality` property can be tested and set.

Example

This sprite script checks the color depth of the computer on which the movie is playing. If the color depth is set to 8 bits or less (256 colors), the script sets the quality of the sprite in channel 5 to `#low`.

```
on beginSprite me
  if the colorDepth <= 8 then
    sprite(1).quality = #low
  end if
end
```

queue()

Syntax

```
sound(channelNum).queue(member(whichMember))
```

```
sound(channelNum).queue([#member: member(whichmember), {#startTime: milliseconds, #endTime: milliseconds, #loopCount: numberOfLoops, #loopStartTime: milliseconds, #loopEndTime: milliseconds, #preloadTime: milliseconds}])
```

```
queue(sound(channelNum), member(whichMember))
```

```
queue(soundObject, [#member: member(whichmember), {#startTime: milliseconds, #endTime: milliseconds, #loopCount: numberOfLoops, #loopStartTime: milliseconds, #loopEndTime: milliseconds, #preloadTime: milliseconds}])
```

Description

Function; adds the given sound cast member to the queue of sound channel *channelNum*.

Once a sound has been queued, it can be played immediately with the `play()` command. This is because Director preloads a certain amount of each sound that is queued, preventing any delay between the `play()` command and the start of playback. The default amount of sound that is preloaded is 1500 milliseconds. This parameter can be modified by passing a property list containing one or more parameters with the `queue()` command.

You can specify these properties may be specified with the `queue()` command:

Property	Description
#member	The sound cast member to queue. This property must be provided; all others are optional.
#startTime	The time within the sound at which playback begins, in milliseconds. The default is the beginning of the sound. See startTime.
#endTime	The time within the sound at which playback ends, in milliseconds. The default is the end of the sound. See endTime.
#loopCount	The number of times to play a loop defined with #loopStartTime and #loopEndTime. The default is 1. See loopCount.
#loopStartTime	The time within the sound to begin a loop, in milliseconds. See loopStartTime.
#loopEndTime	The time within the sound to end a loop, in milliseconds. See loopEndTime.
#preloadTime	The amount of the sound to buffer before playback, in milliseconds. See preloadTime.

These parameters can also be passed with the `setPlayList()` command.

To see an example of `queue()` used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This handler queues and plays two sounds. The first sound, cast member Chimes, is played in its entirety. The second sound, cast member introMusic, is played starting at its 3-second point, with a loop repeated 5 times from the 8- second point to the 8.9 second point, and stopping at the 10-second point.

```
on playMusic
```

```
  sound(2).queue([#member: member("Chimes")])
  sound(2).queue([#member: member("introMusic"), #startTime:
3000,\
  #endTime: 10000, #loopCount: 5, #loopStartTime: 8000,
#loopEndTime: 8900])
  sound(2).play()
end
```

See also **startTime**, **endTime**, **loopCount**, **loopStartTime**, **loopEndTime**, **preLoadTime**, **setPlaylist()**,**play() (sound)**

quickTimeVersion()

Syntax `quickTimeVersion()`

Description Function; returns a floating-point value that identifies the current installed version of QuickTime and replaces the current `QuickTimePresent` function.

In Windows, if multiple versions of QuickTime 3.0 or later are installed, `quickTimeVersion()` returns the latest version number. If a version before QuickTime 3.0 is installed, `quickTimeVersion()` returns version number 2.1.2 regardless of the version installed.

Example This statement uses `quickTimeVersion()` to display in the Message window the version of QuickTime that is currently installed:

```
put quickTimeVersion()
```

quit

Syntax `quit`

Description Command; exits from Director or a projector to the Windows desktop or Macintosh Finder.

Example This statement tells the computer to exit to the Windows desktop or Macintosh Finder when the user presses Control+Q in Windows or Command+Q on the Macintosh:

```
if the key = "q" and the commandDown then quit
```

See also **restart**, **shutDown**

QUOTE

Syntax `QUOTE`

Description Constant; represents the quotation mark character and refers to the literal quotation mark character in a string, because the quotation mark character itself is used by Lingo scripts to delimit strings.

Example This statement inserts quotation mark characters in a string:

```
put "Can you spell" && QUOTE & "Macromedia" & QUOTE & "?"
```

The result is a set of quotation marks around the word *Macromedia*:

```
Can you spell "Macromedia"?
```

ramNeeded()

Syntax `ramNeeded (firstFrame, lastFrame)`

Description Function; determines the memory needed, in bytes, to display a range of frames. For example, you can test the size of frames containing 32-bit artwork: if `ramNeeded()` is larger than `freeBytes()`, then go to frames containing 8-bit artwork and divide by 1024 to convert bytes to kilobytes (K).

Example This statement sets the variable `frameSize` to the number of bytes needed to display frames 100 to 125 of the movie:

```
put ramNeeded (100, 125) into frameSize
```

Example This statement determines whether the memory needed to display frames 100 to 125 is more than the available memory, and if it is, branches to the section using cast members that have lower color depth:

```
if ramNeeded (100, 125) > the freeBytes then play frame "8-bit"
```

See also **freeBytes()**,**size**

random()

Syntax `random(integerExpression)`

Description Function; returns a random integer in the range 1 to the value specified by *integerExpression*. This function can be used to vary values in a movie, such as to vary the path through a game, assign random numbers, or change the color or position of sprites.

To start a set of possible random numbers with a number other than 1, subtract the appropriate amount from the `random()` function. For example, the expression `random(n + 1) - 1` uses a range from 0 to the number `n`.

Example This statement assigns random values to the variable `diceRoll`:

```
set diceRoll = random(6) + random(6)
```

This statement randomly changes the foreground color of sprite 10:

```
sprite(10).forecolor = random(256) - 1
```

Example This handler randomly chooses which of two movie segments to play:

```
on SelectScene
  if random(2) = 2 then
    play frame "11a"
  else
    play frame "11-b"
  end if
end
```

Example The following statements produce results in a specific range.

This statement produces a random multiple of 5 in the range 5 to 100:

```
theScore = 5 * random(20)
```

This statement produces a random multiple of 5 in the range 0 to 100:

```
theScore = 5 * (random(21) - 1)
```

This statement generates integers between -10 and +10:

```
dirH = random(21) - 11
```

This statement produces a random two-point decimal value:

```
the floatPrecision = 2
theCents = random(100)/100.0 - .01
```

randomSeed

Syntax `the randomSeed`

Description System property; specifies the seed value used for generating random numbers accessed through the `random()` function.

Using the same seed produces the same sequence of random numbers. This property can be useful for debugging during development. Using the `ticks` property is an easy way to produce a unique random seed since the `ticks` value is highly unlikely to be duplicated on subsequent uses.

This property can be tested and set.

Example This statement displays the random seed number in the Message window:

```
put the randomSeed
```

See also **random()**,**ticks**

rawNew()

Syntax

```
parentScript.rawNew()
```

```
rawNew(parentScript)
```

Description Function; creates a child object from a parent script without calling its `on new` handler. This allows a movie to create child objects without initializing the properties of those child objects. This is particularly useful when you want to create large numbers of child objects for later use. To initialize the properties of one of these raw child objects, call its `on new` handler.

Example This statement creates a child object called RedCar from the parent script CarParentScript without initializing its properties:

```
RedCar = script("CarParentScript").rawNew()
```

Example This statement initializes the properties of the child object RedCar:

```
RedCar.new()
```

new(),**script**

recordFont

Syntax `recordFont(whichCastMember, font {[,face]} {,[bitmapSizes]} {,characterSubset} {, userFontName})`

Description Command; embeds a TrueType or Type 1 font as a cast member. Once embedded, these fonts are available to the author just like other fonts installed in the system.

You must create an empty font cast member with the `new()` command before using `recordFont`.

- *font*—Name of original font to be recorded.
- *face*—List of symbols indicating the face of the original font; possible values are `#plain`, `#bold`, `#italic`. If you do not provide a value for this argument, `#plain` is used.
- *bitmapSizes*—List of integers specifying the sizes for which bitmaps are to be recorded. This argument can be empty. If you omit this argument, no bitmaps are generated. These bitmaps typically look better at smaller point sizes (below 14 points) but take up more memory.
- *characterSubset*—String of characters to be encoded. Only the specified characters will be available in the font. If this argument is, all characters are encoded. If only certain characters are encoded but an unencoded character is used, that character is displayed as an empty box.
- *userFontName*— A string to use as the name of the newly recorded font cast member.

The command creates a Shock Font in *whichCastMember* using the font named in the *font* argument. The value returned from the command reports whether the operation was successful. Zero indicates success.

Example This statement creates a simple Shock Font using only the two arguments for the cast member and the font to record:

```
myNewFontMember = new(#font)
recordFont(myNewFontMember, "Lunar Lander")
```

Example This statement specifies the bitmap sizes to be generated and the characters for which the font data should be created:

```
myNewFontMember = new(#font)
recordfont(mynewmember,"lunar lander",  [],  [14, 18, 45], "Lunar
Lander Game High \
  Score First Last Name")
```

Note: Since `recordFont` resynthesizes the font data rather than using it directly, there are no legal restrictions on Shock Font distribution.

See also **new()**

rect()

Syntax

```
rect(left, top, right, bottom)
rect(point1, point2)
```

Description

Function and data type; defines a rectangle.

The `rect(left, top, right, bottom)` format defines a rectangle whose sides are specified by *`left`*, *`top`*, *`right`*, and *`bottom`*. The *`left`* and *`right`* values specify numbers of pixels from the left edge of the Stage. The *`top`* and *`bottom`* values specify numbers of pixels from the top of the Stage.

The `rect(point1, point2)` format defines a rectangle that encloses the points specified by *`point1`* and *`point2`*.

You can refer to rectangle components by list syntax or property syntax. For example, the following two phrases are equivalent:

```
targetWidth = targetRect.right - targetRect.left
targetWidth = targetRect[3] - targetRect[1]
```

You can perform arithmetic operations on rectangles. If you add a single value to a rectangle, Lingo adds it to each element in the rectangle.

To see an example of `rect()` used in a completed movie, see the Imaging movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement sets the variable `newArea` to a rectangle whose left side is at 100, top is at 150, right side is at 300, and bottom is at 400 pixels:

```
set newArea = rect(100, 150, 300, 400)
```

Example

This statement sets the variable `newArea` to the rectangle defined by the points `firstPoint` and `secondPoint`. The coordinates of `firstPoint` are (100, 150); the coordinates of `secondPoint` are (300, 400). Note that this statement creates the same rectangle as the one created in the previous example:

```
put rect(firstPoint, secondPoint)
```

Example

These statements add and subtract values for rectangles:

```
put rect(0,0,100,100) + rect(30, 55, 120, 95)
-- rect(30, 55, 220, 195)
put rect(0,0,100,100) - rect(30, 55, 120, 95)
-- rect(-30, -55, -20, 5)
```

Example

This statement adds 80 to each coordinate in a rectangle:

```
put rect(60, 40, 120, 200) + 80
-- rect(140, 120, 200, 280)
```

Example

This statement divides each coordinate in a rectangle by 3:

```
put rect(60, 40, 120, 200) / 3
-- rect(20, 13, 40, 66)
```

See also

point(), quad

rect (image)

Syntax `imageObject.rect`

Description Read-only property; returns a rectangle describing the size of the given image object. The coordinates are given relative to the top left corner of the image. The left and top values of the rectangle are therefore always 0, and the right and bottom values are the width and height of the cast member.

Example This statement returns the rectangle of the 300 x 400 pixel member Sunrise in the message window:

```
put member("Sunrise").image.rect
-- rect(0, 0, 300, 400)
```

Example This Lingo looks at the first 50 cast members and displays the rectangle and name of each cast member that is a bitmap:

```
on showAllRects
  repeat with x = 1 to 50
    if member(x).type = #bitmap then
      put member(x).image.rect && "-" && member(x).name
    end if
  end repeat
end
```

See also **height**, **width**

rect (member)

Syntax

```
member(whichCastMember).rect
the rect of member whichCastMember
```

Description

Cast member property; specifies the left, top, right, and bottom coordinates, returned as a rectangle, for the rectangle of any graphic cast member, such as a bitmap, shape, movie, or digital video.

For a bitmap, the `rect` member property is measured from the upper left corner of the bitmap, instead of from the upper left corner of the easel in the Paint window.

For an Xtra cast member, the `rect` member property is a rectangle whose upper left corner is at (0,0).

The Director player for Java can't set the `rect` member property.

This property can be tested. It can be set for field cast members only.

Example

This statement displays the coordinates of bitmap cast member 20:

```
put member(20).rect
```

Example

This statement sets the coordinates of bitmap cast member Banner:

```
member("Banner").rect = rect(100, 150, 300, 400)
```

See also

rect(),rect (sprite)

rect (sprite)

Syntax

```
sprite whichSprite.rect
the rect of sprite whichSprite
```

Description

Sprite property; specifies the left, top, right, and bottom coordinates, as a rectangle, for the rectangle of any graphic sprite such as a bitmap, shape, movie, or digital video.

This property can be tested and set.

Example

This statement displays the coordinates of bitmap sprite 20:

```
put sprite(20).rect
```

rect (window)

Syntax

```
window whichWindow.rect
the rect of window whichWindow
```

Description

Window property; specifies the left, top, right, and bottom coordinates, as a rectangle, of the window specified by *whichWindow*.

If the size of the rectangle specified is less than that of the Stage where the movie was created, the movie is cropped in the window, not resized.

To pan or scale the movie playing in the window, set the `drawRect` or `sourceRect` property of the window.

This property can be tested and set.

Example

This statement displays the coordinates of the window Control Panel:

```
put window("Control Panel").rect
```

See also

drawRect, **sourceRect**

ref

Syntax `chunkExpression.ref`

Description Text chunk expression property; this provides a convenient way to refer to a chunk expression within a text cast member.

Example For example, without references you would need statements like these:

```
member(whichTextMember).line[whichLine].word[firstWord..lastWord].font = "Palatino"
member(whichTextMember).line[whichLine].word[firstWord..lastWord].fontSize = 36
member(whichTextMember).line[whichLine].word[firstWord..lastWord].fontStyle = [#bold]
```

But with a `ref` property you can refer to the same chunk as follows:

```
myRef =
member(whichTextMember).line[whichLine].word[firstWord..lastWord].ref
```

The variable `myRef` is now shorthand for the entire chunk expression. This allows something like the following:

```
put myRef.font
-- "Palatino"
```

Or you can set a property of the chunk as follows:

```
myRef.fontSize = 18
myRef.fontStyle = [#italic]
```

You can get access to the string referred to by the reference using the text property of the reference:

```
put myRef.text
```

This would result in the actual string data, and not information about the string.

regPoint

Syntax

```
member(whichCastMember).regPoint
the regPoint of member whichCastMember
```

Description

Cast member property; specifies the registration point of a cast member. The registration point is listed as the horizontal and vertical coordinates of a point in the form `point(horizontal, vertical)`. Nonvisual members such as sounds do not have a useful `regPoint` property.

You can use the `regPoint` property to animate individual graphics in a film loop, changing the film loop's position in relation to other objects on the Stage.

You can also use `regPoint` to adjust the position of a mask being used on a sprite.

When a Flash movie cast member is first inserted into the cast, its registration point is its center and its `centerRegPoint` property is set to `TRUE`. If you subsequently use the `regPoint` property to reposition the registration point, the `centerRegPoint` property is automatically set to `FALSE`.

This property can be tested and set.

Example

This statement displays the registration point of the bitmap cast member Desk in the Message window:

```
put member("Desk").regPoint
```

Example

This statement changes the registration point of the bitmap cast member Desk to the values in the list:

```
member("Desk").regPoint = point(300, 400)
```

See also

centerRegPoint, **mask**

regPointVertex

Syntax

```
vectorMember.regPointVertex
the regPointVertex of vectorMember
```

Description

Cast member property; indicates whether a vertex of *vectorCastMember* is used as the registration point for that cast member. If the value is zero, the registration point is determined normally, using the `centerRegPoint` and `regPoint` properties. If the value is nonzero, it indicates the position in the `vertextList` of the vertex being used as the registration point. The `centerRegPoint` is set to `FALSE` and the `regPoint` is set to the location of that vertex.

Example

This statement makes the registration point for the vector shape cast member Squiggle correspond to the the location of the third vertex:

```
member("squiggle").regPointVertex=3
```

centerRegPoint, **regPoint**

relative

See @ (pathname)

repeat while

Syntax

```
repeat while testCondition
     statement(s)
end repeat
```

Description

Keyword; repeatedly executes *statement(s)* so long as the condition specified by *testCondition* is TRUE. This structure can be used in Lingo that continues to read strings until the end of a file is reached, checks items until the end of a list is reached, or repeatedly performs an action until the user presses or releases the mouse button.

While in a repeat loop, Lingo ignores other events. To check the current key in a repeat loop, use the keyPressed property.

Only one handler can run at a time. If Lingo stays in a repeat loop for a long time, other events stack up waiting to be evaluated. Therefore, repeat loops are best used for short, fast operations or when users are idle.

If you need to process something for several seconds or more, evaluate the function in a loop with some type of counter or test to track progress.

The Director player for Java doesn't detect mouse movements, update properties that indicate the mouse's position, or update the status of mouse button presses when Lingo is in a repeat loop.

If the stop condition is never reached or there is no exit from the repeat loop, you can force Director to stop by using Control+Alt+period (Windows) or Command+period (Macintosh).

Example

This handler starts the timer counting, resets the timer to 0, and then has the timer count up to 60 ticks:

```
on countTime
  startTimer
  repeat while the timer < 60
  -- waiting for time
  end repeat
end countTime
```

See also

exit, **exit repeat**, **repeat with**, **keyPressed()**

repeat with

Syntax

```
repeat with counter = start to finish
  statement(s)
end
```

Description

Keyword; executes the Lingo specified by *statement(s)* the number of times specified by *counter*. The value of *counter* is the difference between the value specified by *start* and the value specified by *finish*. The counter is incremented by 1 each time Lingo cycles through the repeat loop.

The `repeat with` structure is useful for repeatedly applying the same effect to a series of sprites or for calculating a series of numbers to some exponent.

While in a repeat loop, Lingo ignores other events. To check the current key in a repeat loop, use the `keyPressed` property.

Only one handler can run at a time. If Lingo stays in a repeat loop for a long time, other events stack up waiting to be evaluated. Therefore, repeat loops are best used for short, fast operations or when users are idle.

If you need to process something for several seconds or more, evaluate the function in a loop with some type of counter or test to track progress.

If the stop condition is never reached or there is no exit from the repeat loop, you can force Director to stop by using Control+Alt+period (Windows) or Command+period (Macintosh).

The Director player for Java doesn't detect mouse movements, update properties that indicate the mouse's position, or update the status of mouse button presses when Lingo is in a repeat loop.

Example

The following handler turns sprites 1 through 30 into puppets:

```
on puppetize
  repeat with channel = 1 to 30
    puppetSprite channel, TRUE
  end repeat
end puppetize
```

See also

exit, **exit repeat**, **repeat while**

repeat with...down to

Syntax `repeat with variable = startValue down to endValue`

Description Keyword; counts down by increments of 1 from *`startValue`* to *`endValue`*.

Only one handler can run at a time. If Lingo stays in a repeat loop for a long time, other events stack up waiting to be evaluated. Therefore, repeat loops are best used for short, fast operations or when you know the user won't be doing other things.

While in a repeat loop, Lingo ignores other events. To check the current key in a repeat loop, use the `keyPressed` property.

If you need to process something for several seconds or more, evaluate the function in a loop with some type of counter or test to track progress.

If the stop condition is never reached or there is no exit from the repeat loop, you can force Director to stop by using Control+Alt+period (Windows) or Command+period (Macintosh).

The Director player for Java doesn't detect mouse movements, update properties that indicate the mouse's position, or update the status of mouse button presses when Lingo is in a repeat loop.

Example This handler contains a repeat loop that counts down from 20 to 15:

```
on CountDown
  repeat with i = 20 down to 15
    sprite(6).memberNum = 10 + i
    updateStage
  end repeat
end
```

repeat with...in list

Syntax `repeat with variable in someList`

Description Keyword; assigns successive values from the specified list to the variable.

While in a repeat loop, Lingo ignores other events except keypresses. To check the current key in a repeat loop, use the `keyPressed` property.

Only one handler can run at a time. If Lingo stays in a repeat loop for a long time, other events stack up waiting to be evaluated. Therefore, repeat loops are best used for short, fast operations or when users are idle.

If you need to process something for several seconds or more, evaluate the function in a loop with some type of counter or test to track progress.

If the stop condition is never reached or there is no exit from the repeat loop, you can force Director to stop by using Control+Alt+period (Windows) or Command+period (Macintosh).

The Director player for Java doesn't detect mouse movements, update properties that indicate the mouse's position, or update the status of mouse button presses when Lingo is in a repeat loop.

Example This statement displays four values in the Message window:

```
repeat with i in [1, 2, 3, 4]
  put i
end repeat
```

on resizeWindow

Syntax

```
on resizeWindow
  statement(s)
end
```

Description

System message and event handler; contains statements that run when a movie is running as a movie in a window (MIAW) and the user resizes the window by dragging the window's resize box or one of its edges.

An `on resizeWindow` event handler is a good place to put Lingo related to the window's dimensions, such as Lingo that positions sprites or crops digital video.

Example

This handler moves sprite 3 to the coordinates stored in the variable `centerPlace` when the window that the movie is playing in is resized:

```
on resizeWindow centerPlace
  sprite(3).loc = centerPlace
end
```

See also

drawRect, **sourceRect**

restart

Syntax
```
restart
```

Description
Command; closes all open applications and restarts the computer.

Example
This statement restarts the computer when the user presses Command+R (Macintosh) or Control+R (Windows):

```
if the key = "r" and the commandDown then restart
```

See also
quit, **shutDown**

result

Syntax `the result`

Description Function; displays the value of the return expression from the last handler executed.

The `result` function is useful for obtaining values from movies that are playing in windows and tracking Lingo's progress by displaying results of handlers in the Message window as the movie plays.

This function has no effect in the Director player for Java.

To return a result from a handler, assign the result to a variable and then check the variable's value. Use a statement such as `set myVariable = function()`, where `function()` is the name of a specific function.

Example The following handler returns a random roll for two dice:

```
on diceRoll
  return random(6) + random(6)
end
```

Example The two statements

```
diceRoll
roll = the result
```

are equivalent to this statement:

```
set roll = diceRoll()
```

Note that `set roll = diceRoll` would not call the handler because there are no parentheses following `diceRoll`; `diceRoll` here is considered a variable reference.

See also **return (keyword)**

resume sprite

Syntax `sprite(whichGIFSpriteNumber ).resume()`

resume(sprite whichGIFSpriteNumber)

Description Animated GIF command; causes the sprite to resume playing from the frame after the current frame if it's been paused. This command has no effect if the animated GIF sprite has not been paused.

See also **pause sprite**, **rewind sprite**

RETURN (constant)

Syntax `RETURN`

Description Constant; represents a carriage return.

Example This statement causes a paused movie to continue when the user presses the carriage return:

```
if (the key = RETURN) then go to the frame + 1
```

Example This statement uses the `RETURN` character constant to insert a carriage return between two lines in an alert message:

```
alert "Last line in the file." & RETURN & "Click OK to exit."
```

Example In Windows, it is standard practice to place an additional line-feed character at the end of each line. This statement creates a two-character string named CRLF that provides the additional line feed:

```
CRLF = RETURN & numToChar(10)
```

return (keyword)

Syntax `return expression`

Description Keyword; returns the value of *`expression`* and exits from the handler. The *`expression`* argument can be any Lingo value.

When calling a handler that serves as a user-defined function and has a return value, you must use parentheses around the argument lists, even if there are no arguments, as in the `diceRoll` function handler discussed under the entry for the `result` function.

The function of the `return` keyword is similar to that of the `exit` command, except that `return` also returns a value to whatever called the handler. The `return` command in a handler immediately exits from that handler, but it can return a value to the Lingo that called it.

The use of `return` in object-oriented scripting can be difficult to understand. It's easier to start by using `return` to create functions and exit handlers. Later, you will see that the `return me` line in an `on new` handler gives you a way to pass back a reference to an object that was created so it can be assigned to a variable name.

The `return` keyword isn't the same as the character constant `RETURN`, which indicates a carriage return. The function depends on the context.

To retrieve a returned value, use parentheses after the handler name in the calling statement to indicate that the named handler is a function.

To see an example of `return (keyword)` used in a completed movie, see the Parent Scripts movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This handler returns a random multiple of 5 between 5 and 100:

```
on getRandomScore
  theScore = 5 * random(20)
  return theScore
end getRandomScore
```

Call this handler with a statement similar to the following:

```
set thisScore to GetRandomScore()
```

In this example, the variable `thisScore` is assigned the return value from the function `GetRandomScore`. A parent script performs the same function: by returning the object reference, the variable name in the calling code provides a handle for subsequent references to that object.

See also **result**, **RETURN (constant)**

rewind()

Syntax

```
sound(channelNum).rewind()
rewind(sound(channelNum))
```

Description

Function; interrupts the playback of the current sound in sound channel channelNum and restarts it at its `startTime`. If the sound is paused, it remains paused, with the `currentTime` set to the `startTime`.

Example

This restarts playback of the sound cast member playing in sound channel 1 from the beginning.

```
sound(1).rewind()
```

See also

pause() (sound playback), **play() (sound)**, **playNext()**,**queue()**,**stop() (sound)**

rewind sprite

Syntax `sprite(whichFlashOrGIFAnimSprite).rewind()`

rewind sprite whichFlashOrGIFAnimSprite

Description Command; returns a Flash or animated GIF movie sprite to frame 1 when the sprite is stopped or when it is playing.

Example This frame script checks whether the Flash movie sprite in the sprite the behavior was placed in is playing and, if so, continues to loop in the current frame. When the movie is finished, the sprite rewinds the movie (so the first frame of the movie appears on the Stage) and lets the playback head continue to the next frame.

```
property spriteNum
on exitFrame
  if sprite(spriteNum).playing then
    go the frame
  else
    sprite(spriteNum).rewind()
    updateStage
  end if
end
```

right

Syntax

```
sprite(whichSprite).right
```

```
the right of sprite whichSprite
```

Description

Sprite property; indicates the distance, in pixels, of the specified sprite's right edge from the left edge of the Stage.

When a movie plays back as an applet, this property's value is relative to the upper left corner of the applet.

Sprite coordinates are expressed relative to the upper left corner of the Stage.

This property can be tested and set.

Example

This statement calls the handler `offRightEdge` when the right edge of sprite 3 is past the right edge of the Stage:

```
if sprite(3).right > (the stageRight - the stageLeft) then
offRightEdge
```

See also

bottom, **height**, **left**, **locH**, **locV**, **top**, **width**

rightIndent

Syntax `chunkExpression.rightIndent`

Description Text cast member property; contains the offset distance, in pixels, of the right margin of *`chunkExpression`* from the right side of the text cast member.

The value is an integer greater than or equal to 0.

This property can be tested and set.

See also **firstIndent**, **leftIndent**

on rightMouseDown (event handler)

Syntax

```
on rightMouseDown
  statement(s)
end
```

Description

System message and event handler; in Windows, specifies statements that run when the right mouse button is pressed. On Macintosh computers, the statements run when the mouse button and Control key are pressed simultaneously and the `emulateMultiButtonMouse` property is set to `TRUE`; if this property is set to `FALSE`, this event handler has no effect on the Macintosh.

Example

This handler opens the window Help when the user clicks the right mouse button in Windows:

```
on rightMouseDown
  window("Help").open()
end
```

rightMouseDown (system property)

Syntax `the rightMouseDown`

Description System property; indicates whether the right mouse button (Windows) or the mouse button and Control key (Macintosh) are being pressed (`TRUE`) or not (`FALSE`).

On the Macintosh, `rightMouseDown` is `TRUE` only if the `emulateMultiButtonMouse` property is `TRUE`.

Example This statement checks whether the right mouse button in Windows is being pressed and plays the sound Oops in sound channel 2 if it is:

```
if the rightMouseDown then puppetSound 2, "Oops"
```

See also **emulateMultiButtonMouse**

on rightMouseUp (event handler)

Syntax

```
on rightMouseUp
  statement(s)
end
```

Description

System message and event handler; in Windows, specifies statements that run when the right mouse button is released. On Macintosh computers, the statements run if the mouse button is released while the Control key is pressed and the `emulateMultiButtonMouse` property is set to `TRUE`; if this property is set to `FALSE`, this event handler has no effect on the Macintosh.

Example

This handler opens the Help window when the user releases the right mouse button in Windows:

```
on rightMouseUp
  window("Help").open()
end
```

rightMouseUp (system property)

Syntax `the rightMouseUp`

Description System property; indicates whether the right mouse button (Windows) or the mouse button and Control key (Macintosh) are currently not being pressed (`TRUE`) or are currently being pressed (`FALSE`).

On the Macintosh, `rightMouseUp` is `TRUE` only if the `emulateMultiButtonMouse` property is `TRUE`.

Example This statement checks whether the right mouse button in Windows is released and plays the sound Click Me if it is:

```
if the rightMouseUp then puppetSound 2, "Click Me"
```

See also **emulateMultiButtonMouse**

rollOver()

Syntax
```
rollOver(whichSprite)
the rollOver
```

Description Function; indicates whether the cursor is currently over the bounding rectangle of the sprite specified by *whichSprite* (TRUE or 1) or not (FALSE or 0).

This function has two possible syntax formats:

- When rollOver isn't preceded by the, include parentheses.
- When rollOver is preceded by the, don't include parentheses.

The rollOver function is typically used in frame scripts and is useful for creating handlers that perform an action when the user places the cursor over a specific sprite. It can also simulate additional sprite channels by splitting the Stage into regions that each send the playback head to a different frame that subdivides the region for the available sprite channels.

If the user continues to roll the mouse, the value of rollOver can change while Lingo is running a handler. You can make sure that a handler uses a consistent rollover value by assigning rollOver to a variable when the handler starts.

When the cursor is over the location of a sprite that no longer appears in the Score in the current section, rollOver still occurs and reports the sprite as being there. Avoid this problem by not performing rollovers over these locations or by moving the sprite above the menu bar before removing it.

Example This statement changes the content of the field cast member Message to "This is the place." when the cursor is over sprite 6:

```
if rollover(6) then member("Message").text = "This is the place.""
```

Example This handler sends the playback head to different frames when the cursor is over certain sprites on the Stage. It first assigns the rollOver value to a variable. This lets the handler use the rollOver value that was in effect when the rollover started, regardless of whether the user continues to move the mouse.

```
on exitFrame
  set currentSprite = the rollover
  case currentSprite of
    1: go to frame "Left"
    2: go to frame "Middle"
    3: go to frame "Right"
  end case
end exitFrame
```

See also **mouseMember**

romanLingo

Syntax `the romanLingo`

Description System property; specifies whether Lingo uses a single-byte (`TRUE`) or double-byte interpreter (`FALSE`).

The Lingo interpreter is faster with single-byte character sets. Some versions of Macintosh system software—Japanese, for example—use a double-byte character set. U.S. system software uses a single-byte character set. Normally, `romanLingo` is set when Director is first started and is determined by the local version of the system software.

If you are using a non-Roman script system but don't use any double-byte characters in your script, set this property to `TRUE` for faster execution of your Lingo scripts.

Example This statement sets `romanLingo` to `TRUE`, which causes Lingo to use a single-byte character set:

```
set the romanLingo to TRUE
```

rotation

Syntax

```
the rotation of member whichQuickTimeMember
member(whichQuickTimeMember).rotation
sprite(whichSprite).rotation
the rotation of sprite whichSprite
```

Description

Cast member property and sprite property; controls the rotation of a QuickTime movie, animated GIF, Flash movie, or bitmap sprite within the sprite's bounding rectangle, without rotating that rectangle or the sprite's controller (in the case of QuickTime). In effect, the sprite's bounding rectangle acts as a window through which you can see the Flash or QuickTime movie. The bounding rectangles of bitmaps and animated GIFs change to accommodate the rotating image.

Score rotation works for a Flash movie only if `obeyScoreRotation` is set to `TRUE`.

A Flash movie rotates around its origin point as specified by its `originMode` property. A QuickTime movie rotates around the center of the bounding rectangle of the sprite. A bitmap rotates around the registration point of the image.

For QuickTime media, if the sprite's `crop` property is set to `TRUE`, rotating the sprite frequently moves part of the image out of the viewable area; when the sprite's `crop` property is set to `FALSE`, the image is scaled to fit within the bounding rectangle (which may cause image distortion).

You specify the rotation in degrees as a floating-point number.

The Score can retain information for rotating an image from +21,474,836.47° to -21,474,836.48°, allowing 59,652 full rotations in either direction.

When the rotation limit is reached (slightly past the 59,652th rotation), the rotation resets to +116.47° or -116.48°—not 0.00°. This is because +21,474,836.47° is equal to +116.47°, and -21,474,836.48° is equal to - 116.48° (or +243.12°). To avoid this reset condition, when you use Lingo to perform continuous rotation, constrain the angles to ±360°.

This property can be tested and set. The default value is 0.

Example

The following behavior causes a sprite to rotate continuously by 2° every time the playback head advances, limiting the angle to 360°:

```
property spriteNum
on prepareFrame me
  sprite(spriteNum).rotation = integer(sprite(spriteNum).rotation
+ 2) mod 360
end
```

Example

This frame script keeps the playback head looping in the current frame while it rotates a QuickTime sprite in channel 5 a full 360° in 16° increments. When the sprite has been rotated 360°, the playback head continues to the next frame.

```
on exitFrame
  if sprite(5).rotation < 360 then
    sprite(5).rotation = sprite(5).rotation + 16
    go the frame
  end if
end
```

Example

This handler accepts a sprite reference as a parameter and rotates a Flash movie sprite

360° in 10° increments:

```
on rotateMovie whichSprite
  repeat with i = 1 to 36
    sprite(whichSprite).rotation = i * 10
    updatestage
  end repeat
end
```

See also **flipH**, **flipV**, **obeyScoreRotation**, **originMode**

RTF

Syntax `member(whichMember).RTF`

Description Cast member property; allows access to the text and tags that control the layout of the text within a text cast member containing text in rich text format.

This property can be tested and set.

Example This statement displays in the Message window the RTF formatting information embedded in the text cast member Resume:

```
put member("Resume").RTF
```

See also **HTML**, **importFileInto**

runMode

Syntax `the runMode`

Description Function; returns a string indicating the mode in which the movie is playing. Possible values are as follows:

- `Author`—The movie is running in Director.
- `Projector`—The movie is running as a projector.
- `BrowserPlugin`—The movie is running as a Shockwave plug-in or other scripting environment, such as LiveConnect or ActiveX.
- `Java Applet`—The movie is playing back as a Java applet.

The safest way to test for particular values in this property is to use the `contains` operator. This helps avoid errors and allows partial matches.

Example This statement determines whether or not external parameters are available and obtains them if they are:

```
if the runMode contains "Plugin" then
  -- decode the embed parameter
  if externalParamName(swURL) = swURL then
    put externalParamValue(swURL) into myVariable
  end if
end if
```

See also **environment**, **platform**

on runPropertyDialog

Syntax

```
on runPropertyDialog me, currentInitializerList
  statement(s)
end
```

Description

System message and event handler; contains Lingo that defines specific values for a behavior's parameters in the Parameters dialog box. The `runPropertyDialog` message is sent whenever the behavior is attached to a sprite, or when the user changes the initial property values of a sprite's behavior.

The current settings for a behavior's initial properties are passed to the handler as a property list. If the `on runPropertyDialog` handler is not defined within the behavior, Director runs a behavior customization dialog box based on the property list returned by the `on getPropertyDescriptionList` handler.

Example

This handler overrides the behavior's values set in the Parameters dialog box for the behavior. New values are contained in the list `currentInitializerList`. Normally, the Parameters dialog box allows the user to set the mass and gravitational constants. However, this handler assigns these parameters constant values without displaying a dialog box:

```
property mass
property gravitationalConstant
on runPropertyDialog me, currentInitializerList
  --force mass to 10
  setaProp currentInitializerList, #mass, 10
  -- force gravitationalConstant to 9.8
  setaProp currentInitializerList, #gravitationalConstant, 9.8
  return currentInitializerList
end
```

See also

on getBehaviorDescription, **on getPropertyDescriptionList**

safePlayer

Syntax `the safePlayer`

Description System property; controls whether or not safety features in Director are turned on.

In a Shockwave movie, this property can be tested but not set. It is always `TRUE` in Shockwave.

In the authoring environment and in projectors, the default value is `FALSE`. This property may be tested, but it may only be set to `TRUE`. Once it has been set to `TRUE`, it cannot be set back to `FALSE` without restarting Director or the projector.

When `safePlayer` is `TRUE`, the following safety features are in effect:

- Only safe Xtras may be used.
- The `safePlayer` property cannot be reset.
- Pasting content from the Clipboard by using the `pasteClipBoardInto` command generates a warning dialog box that allows the user to cancel the operation.
- Handling Macintosh resource files by using the obsolete `openResFile` or `closeResFile` command is disabled.
- Saving a movie or cast by using Lingo is disabled.
- Printing by using the `printFrom` command is disabled.
- Opening an application by using the `open` command is disabled.
- The ability to stop an application or the user's computer by using the `restart` or `shutDown` command is disabled.
- Sending strings to Windows Media Control Interface (MCI) by using `mci` is disabled.
- Opening a file that is outside the DSWMedia folder is disabled.
- Discovering a local file name is disabled.
- Using `getNetText()` or `postNetText()`, or otherwise accessing a URL that does not have the same domain as the movie, generates a security dialog box.

sampleCount

Syntax

```
sound(channelNum).sampleCount
```

```
the sampleCount of sound(channelNum)
```

Description

Read-only property; the number of sound samples in the currently playing sound in sound channel *channelNum*. This is the total number of samples, and depends on the `sampleRate` and `duration` of the sound. It does not depend on the `channelCount` of the sound.

A 1-second, 44.1 KHz sound contains 44,100 samples.

Example

This statement displays the name and `sampleCount` of the cast member currently playing in sound channel 1 in the Message window.

```
put "Sound cast member" && sound(1).member.name && "contains" && \
sound(1).sampleCount && "samples."
```

See also

channelCount, **sampleRate**, **sampleSize**

sampleRate

Syntax

```
member(whichCastMember).sampleRate
the sampleRate of member whichCastMember
sound(channelNum).sampleRate
```

Description

Cast member property; returns, in samples per second, the sample rate of the sound cast member or in the case of SWA sound, the original file that has been Shockwave Audio–encoded. This property is available only after the SWA sound begins playing or after the file has been preloaded using the `preLoadBuffer` command. When a sound channel is given, the result is the sample rate of the currently playing sound cast member in the given sound channel.

This property can be tested but not set. Typical values are 8000, 11025, 16000, 22050, and 44100.

When multiple sounds are queued in a sound channel, Director plays them all with the `channelCount`, `sampleRate`, and `sampleSize` of the first sound queued, resampling the rest for smooth playback. Director resets these properties only after the channel's sound queue is exhausted or a stop() command is issued. The next sound to be queued or played then determines the new settings.

Example

This statement assigns the original sample rate of the file used in SWA streaming cast member Paul Robeson to the field cast member Sound Quality:

```
member("Sound Quality").text = string(member("Paul
Robeson").sampleRate)"
```

Example

This statement displays the sample rate of the sound playing in sound channel 1 in the Message window:

```
put sound(1).sampleRate
```

sampleSize

Syntax

```
member(whichCastMember).sampleSize
```

```
the sampleSize of member whichCastMember
```

sound(*channelNum*).sampleSize

Description

Cast member property; determines the sample size of the specified cast member. The result is usually a size of 8 or 16 bits. If a sound channel is given, the value if for the sound member currently playing in the given sound channel.

This property can be tested but not set.

Example

This statement checks the sample size of the sound cast member Voice Over and assigns the value to the variable `soundSize`:

```
soundSize = member("Voice Over").sampleSize
```

Example

This statement displays the sample size of the sound playing in sound channel 1 in the Message window:

```
put sound(1).sampleSize
```

save castLib

Syntax

```
castLib(whichCast).save()
castLib(whichCast).save(pathName & newFileName)
save castLib whichCast {,pathName&newFileName}
```

Description

Command; saves changes to the cast in the cast's original file or, if the optional parameter *pathName:newFileName* is included, in a new file. If no file name is given, the original cast must be linked. Further operations or references to the cast use the saved cast member.

This command does not work with compressed files.

The `save CastLib` command doesn't support URLs as file references.

Example

This statement causes Director to save the revised version of the Buttons cast in the new file UpdatedButtons in the same folder:

```
castLib("Buttons").save(the moviePath & "UpdatedButtons.cst")
```

See also

@ (pathname)

on savedLocal

Syntax

```
on savedLocal
  statement(s)
end
```

Description System message and event handler; This property is provided to allow for enhancements in future versions of Shockwave.

See also **allowSaveLocal**

saveMovie

Syntax `saveMovie {pathName&fileName}`

Description Command; saves the current movie. Including the optional parameter saves the movie to the file specified by *`pathName:fileName`*. This command does not work with compressed files. The specified filename must include the .dir file extension.

The `saveMovie` command doesn't support URLs as file references.

Example This statement saves the current movie in the Update file:

```
saveMovie the moviePath & "Update.dir"
```

See also **@ (pathname)**, **setPref**

scale

Syntax

```
member(whichCastMember).scale
the scale of member whichCastMember
sprite(whichSprite).scale
the scale of sprite whichSprite
```

Description

Cast member property and sprite property; controls the scaling of a QuickTime, vector shape, or Flash movie sprite.

For QuickTime, this property does not scale the sprite's bounding rectangle or the sprite's controller. Instead, it scales the image around the image's center point within the bounding rectangle. The scaling is specified as a Director list containing two percentages stored as float-point values:

```
[xPercent, yPercent]
```

The *xPercent* parameter specifies the amount of horizontal scaling; the *yPercent* parameter specifies vertical scaling.

When the sprite's `crop` property is set to `TRUE`, the `scale` property can be used to simulate zooming within the sprite's bounding rectangle. When the sprite's `crop` property is set to `FALSE`, the `scale` property is ignored.

This property can be tested and set. The default value is [1.0000,1.0000].

For Flash movie or vector shape cast members, the scale is a floating-point value. The movie is scaled from its origin point, as specified by its `originMode` property.

Note: This property must be set to the default value if the scaleMode property is set to #autoSize; otherwise the sprite does not display correctly.

Example

This sprite script keeps the playback head looping in the current frame while the QuickTime sprite in channel 5 is scaled down in 5% increments. When the sprite is no longer visible (because its horizontal scale value is 0% or less), the playback head continues to the next frame.

```
on exitFrame me
  scaleFactor = sprite(spriteNum).scale[1]
  currentMemberNum = sprite(spriteNum).memberNum
  if member(currentMemberNum).crop = FALSE then
    member(currentMemberNum).crop = TRUE
  end if
  if scaleFactor > 0 then
    scaleFactor = scaleFactor - 5
    sprite(spriteNum).scale = [scaleFactor, scaleFactor]
    go the frame
  end if
end
```

Example

This handler accepts a reference to a Flash movie sprite as a parameter, reduces the movie's scale to 0% (so it disappears), and then scales it up again in 5% increments until it is full size (100%) again.

```
on scaleMovie whichSprite
  sprite(whichSprite).scale = 0
  updatestage
  repeat with i = 1 to 20
```

```
    sprite(whichSprite).scale = i * 5
    updatestage
  end repeat
end
```

See also **scaleMode**, **originMode**

scaleMode

Syntax

```
sprite(whichVectorOrFlashSprite).scaleMode
the scaleMode of sprite whichVectorOrFlashSprite
member(whichVectorOrFlashMember).scaleMode
the scaleMode of member whichVectorOrFlashMember
```

Description Cast member property and sprite property; controls the way a Flash movie or vector shape is scaled within a sprite's bounding rectangle. When you scale a Flash movie sprite by setting its `scale` and `viewScale` properties, the sprite itself is not scaled; only the view of the movie within the sprite is scaled. The `scaleMode` property can have these values:

- `#showAll` (default for Director movies prior to version 7)— Maintains the aspect ratio of the original Flash movie cast member. If necessary, fill in any gap in the horizontal or vertical dimension using the background color.
- `#noBorder`—Maintains the aspect ratio of the original Flash movie cast member. If necessary, crop the horizontal or vertical dimension.
- `#exactFit`—Does not maintain the aspect ratio of the original Flash movie cast member. Stretch the Flash movie to fit the exact dimensions of the sprite.
- `#noScale`—preserves the original size of the Flash media, regardless of how the sprite is sized on the Stage. If the sprite is made smaller than the original Flash movie, the movie displayed in the sprite is cropped to fit the bounds of the sprite.
- `#autoSize` (default)—This specifies that the sprite rectangle is automatically sized and positioned to account for `rotation`, `skew`, `flipH`, and `flipV`. This means that when a Flash sprite is rotated, it will not crop as in earlier versions of Director. The `#autoSize` setting only functions properly when `scale`, `viewScale`, `originPoint`, and `viewPoint` are at their default values.

This property can be tested and set.

Example This sprite script checks the Stage color of the Director movie and, if the Stage color is indexed to position 0 in the current palette, the script sets the `scaleMode` property of a Flash movie sprite to `#showAll`. Otherwise, it sets the `scaleMode` property to `#noBorder`.

```
on beginsprite me
  if the stagecolor = 0 then
    sprite(me.spriteNum).scaleMode = #showAll
  else
    sprite(me.spriteNum).scaleMode = #noBorder
  end if
end
```

See also **scale**

score

Syntax

```
the score
```

Description

Movie property; determines which Score is associated with the current movie. This property can be useful for storing the current contents of the Score before wiping out and generating a new one or for assigning the current Score contents to a film loop.

This property can be tested and set.

Example

This statement assigns the film loop cast member Waterfall to the Score of the current movie:

```
the score = member("Waterfall").media
```

scoreColor

Syntax

```
sprite(whichSprite).scoreColor
the scoreColor of sprite whichSprite
```

Description

Sprite property; indicates the Score color assigned to the sprite specified by *whichSprite*. The possible values correspond to color chips 0 to 5 in the current palette.

This property can be tested and set. Setting this property is useful only during authoring and Score recording.

Example

This statement displays in the Message window the value for the Score color assigned to sprite 7:

```
put sprite(7).scorecolor
```

scoreSelection

Syntax `the scoreSelection`

Description Movie property; determines which channels are selected in the Score window. The information is formatted as a linear list of linear lists. Each contiguous selection is in a list format consisting of the starting channel number, ending channel number, starting frame number, and ending frame number. Specify sprite channels by their channel numbers; use the following numbers to specify the other channels.

To specify:	**Use:**
Frame script channel	0
Sound channel 1	-1
Sound channel 2	-2
Transition channel	-3
Palette channel	-4
Tempo channel	-5

Description

You can select discontinuous channels or frames.

This property can be tested and set.

Example This statement selects sprite channels 15 through 25 in frames 100 through 200:

```
set the scoreSelection = [[15, 25, 100, 200]]
```

Example This statement selects sprite channels 15 through 25 and 40 through 50 in frames 100 through 200:

```
set the scoreSelection = [[15, 25, 100, 200], [40, 50, 100, 200]]
```

Example This statement selects the frame script in frames 100 through 200:

```
set the scoreSelection = [[0, 0, 100, 200]]
```

script

Syntax `the script of menuItem whichItem of menu whichMenu`

`childObject.script`

the script of *childObject*

Description Menu item and child object property.

For menu items, determines which Lingo statement is executed when the specified menu item is selected. The *`whichItem`* expression can be either a menu item name or a menu item number; the *`whichMenu`* expression can be either a menu name or a menu number.

When a menu is installed, the script is set to the text following the "Å" character in the menu definition.

This property can be tested and set.

Note: Menus are not available in Shockwave.

For child objects, the return value is the name of the child's parent script. This property can be tested but not set.

To see an example of `script` used in a completed movie, see the Parent Scripts movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement makes `goHandler` the handler that is executed when the user chooses the Go command from the custom menu Control:

```
set the script of menuItem "Go" of menu "Control" to "goHandler"
```

Example This Lingo checks whether a child object is an instance of the parent script Warrior Ant:

```
if bugObject.script = script("Warrior Ant") then
  bugObject.attack()
end if
```

See also **checkMark**, **installMenu**, **menu**, handlers(), scriptText

scriptInstanceList

Syntax
```
sprite(whichSprite).scriptInstanceList
the scriptInstanceList of sprite whichSprite
```

Description Sprite property; creates a list of script references attached to a sprite. This property is available only during run time. The list is empty when the movie is not running. Modifications to the list are not saved in the Score. This property is useful for the following tasks:

- Attaching a behavior to a sprite for use during run time
- Determining if behaviors are attached to a sprite; determining what the behaviors are
- Finding a behavior script reference to use with the `sendSprite` command

This property can be tested and set. (It can be set only if the sprite already exists and has at least one instance of a behavior already attached to it.)

Example This handler displays the list of script references attached to a sprite:

```
on showScriptRefs spriteNum
  put sprite(spriteNum).scriptInstanceList
end
```

Example These statements attach the script Big Noise to sprite 5:

```
x = script("Big Noise").new()
sprite(5).scriptInstanceList.add(x)
```

See also **scriptNum**, **sendSprite**

scriptList

Syntax
```
sprite(whichSprite).scriptList
the scriptList of sprite whichSprite
```

Description Sprite property; returns the list of behaviors attached to the given sprite and their properties. This property may only be set by using `setScriptList()`. It may not be set during a score recording session.

Example This statement displays the list of scripts attached to sprite 1 in the Message window:

```
put sprite(1).scriptList
-- [[(member 2 of castLib 1), "[#myRotateAngle: 10.0000,
#myClockwise: 1, #myInitialAngle: 0.0000]"], [(member 3 of castLib
1), "[#myAnglePerFrame: 10.0000, #myTurnFrames: 10,
#myHShiftPerFrame: 10, #myShiftFrames: 10, #myTotalFrames: 60,
#mySurfaceHeight: 0]"]]
```

See also **setScriptList()**, **value()**

scriptNum

Syntax
```
sprite(whichSprite). scriptNum
scriptNum of sprite whichSprite
```

Description
Sprite property; indicates the number of the script attached to the sprite specified by *whichSprite*. If the sprite has multiple scripts attached, `scriptNum` sprite property returns the number of the first script. (To see a complete list of the scripts attached to a sprite, see the behaviors listed for that sprite in the Behavior Inspector.)

This property can be tested and set during Score recording.

Example
This statement displays the number of the script attached to sprite 4:

```
put sprite(4).scriptNum
```

See also
scriptInstanceList

scriptsEnabled

Syntax

```
member(whichCastMember).scriptsEnabled
the scriptsEnabled of member whichCastMember
```

Description

Director movie cast member property; determines whether scripts in a linked movie are enabled (`TRUE` or 1) or disabled (`FALSE` or 0).

This property is available for linked Director movie cast members only.

This property can be tested and set.

Example

This statement turns off scripts in the linked movie Jazz Chronicle:

```
member("Jazz Chronicle").scriptsEnabled = FALSE
```

scriptText

Syntax

```
member(whichCastMember).scriptText
the scriptText of member whichCastMember
```

Description

Cast member property; indicates the content of the script, if any, assigned to the cast member specified by *whichCastMember*.

The text of a script is removed when a movie is converted to a projector, protected, or compressed for Shockwave. Such movies then lose their values for the `scriptText` cast member property. Therefore, the movie's `scriptText` cast member property values can't be retrieved when the movie is played back by a projector. However, Director can set new values for `scriptText` cast member property inside the projector. These newly assigned scripts are automatically compiled so that they execute quickly.

This property can be tested and set.

Example

This statement makes the contents of field cast member 20 the script of cast member 30:

```
member(30).scriptText = member(20).text
```

scriptType

Syntax
```
member whichScript.scriptType
the scriptType of member whichScript
```

Description Cast member property; indicates the specified script's type. Possible values are `#movie`, `#score`, and `#parent`.

Example This statement makes the script member Main Script a movie script:

```
member("Main Script").scriptType = #movie
```

scrollByLine

Syntax
```
member(whichCastMember). scrollByLine(amount)
scrollByLine member whichCastMember, amount
```

Description Command; scrolls the specified field or text cast member up or down by the number of lines specified in *amount*. (Lines are defined as lines separated by carriage returns, not lines caused by wrapping.)

- When *amount* is positive, the field scrolls down.
- When *amount* is negative, the field scrolls up.

Example This statement scrolls the field cast member Today's News down five lines:

```
member("Today's News").scrollbyline(5)
```

Example This statement scrolls the field cast member Today's News up five lines:

```
scrollByLine member "Today's News", -1
```

scrollByPage

Syntax `member(whichCastMember). scrollByPage(amount)`

`scrollByPage member whichCastMember, amount`

Description Command; scrolls the specified field or text cast member up or down by the number of pages specified in *amount*. A page is equal to the number of lines of text visible on the screen.

- When *amount* is positive, the field scrolls down.
- When *amount* is negative, the field scrolls up.

The Director player for Java doesn't support the `scrollByPage` member property. Use the `scrollTop` member property to write Lingo that scrolls text.

Example This statement scrolls the field cast member Today's News down one page:

```
member("Today's News").scrollbypage(1)
```

Example This statement scrolls the field cast member Today's News up one page:

```
member("Today's News").scrollbypage(-1)
```

See also **scrollTop**

scrollTop

Syntax

```
member(whichCastMember).scrollTop
the scrollTop of member whichCastMember
```

Description

Cast member property; determines the distance, in pixels, from the top of a field cast member to the top of the field that is currently visible in the scrolling box. By changing the value for `scrollTop` member property while the movie plays, you can change the section of the field that appears in the scrolling field.

This is a way to make custom scrolling behaviors for text and field members.

For example, the following Lingo moves the field cast member Credits up or down within a field's box, depending on the value in the variable `sliderVal`:

```
global sliderVal
on prepareFrame
  newVal = sliderVal * 100
  member("Credits").scrolltop = newVal
end
```

The global variable `sliderVal` could measure how far the user drags a slider. The statement `set newVal = sliderVal * 100` multiplies `sliderVal` to give a value that is greater than the distance the user drags the slider. If `sliderVal` is positive, the text moves up; if `sliderVal` is negative, the text moves down.

Example

This repeat loop makes the field Credits scroll by continuously increasing the value of `scrollTop`:

```
on wa
  member("Credits").scrollTop = 1
  repeat with count = 1 to 150
    member("Credits").scrollTop = member("Credits").scrollTop + 1
    updateStage
  end repeat
end
```

searchCurrentFolder

Syntax `the searchCurrentFolder`

Description System property; determines whether Director searches the current folder when searching file names. This property is `TRUE` by default.

- When the `searchCurrentFolder` property is `TRUE` (1), Director searches the current folder when resolving file names.
- When the `searchCurrentFolder` property is `FALSE` (0), Director does not search the current folder when resolving file names.

This property can be tested and set.

Example This statement displays the status of the `searchCurrentFolder` property in the Message window:

```
put the searchCurrentFolder
```

The result is 1, which is the numeric equivalent of `TRUE`.

Example This statement sets the `searchCurrentFolder` property to `TRUE`, which causes Director to search the current folder when resolving file names:

```
the searchCurrentFolder = TRUE
```

See also **searchPaths**

searchPath

This is obsolete. Use **searchPaths**.

searchPaths

Syntax `the searchPaths`

Description System property; a list of paths that Director searches when trying to find linked media such as digital video, GIFs, bitmaps, or sound files. Each item in the list is a fully qualified pathname as it appears on the current platform at run time.

The value of `searchPaths` is a linear list that you can manipulate the same as any other list by using commands such as `add`, `addAt`, `append`, `deleteAt`, and `setAt`.

URLs should not be used as file references in the search paths.

Adding a large number of paths to `searchPaths` slows searching. Try to minimize the number of paths in the list.

This property can be tested and set, and is an empty list by default.

Note: This property will function on all subsequent movies after being set. Because the current movie's assets have already been loaded, changing the setting will not affect any of these assets.

Example This statement displays the paths that Director searches when resolving file names:

```
put the searchPaths
```

Example This statement assigns two folders to `searchPaths` in Windows. This version includes optional trailing backslashes.

```
set the searchPaths = ["c:\director\projects\", "d:
\cdrom\sources\"]
```

This statement is the same, except that trailing backslashes have been omitted:

```
set the searchPaths = ["c:\director\projects", "d:\cdrom\sources"]
```

Example This statement assigns two folders to `searchPaths` on a Macintosh. This version includes optional trailing colons.

```
set the searchPaths = ["hard drive:director:projects:",
"cdrom:sources:"]
```

This statement is the same, except that trailing colons have been omitted:

```
set the searchPaths = ["hard drive:director:projects",
"cdrom:sources"]
```

Example These statements cause Director to search in a folder named Sounds, which is in the same folder as the current Director movie:

```
set soundPaths = the moviePath & "Sounds"
add the searchPaths, soundPath
```

See also **searchCurrentFolder**, **@ (pathname)**

seconds

Syntax `dateObject.seconds`

Description Property; gives the seconds passed since midnight of the given date object. Only the `systemDate` , `creationDate`, and `modifiedDate` have a default `seconds` value. You must specify a `seconds` value for date objects that you create.

This property can be used with the `creationDate` and the `modifiedDate` for source control purposes.

Example These staments display the seconds since midnight on the authoring computer:

```
mydate = the systemdate
put mydate.seconds
    -- 1233
```

selectedText

Syntax `member(whichTextMember).selectedText`

Description Text cast member property; returns the currently selected chunk of text as a single object reference. This allows access to font characteristics as well as to the string information of the actual characters.

Example This handler displays the currently selected text being placed in a local variable object. Then that object is used to reference various characteristics of the text, which are detailed in the Message window.

```
property spriteNum
on mouseUp me
  mySelectionObject = sprite(spriteNum).member.selectedText
  put mySelectionObject.text
  put mySelectionObject.font
  put mySelectionObject.fontSize
  put mySelectionObject.fontStyle
end
```

selection() (function)

Syntax `the selection`

Description Function; returns a string containing the highlighted portion of the current editable field. This function is useful for testing what a user has selected in a field.

The `selection` function only indicates which string of characters is selected; you cannot use `selection` to select a string of characters.

Example This statement checks whether any characters are selected and, if none are, displays the alert “Please select a word.”:

```
if the selection = EMPTY then alert "Please select a word."
```

See also **selStart**, **selEnd**

selection (cast property)

Syntax

```
castLib (whichCast).selection

the selection of castLib whichCast

set the selection of castLib whichCast =[ [ startMember1 ,
endMember1 ], \
{[ startMember2 , endMember2 ], [ startMember3 , endMember3 ]...]}

castLib(whichCast). selection =[ [ startMember1 , endMember1 ], \
{[ startMember2 , endMember2 ], [ startMember3 , endMember3 ]...]}
```

Description

Cast property; determines which cast members are selected in the specified Cast window. The range appears as a list of the starting and ending cast member numbers for the selected range. You can include more than one selection by specifying additional ranges of cast members. (To specify more than one selection, Control-drag (Windows) or Command-drag (Macintosh).

This property can be tested and set.

Example

This statement selects cast members 1 through 10 in castLib 1:

```
castLib(1).selection = [[1, 10]]
```

This statement selects cast members 1 through 10, and 30 through 40, in castLib 1:

```
castLib(1).selection = [[1, 10], [30, 40]]
```

selection (text cast member property)

Syntax `member(whichTextMember).selection`

Description Text cast member property; returns a list of the first and last character selected in the text cast member.

This property can be tested and set.

Example The following statement sets the selection displayed by the sprite of text member myAnswer so that characters 6 through 10 are highlighted:

```
member("myAnswer").selection = [6, 10]
```

See also **color()**,**selStart**, **selEnd**

selEnd

Syntax `the selEnd`

Description Global property; specifies the last character of a selection. It is used with `selStart` to identify a selection in the current editable field, counting from the beginning character.

This property can be tested and set. The default value is 0.

Example These statements select “cde” from the field “abcdefg”:

```
the selStart = 3
the selEnd = 5
```

Example This statement calls the handler `noSelection` when `selEnd` is the same `as selStart`:

```
if the selEnd = the selStart then noSelection
```

Example This statement makes a selection 20 characters long:

```
the selEnd = the selStart + 20
```

See also **editable**, **hilite (command)**, **selection() (function)**, **selStart**, **text**

selStart

Syntax `the selStart`

Description Cast member property; specifies the starting character of a selection. It is used with `selEnd` to identify a selection in the current editable field, counting from the beginning character.

This property can be tested and set. The default value is 0.

Example These statements select “cde” from the field “abcdefg”:

```
the selStart = 3
the selEnd = 5
```

Example This statement calls the handler `noSelection` when `selEnd` is the same as `selStart`:

```
if the selEnd = the selStart then noSelection
```

Example This statement makes a selection 20 characters long:

```
the selEnd = the selStart + 20
```

See also **selection() (function)**, **selEnd**, **text**

sendAllSprites

Syntax `sendAllSprites (#customEvent, args)`

Description Command; sends a designated message to all sprites, not just the sprite that was involved in the event. As with any other message, the message is sent to every script attached to the sprite, unless the `stopEvent` command is used.

For best results, send the message only to those sprites that will properly handle the message through the `sendSprite` command. No error will occur if the message is sent to all the sprites, but performance may decrease. There may also be problems if different sprites have the same handler in a behavior, so avoid conflicts by using unique names for messages that will be broadcast.

After the message has been passed to all behaviors, the event follows the regular message hierarchy: cast member scripts, frame script, and then movie script.

When you use the `sendAllSprites` command, be sure to do the following:

- Replace *customEvent* with the message.
- Replace *args* with any arguments to be sent with the message.

If no sprite has an attached behavior containing the given handler, `sendAllSprites` returns `FALSE`.

Example This handler sends the custom message `allSpritesShouldBumpCounter` and the argument `2` to all sprites when the user clicks the mouse:

```
on mouseDown me
  sendAllSprites (#allSpritesShouldBumpCounter, 2)
end
```

See also **sendSprite**

sendSprite

Syntax `sendSprite (whichSprite, #customMessage, args)`

Description Command; sends a message to all scripts attached to a specified sprite.

Messages sent using `sendSprite` are sent to each of the scripts attached to the sprite. The messages then follow the regular message hierarchy: cast member script, frame script, and movie script.

If the given sprite does not have an attached behavior containing the given handler, `sendSprite` returns `FALSE`.

Example This handler sends the custom message `bumpCounter` and the argument `2` to sprite 1 when the user clicks:

```
on mouseDown me
  sendSprite (1, #bumpCounter, 2)
end
```

See also **sendAllSprites**

serialNumber

Syntax `the serialNumber`

Description Movie property; a string containing the serial number entered when Director was installed.

This property is available in the authoring environment only. It could be used in a movie in a window (MIAW) tool that is personalized to show the user's information.

Example This handler would be located in a movie script of a MIAW. It places the user's name and the serial number into a display field when the window is opened:

```
on prepareMovie
  displayString = the userName
  put RETURN&the organizationName after displayString
  put RETURN&the serialNumber after displayString
  member("User Info").text = displayString
end
```

See also **organizationName**, **userName**, **window**

set...to, set...=

Syntax
```
set the lingoProperty to expression
the lingoProperty = expression
set variable to expression
variable = expression
```

Description Command; evaluates an expression and puts the result in the property specified by *lingoProperty* or the variable specified by *variable*.

Example This statement sets the name of member 3 to Sunset:

```
set member(3).name = "Sunset"
```

Example This statement sets the `soundEnabled` property to the opposite of its current state. When `soundEnabled` is `TRUE` (the sound is on), this statement turns it off. When `soundEnabled` is `FALSE` (the sound is off), this statement turns it on.

```
set the soundEnabled = not (the soundEnabled)
```

Example This statement sets the variable `vowels` to the string “aeiou”:

```
set vowels = "aeiou"
```

See also **property**

setAlpha()

Syntax

```
imageObject.setAlpha(alphaLevel)
imageObject.setAlpha(alphaImageObject)
```

Description

Function; sets the alpha channel of an image object to a flat alphaLevel or to an existing *alphaImageObject*. The *alphaLevel* must be a number from 0–255. Lower values cause the image to appear more transparent. Higher values cause the image to appear more opaque. The value 255 has the same effect as a value of zero. In order for the *alphaLevel* to have effect, the `useAlpha()` of the image object must be set to `TRUE`.

The image object must be 32-bit. If you specify an alpha image object, it must be 8-bit. Both images must have the same dimensions. If these conditions are not met, `setAlpha()` has no effect and returns FALSE. The function returns TRUE when it is successful.

Example

This Lingo statement makes the image of the bitmap cast member Foreground opaque and disables the alpha channel altogether. This is a good method for removing the alpha layer from an image:

```
member("Foreground").image.setAlpha(255)
member("Foreground").image.useAlpha = FALSE
```

This Lingo gets the alpha layer from the cast member Sunrise and places it into the alpha layer of the cast member Sunset:

```
tempAlpha = member("Sunrise").image.extractAlpha()
member("Sunset").image.setAlpha(tempAlpha)
```

See also

useAlpha (),**extractAlpha()**

setaProp

Syntax
```
setaProp list, listProperty, newValue
setaProp (childObject, listProperty, newValue)
list.listProperty = newValue
list[listProperty] = newValue
childObject.listProperty = newValue
```

Description Command; replaces the value assigned to *listProperty* with the value specified by *newValue*. The `setaProp` command works with property lists and child objects. Using `setaProp` with a linear list produces a script error.

- For property lists, `setaProp` replaces a property in the list specified by *list*. When the property isn't already in the list, Lingo adds the new property and value.
- For child objects, `setaProp` replaces a property of the child object. When the property isn't already in the object, Lingo adds the new property and value.
- The `setaProp` command can also set `ancestor` properties.

Example These statements create a property list and then adds the item `#c:10` to the list:

```
newList = [#a:1, #b:5]
put newList
-- [#a:1, #b:5]
setaProp newList, #c, 10
put newList
```

Example Using the dot operator, you can alter the property value of a property already in a list without using `setaProp`:

```
newList = [#a:1, #b:5]
put newList
-- [#a:1, #b:5]
newList.b = 99
put newList
-- [#a:1, #b:99]
```

Note: To use the dot operator to manipulate a property, the property must already exist in the list, child object, or behavior.

See also **ancestor**, **property**, **. (dot operator)**

setAt

Syntax

```
setAt list, orderNumber, value
list[orderNumber] = value
```

Description

Command; replaces the item specified by *orderNumber* with the value specified by *value* in the list specified by *list*. When *orderNumber* is greater than the number of items in a property list, the `setAt` command returns a script error. When *orderNumber* is greater than the number of items in a linear list, Director expands the list's blank entries to provide the number of places specified by *orderNumber*.

Example

This handler assigns a name to the list [12, 34, 6, 7, 45], replaces the fourth item in the list with the value 10, and then displays the result in the Message window:

```
on enterFrame
  set vNumbers = [12, 34, 6, 7, 45]
  setAt vnumbers, 4, 10
  put vNumbers
end enterFrame
```

When the handler runs, the Message window displays the following:

```
 [12, 34, 6, 10, 45]
```

You can perform this same operation may be done using bracket access to the list in the following manner:

```
on enterFrame
  set vNumbers = [12, 34, 6, 7, 45]
  vnumbers[4] = 10
  put vNumbers
end enterFrame
```

When the handler runs, the Message window displays the following:

```
 [12, 34, 6, 10, 45]
```

See also

[ ] (bracket access)

setFlashProperty()

Syntax `sprite(spriteNum).setFlashProperty("targetName", #property, newValue)`

Description Function; allows Lingo to call the Flash action script function `setProperty()` on the given Flash sprite. Use the setFlashProperty() function to set the properties of movie clips or levels within a Flash movie. This is similar to setting sprite properties within Director.

The *`targetName`* is the name of the movie clip or level whose property you want to set within the given Flash sprite.

The *`#property`* is the name of the property to set. You can set the following movie clip properties: `#posX`, `#posY`, `#scaleX`, `#scaleY`, `#visible`, `#rotate`, `#alpha`, and `#name`.

To set a global property of the Flash sprite, pass an empty string as the *`targetName`*. You can set the global Flash properties: `#focusRect` and `#spriteSoundBufferTime`.

See the Flash documentation for descriptions of these properties.

This statement sets the value of the `#rotate` property of the movie clip Star in the Flash member in sprite 3 to 180.

```
sprite(3).setFlashProperty("Star", #rotate, 180)
```

See also **getFlashProperty()**

setPixel()

Syntax

```
imageObject.setPixel(x, y, colorObject)
imageObject.setPixel(point(x, y), colorObject)
imageObject.setPixel(x, y, integerValue)
imageObject.setPixel(point(x, y), integerValue)
```

Description

Function; sets the color value of the pixel at the specified point in the given image object, to either *colorObject* or to a raw integer with *integerValue*. If you are setting many pixels to the color of another pixel with `getPixel()`, it is faster to set them as raw integers.

For best performance with color objects, with 8-bit or lower images use an indexed color object, and with 16-bit or higher images, use an RGB color object.

The `SetPixe()` function returns `FALSE` if the specified pixel falls outside the specified image.

To see an example of `setPixel()` used in a completed movie, see the Imaging movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This Lingo statement draws a horizontal black line 50 pixels from left to right in cast member 5.

```
repeat with x = 1 to 50
  member(5).image.setPixel(x, 0, rgb(0, 0, 0))
end repeat
```

See also

getPixel(), **draw()**, **fill()**, **color()**

setPlaylist()

Syntax

```
sound(channelNum).setPlaylist([#member: member(whichmember),
{#startTime: milliseconds, #endTime: milliseconds, #loopCount:
numberOfLoops, #loopStartTime: milliseconds, #loopEndTime:
milliseconds, #preloadTime: milliseconds}]. . . )
```

```
setPlaylist(sound(channelNum), [#member: member(whichmember),
{#startTime: milliseconds, #endTime: milliseconds, #loopCount:
numberOfLoops, #loopStartTime: milliseconds, #loopEndTime:
milliseconds, #preloadTime: milliseconds}]. . . )
```

Description

Function; sets or resets the playlist of the given sound channel. This command is useful for queueing several sounds at once.

You can specifiy these parameters for each sound to be queued:

Property	Description
#member	The sound cast member to queue. This property must be provided; all others are optional.
#startTime	The time within the sound at which playback begins, in milliseconds. The default is the beginning of the sound. See startTime.
#endTime	The time within the sound at which playback ends, in milliseconds. The default is the end of the sound. See endTime.
#loopCount	The number of times to play a loop defined with #loopStartTime and #loopEndTime. The default is 1. See loopCount.
#loopStartTime	The time within the sound to begin a loop , in milliseconds. See loopStartTime.
#loopEndTime	The time within the sound to end a loop, in milliseconds. See loopEndTime.
#preloadTime	The amount of the sound to buffer before playback, in milliseconds. See preloadTime.

To see an example of `setPlayList()` used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This handler queues and plays the cast member introMusic, starting at its 3- second point, with a loop repeated 5 times from the 8-second point to the 8.9- second point, and stopping at the 10-second point.

```
on playMusic
  sound(2).queue([#member: member("introMusic"), #startTime:
3000,\
  #endTime: 10000, #loopCount: 5, #loopStartTime: 8000,
#loopEndTime: 8900])
  sound(2).play()
end
```

See also

endTime, **getPlaylist()**,**startTime**, **loopCount**, **loopEndTime**, **loopStartTime**, **member (sound property)**, **play() (sound)**, **preLoadTime**, **queue()**

setPref

Syntax `setPref prefName, prefValue`

Description Command; writes the string specified by *`prefValue`* in the file specified by *`prefName`* on the computer's local disk. The file is a standard text file.

The *`prefName`* argument must be a valid file name. To make sure the file name is valid on all platforms, use no more than eight alphanumeric characters for the file name.

After the `setPref` command runs, if the movie is playing in a browser, a folder named Prefs is created in the Plug-In Support folder. The `setPref` command can write only to that folder.

If the movie is playing in a projector or Director, a folder is created in the same folder as the application. The folder receives the name *Prefs*.

Do not use this command to write to read-only media. Depending on the platform and version of the operating system, you may encounter errors or other problems.

This command does not perform any sophisticated manipulation of the string data or its formatting. You must perform any formatting or other manipulation in conjunction with `getPref()`; you can manipulate the data in memory and write it over the old file using `setPref`.

In a browser, data written by `setPref` is not private; any Shockwave movie can read this information and upload it to a server. Do not store confidential information using `setPref`.

To see an example of `setPref` used in a completed movie, see the Read and Write Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This handler saves the contents of the field cast member Text Entry in a file named DayWare settings:

```
on mouseUp me
  setPref "CurPrefs", member("Text Entry").text
end
```

See also **getPref()**

setProp

Syntax

```
setProp list, property, newValue
list.listProperty = newValue
list[listProperty] = newValue
```

Description

Command; replaces the value assigned to *property* with the value specified by *newValue* in the list specified by *list*. If the list doesn't contain the specified property, `setProp` returns a script error.

The `setProp` command works with property lists only. Using `setProp` with a linear list produces a script error.

This command is similar to the `setaProp` command, except that `setProp` returns an error when the property is not already in the list.

Example

This statement changes the value assigned to the age property of property list x to 11:

```
setProp x, #age, 11
```

Example

Using the dot operator, you can alter the property value of a property already in a list, exactly as above:

```
x.age = 11
```

See also

setaProp

setScriptList()

Syntax `sprite(whichSprite).setScriptList(scriptList)`

Description This command sets the `scriptList` of the given sprite. The `scriptList` indicates which scripts are attached to the sprite and what the settings of each script property are. By setting this list, you can change which behaviors are attached to a sprite or change the behavior properties.

The list takes the form:

```
[ [ (whichBehaviorMember), " [ #property1: value, #property2:
value, . . . ] ",
[(whichBehaviorMember), " [ #property1: value, #property2:
value, . . . ] " ] ]
```

This command cannot be used during a score recording session. Use `setScriptList()` for sprites added during score recording after the score recording session has ended.

See also **scriptList**, **value()**,**string()**

setTrackEnabled

Syntax
```
sprite(whichSprite).setTrackEnabled(whichTrack, trueOrFalse )
setTrackEnabled(sprite whichSprite, whichTrack, trueOrFalse)
```

Description Command; determines whether the specified track in the digital video is enabled to play.

- When `setTrackEnabled` is `TRUE`, the specified track is enabled and playing.
- When `setTrackEnabled` is `FALSE`, the specified track is disabled and muted. For video tracks, this means they will no longer be updated on the screen.

To test whether a track is already enabled, test the `trackEnabled` sprite property.

Example This statement enables track 3 of the digital video assigned to sprite channel 8:

```
sprite(8).setTrackEnabled(3, TRUE)
```

See also **trackEnabled**

setVariable()

Syntax `setVariable(sprite flashSpriteNum, "variableName", newValue)`

Function; sets the value of the given variable in the given Flash sprite. Flash variables were introduced in Flash version 4.

This statement sets the value of the variable currentURL in the Flash cast member in sprite 3. The new value of currentURL will be “http://www.macromedia.com/software/flash/”.

```
setVariable(sprite 3, "currentURL",
"http://www.macromedia.com/software/flash/")
```

See also **hitTest()**,**getVariable()**

shapeType

Syntax

```
member(whichCastMember).shapeType
```

```
the shapeType of member whichCastMember
```

Description

Shape cast member property; indicates the specified shape's type. Possible types are `#rect`, `#roundRect`, `#oval`, and `#line`. You can use this property to specify a shape cast member's type after creating the shape cast member using Lingo.

Example

These statements create a new shape cast member numbered 100 and then define it as an oval:

```
new(#shape, member 100)
member(100).shapeType = #oval
```

shiftDown

Syntax `the shiftDown`

Description System property; indicates whether the user is pressing the Shift key (`TRUE`) or not (`FALSE`).

In the Director player for Java, this function returns `TRUE` only if the Shift key and another key are pressed simultaneously. If the Shift key is pressed by itself, `shiftDown` returns `FALSE`.

The Director player for Java supports key combinations with the Shift key. However, the browser receives the keys before the movie and thus responds to and intercepts any key combinations that are also browser keyboard shortcuts.

This property must be tested in conjunction with another key.

Example This statement checks whether the Shift key is being pressed and calls the handler `doCapitalA` if it is:

```
if (the shiftDown) then doCapitalA (the key)
```

See also **commandDown**, **controlDown**, **key()**,**optionDown**

short

See **date() (system clock)**, **time()**

showGlobals

Syntax `showGlobals`

Description Command; displays all global variables in the Message window. This command is useful for debugging scripts.

Example This statement displays all global variables in the Message window:

```
showGlobals
```

See also **clearGlobals**, **showLocals**, **global**, **globals**

showLocals

Syntax `showLocals`

Description Command; displays all global variables in the Message window. This command is useful only within handlers or parent scripts that contain local variables to display. All variables used in the Message window are automatically global.

Local variables in a handler are no longer available after the handler executes. Inserting the statement

```
showLocals
```

in a handler displays all the local variables in that handler in the Message window.

This command is useful for debugging scripts.

See also **clearGlobals**, **showGlobals**, **global**

showProps()

Syntax

```
member(whichFlashOrVectorCastMember). showProps
member(whichFlashOrVectorCastMember).showProps()
sprite(whichFlashOrVectorSprite).showProps()
sound(channelNum).showProps()
```

Description

Command; displays a list of the current property settings of a Flash movie, Vector member, or currently playing sound in the Message window. This command is useful for authoring only; it does not work in projectors or in Shockwave movies.

Example

This handler accepts the name of a cast as a parameter, searches that cast for Flash movie cast members, and displays the cast member name, number, and properties in the Message window:

```
on ShowCastProperties whichCast
  repeat with i = 1 to the number of members of castLib whichCast
    castType = member(i, whichCast).type
    if (castType = #flash) OR (castType = #vectorShape) then
      put castType&&"cast member" && i & ":" && member(i,
whichCast).name
      put RETURN
      member(i ,whichCast).showProps()
    end if
  end repeat
end
```

See also

queue(), **setPlaylist()**

showResFile

Description This Lingo is obsolete.

showXlib

Syntax `showXlib {Xlibfilename}`

Description Command; shows all Xtras in *`Xlibfilename`* (which must be open), or all open Xlibraries if no file is specified. Xlibrary files are resource files that contain DLLs (Windows) or Xtra resources (Macintosh). Because the types of Xlibrary files in Windows and on the Macintosh differ, the list of files that the `showXlib` command generates is different on different platforms.

The `showXlib` command doesn't support URLs as file references.

You can use `interface()` to display online documentation for an Xtra.

Note: This command is not supported in Shockwave.

Example This statement displays the Xtras in the VideoDisc Xlibrary:

```
showXlib "VideoDisc Xlibrary"
```

See also **closeXlib**, **interface()**,**openXlib**

shutDown

Syntax `shutDown`

Description Command; closes all open applications and turns off the computer.

Example This statement checks whether the user has pressed Control+S (Windows) or Command+S (Macintosh) and, if so, shuts down the computer:

```
if the key = "s" and the commandDown then
  shutDown
end if
```

See also **quit**, **restart**

sin()

Syntax `sin(`*`angle`*`)`

Description Math function; calculates the sine of the specified angle. The angle must be expressed in radians as a floating-point number.

Example The following statement calculates the sine of `pi`/2:

```
put sin (PI/2.0)
-- 1
```

See also **PI**

size

Syntax
```
member(whichCastMember).size
the size of member whichCastMember
```

Description Cast member property; returns the size in memory, in bytes, of a specific cast member number or name. Divide bytes by 1024 to convert to kilobytes.

Example The following line outputs the size of the cast member Shrine in a field named How Big:

```
member("How Big").text = string(member("shrine").size)
```

skew

Syntax `sprite(whichSpriteNumber).skew`

Description Sprite property; returns, as a float value in hundredths of a degree, the angle to which the vertical edges of the sprite are tilted (skewed) from the vertical. Negative values indicate a skew to the left; positive values indicate a skew to the right. Values greater than 90° flip an image vertically.

The Score can retain information for skewing an image from +21,474,836.47° to -21,474,836.48°, allowing 59,652 full rotations in either direction.

When the skew limit is reached (slightly past the 59,652th rotation), the skew resets to +116.47° or -116.48°—not 0.00°. This is because +21,474,836.47° is equal to +116.47°, and -21,474,836.48° is equal to -116.48° (or +243.12°). To avoid this reset condition, constrain angles to ±360° in either direction when using Lingo to perform continuous skewing.

Example The following behavior causes a sprite to skew continuously by 2° every time the playback head advances, while limiting the angle to 360°:

```
property spriteNum
on prepareFrame me
  sprite(spriteNum).skew = integer(sprite(spriteNum).skew + 2)
mod 360
end
```

See also **flipH**, **flipV**, **rotation**

sort

Syntax
```
list.sort()
sort list
```

Description Command; puts list items into alphanumeric order.

- When the list is a linear list, the list is sorted by values.
- When the list is a property list, the list is sorted alphabetically by properties.

  After a list is sorted, it maintains its sort order even when you add new variables using the `add` command.

Example This statement puts the list Values, which consists of [`#a: 1, #d: 2, #c: 3`], into alphanumeric order. The result appears below the statement.

```
put values
-- [#a: 1, #d: 2, #c: 3]
values.sort()
put values
--[#a: 1, #c: 3, #d: 2]
```

sound

Syntax

```
member(whichCastMember).sound
```

```
the sound of member whichCastMember
```

Description

Cast member property; controls whether a movie, digital video, or Flash movie's sound is enabled (`TRUE`, default) or disabled (`FALSE`). In flash members, the new setting takes effect after the currently playing sound ends.

This property can be tested and set.

To see an example of `sound` used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This handler accepts a member reference and toggles the member's `sound` property on or off:

```
on ToggleSound whichMember
  member(whichMember).sound = not member(whichMember).sound
end
```

soundBusy()

Syntax `soundBusy(whichChannel)`

Description Function; determines whether a sound is playing (`TRUE`) or not playing (`FALSE`) in the sound channel specified by *`whichChannel`*.

Make sure that the playback head has moved before using `soundBusy()` to check the sound channel. If this function continues to return `FALSE` after a sound should be playing, add the `updateStage` command to start playing the sound before the playback head moves again.

This function works for those sound channels occupied by actual audio cast members. QuickTime, Flash, and Shockwave audio handle sound differently, and this function will not work with those media types.

Example This statement checks whether a sound is playing in sound channel 1 and loops in the frame if it is. This allows the sound to finish before the playback head goes to another frame.

```
if soundBusy(1) then go to the frame
```

See also **sound playFile**, **sound stop**

soundChannel

Syntax `member(whichCastMember).soundChannel`

the soundChannel of member *whichCastMember*

Description Shockwave Audio (SWA) cast member property; specifies the sound channel in which the SWA sound plays.

If no channel number or channel 0 is specified, the SWA streaming cast member assigns the sound to the highest numbered sound channel that is unused.

Shockwave Audio streaming sounds can appear as sprites in sprite channels, but they play sound in a sound channel. Refer to SWA sound sprites by their sprite channel number, not their sound channel number.

This property can be tested and set.

Example This statement tells the SWA streaming cast member Frank Zappa to play in sound channel 3:

```
member("Frank Zappa").soundChannel = 3
```

sound close

This is obsolete. Use **puppetSound** instead.

soundDevice

Syntax `the soundDevice`

Description System property; allows the sound mixing device to be set while the movie plays. The possible settings are the devices listed in `soundDeviceList`.

Several sound devices can be referenced. The various sound devices for Windows have different advantages.

- DirectSound (Windows) —A sound device Xtra that permits low-latency audio playback performance on Windows and ensures reliable playing, mixing, and switching between different audio sources such as Director, AVI, and QuickTime sound. DirectSound is the default value for `soundDevice`.

  This Xtra requires DirectSound 5 or later from Microsoft. The DirectSound Xtra (Directsound.x32) and Microsoft's DirectSound must both be available in order to work. Windows NT 4 does not support DirectSound.

- MacroMix (Windows)—The lowest common denominator for Windows playback. This device functions on any Windows computer, but its latency is not as good as that of other devices.
- QT3Mix (Windows)—Mixes sound with QuickTime audio and possibly with other applications if they use DirectSound. This device requires that QuickTime be installed and has better latency than MacroMix.
- MacSoundManager (Macintosh)—The only sound device available on the Macintosh.

Example This statement sets the sound device to the MacroMix for a Windows computer. If the newly assigned device fails, the `soundDevice` property is not changed.

```
set the soundDevice = "MacroMix"
```

See also **soundDeviceList**

soundDeviceList

Syntax `the soundDeviceList`

Description System property; creates a linear list of sound devices available on the current computer.

For the Macintosh, this property lists one device, MacSoundManager.

This property can be tested but not set.

Example This statement displays a typical sound device list on a Windows computer:

```
Put the soundDeviceList
--["QT3Mix", "MacroMix", "DirectSound"]
```

See also **soundDevice**

soundEnabled

Syntax `the soundEnabled`

Description System property; determines whether the sound is on (`TRUE`, default) or off (`FALSE`).

When you set this property to `FALSE`, the sound is turned off, but the volume setting is not changed.

This property can be tested and set.

Example This statement sets `soundEnabled` to the opposite of its current setting; it turns the sound on if it is off and turns it off if it is on:

```
the soundEnabled = not(the soundEnabled)
```

See also **soundLevel**, **volume (sound channel)**, **volume (sprite property)**

sound fadeIn

Syntax
```
sound(whichChannel).fadeIn()
sound fadeIn whichChannel
sound(whichChannel).fadeIn(ticks)
sound fadeIn whichChannel, ticks
```

Description Command; fades in a sound in the specified sound channel over a period of frames or ticks.

- When *ticks* is specified, the fade in occurs evenly over that period of time.
- When *ticks* is not specified, the default number of ticks is calculated as 15 * (60 / (tempo setting)) based on the tempo setting for the first frame of the fade in. The fade in continues at a predetermined rate until the number of ticks has elapsed, or until the sound in the specified channel changes.

Example This statement fades in the sound in channel 1 over 5 seconds:

```
sound(1).fadeIn(5 * 60)
```

See also **sound fadeOut**, **fadeTo()**

sound fadeOut

Syntax
```
sound(whichChannel).fadeOut()
sound fadeOut whichChannel
sound(whichChannel).fadeOut(ticks)
sound fadeOut whichChannel, ticks
```

Description Command; fades out a sound in the specified sound channel over a period of frames or ticks.

- When *ticks* is specified, the fade out occurs evenly over that period of time.
- When *ticks* is not specified, the default number of ticks is calculated as 15 * (60 / (tempo setting)) based on the tempo setting for the first frame of the fade out. The fade out continues at a predetermined rate until the number of ticks has elapsed, or until the sound in the specified channel changes.

  If the sound is stopped before it reaches the minimum volume, it remains at the level it was stopped at, causing subsequent playback to be at this volume. Be sure to allow the sound to finish fading completely.

  Note: You may want to use the `volume` sound property to create a custom sound fade to allow more control over the actual volume of the channel.

Example This statement fades out the sound in channel 1 over 5 seconds:

```
sound(1).fadeOut(5 * 60)
```

See also **puppetSound**, **sound fadeIn**, **volume (sprite property)**, fadeTo()

soundKeepDevice

Syntax `the soundKeepDevice`

Description System property; for Windows only, prevents the sound driver from unloading and reloading each time a sound needs to play. The default value is `TRUE`.

You may need to set this property to `FALSE` before playing a sound to ensure that the sound device is unloaded and made available to other applications or processes on the computer after the sound has finished.

Setting this property to `FALSE` may adversely affect performance if sound playback is used frequently throughout the Director application.

This property can be tested and set.

Example This statement sets the `soundKeepDevice` property to `FALSE`:

```
set the soundKeepDevice = FALSE
```

soundLevel

Syntax `the soundLevel`

Description System property; sets the volume level of the sound played through the computer's speaker. Possible values range from 0 (no sound) to 7 (the maximum, default).

In Windows, the system sound setting combines with the volume control of the external speakers. Thus, the actual volume that results from setting the sound level can vary. Avoid setting the `soundLevel` property unless you are sure that the result is acceptable to the user. It is better to set the individual volumes of the channels and sprites with `volume of sound`, `volume of member`, and `volume of sprite`.

These values correspond to the settings in the Macintosh Sound control panel. Using this property, Lingo can change the sound volume directly or perform some other action when the sound is at a specified level.

This property can be tested and set.

To see an example of `soundLevel` used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement sets the variable `oldSound` equal to the current sound level:

```
oldSound = the soundLevel
```

Example This statement sets the sound level to 5:

```
the soundLevel = 5
```

See also **soundEnabled**, **volume (sound channel)**

the soundMixMedia

Syntax `the soundMixMedia`

Description This global property enables Flash cast members to mix their sound with sounds in the score sound channels when it is set to `TRUE`. When it is set to `FALSE`, these sounds will not be mixed and must be played at separate times. This property defaults to TRUE for movies made with Director 7 and later and false for earlier ones.

This property is valid only on Windows. When `the soundMixMedia` is `TRUE`, Director takes over the mixing and playback of sounds from Flash cast members. It is possible that slight differences may occur in the way Flash sounds play back. To hear the Flash sounds exactly they would be rendered in Flash, set this property to `FALSE`.

sound playFile

Syntax `sound playFile whichChannel, whichFile`

Description Command; plays the AIFF, SWA, AU, or WAV sound located at *`whichFile`* in the sound channel specified by *`whichChannel`*. For the sound to be played properly, the correct MIX Xtra must be available to the movie, usually in the Xtras folder of the application.

When the sound file is in a different folder than the movie, *`whichFile`* must specify the full path to the file.

To play sounds obtained from a URL, it's usually a good idea to use `downloadNetThing` or `preloadNetThing()` to download the file to a local disk first. This approach can minimize problems that may occur while the file is downloading.

The `sound playFile` command streams files from disk rather than playing them from RAM. As a result, using the `sound playFile` command when playing digital video or when loading cast members into memory can cause conflicts when the computer tries to read the disk in two places at once.

Example This statement plays the file named Thunder in channel 1:

```
sound playFile 1, "Thunder.wav"
```

Example This statement plays the file named Thunder in channel 3:

```
sound playFile 3, the moviePath & "Thunder.wav"
```

See also **sound stop**

sound stop

Syntax
```
sound(whichChannel).stop()
```
```
sound stop whichChannel
```

Description Command; stops the sound playing in the specified channel.

The `sound stop` command was used in earlier versions of Director. For best results, use the `puppetSound` command instead.

Example These statements stop any sound playing in sound channel 1:

```
sound(1).stop()
```

Example This statement checks whether a sound is playing in sound channel 1 and, if it is, stops the sound:

```
if soundBusy(1) then sound(1).stop()
```

See also **puppetSound**, **soundBusy()**

sourceRect

Syntax
```
window whichWindow.sourceRect
the sourceRect of window whichWindow
```

Description
Window property; specifies the original Stage coordinates of the movie playing in the window specified by *whichWindow*.

This property is useful for returning a window to its original size and position after it has been dragged or its rectangle has been set.

Example
This statement displays the original coordinates of the Stage named Control Panel in the Message window:

```
put window("Control Panel").sourceRect
```

See also
drawRect, **rect()**

SPACE

Syntax `SPACE`

Description Constant; read-only, value that represents the space character.

Example This statement displays “Age Of Aquarius” in the Message window:

```
put "Age"&SPACE&"Of"&SPACE&"Aquarius"
```

sprite

Syntax
```
sprite(whichSprite).property
the property of sprite whichSprite
```

Description
Keyword; tells Lingo that the value specified by *whichSprite* is a sprite channel number. It is used with every sprite property.

A sprite is an occurrence of a cast member in a sprite channel of the Score.

This term has special meaning in the Director player for Java. Don't use the term `sprite` in Java code that you embed within a Lingo script.

Example
This statement sets the variable named `horizontal` to `the locH of sprite 1`:

```
horizontal = sprite(1).loc
```

Example
This statement displays the current member in sprite channel 100 in the Message window:

```
put sprite (100).member
```

See also
puppetSprite, **spriteNum**

sprite...intersects

Syntax

```
sprite(sprite1).intersects(sprite2)
sprite sprite1 intersects sprite2
```

Description

Keyword; operator that compares the position of two sprites to determine whether the quad of *sprite1* touches (TRUE) or does not touch (FALSE) the quad of *sprite2*.

If both sprites have matte ink, their actual outlines, not the quads, are used. A sprite's outline is defined by the nonwhite pixels that make up its border.

This is a comparison operator with a precedence level of 5.

Note: The dot operator is required whenever sprite1 is not a simple expression—that is, one that contains a math operation.

Example

This statement checks whether two sprites intersect and, if they do, changes the contents of the field cast member Notice to "You placed it correctly."

```
if sprite i intersects j then put "You placed it correctly." into
member "Notice"
```

See also

sprite...within, **quad**

sprite...within

Syntax
```
sprite(sprite1).within(sprite2)
sprite sprite1 within sprite2
```

Description
Keyword; operator that compares the position of two sprites and determines whether the quad of *sprite1* is entirely inside the quad of *sprite2* (TRUE) or not (FALSE).

If both sprites have matte ink, their actual outlines, not the quads, are used. A sprite's outline is defined by the nonwhite pixels that make up its border.

This is a comparison operator with a precedence level of 5.

Note: The dot operator is required whenever sprite1 is not a simple expression—that is, one that contains a math operation.

Example
This statement checks whether two sprites intersect and calls the handler `doInside` if they do:

```
if sprite(3).within(2) then doInside
```

See also
sprite...intersects, **quad**

spriteNum

Syntax

```
spriteNum
```

```
the spriteNum of me
```

Description

Sprite property; determines the channel number the behavior's sprite is in and makes it available to any behaviors. Simply declare the property at the top of the behavior, along with any other properties the behavior may use.

If you use an `on new` handler to create an instance of the behavior, the script's `on new` handler must explicitly set the `spriteNum` property to the sprite's number. This provides a way to identify the sprite the script is attached to. The sprite's number must be passed to the `on new` handler as an argument when the `on new` handler is called.

Example

In this handler, the `spriteNum` property is automatically set for script instances that are created by the system:

```
property spriteNum
on mouseDown me
  sprite(spriteNum).member = member("DownPict")
end
```

Example

This handler uses the automatic value inserted into the `spriteNum` property to assign the sprite reference to a new property variable `pMySpriteRef`, as a convenience:

```
property spriteNum, pMySpriteRef
on beginSprite me
  pMySpriteRef = sprite(me.spriteNum)
end
```

This approach allows the use of the reference `pMySpriteRef` later in the script, with the handler using the syntax

```
currMember = pMySpriteRef.member
```

instead of the following syntax which is somewhat longer:

```
currMember = sprite(spriteNum).member
```

This alternative approach is merely for convenience, and provides no different functionality.

See also

on beginSprite, **on endSprite**, **currentSpriteNum**

sqrt()

Syntax
```
sqrt(number)
the sqrt of number
```

Description Math function; returns the square root of the number specified by *number*, which is either a floating-point number or an integer rounded to the nearest integer.

The value must be a decimal number greater than 0. Negative values return 0.

Example This statement displays the square root of 3.0 in the Message window:

```
put sqrt(3.0)
-- 1.7321
```

Example This statement displays the square root of 3 in the Message window:

```
put sqrt(3)
-- 2
```

See also **floatPrecision**

stage

Syntax `the stage`

Description System property; refers to the main movie.

This property is useful when using the `tell` command to send a message to the main movie from a child movie.

Example This statement causes the main Stage movie to go to the marker named Menu. This statement can be used in a movie in a window (MIAW):

```
tell the Stage to go to "Menu"
```

Example This statement displays the current setting for the Stage:

put the stage.rect

```
--rect (0, 0, 640, 480)
```

stageBottom

Syntax `the stageBottom`

Description Function; along with `stageLeft`, `stageRight`, and `stageTop`, indicates where the Stage is positioned on the desktop. It returns the bottom vertical coordinate of the Stage relative to the upper left corner of the main screen. The height of the Stage in pixels is determined by `the stageBottom - the stageTop`.

When the movie plays back as an applet, the `stageBottom` property is the height of the applet in pixels.

This function can be tested but not set.

Example These statements position sprite 3 a distance of 50 pixels from the bottom edge of the Stage:

```
stageHeight = the stageBottom - the stageTop
sprite(3).locV = stageHeight - 50
```

Sprite coordinates are expressed relative to the upper left corner of the Stage. See *Using Director* for more information.

See also **stageLeft**, **stageRight**, **stageTop**, **locH**, **locV**

stageColor

Syntax `the stageColor`

Description System property; determines the color of the movie background for index color only.

Use **bgColor** for more accurate, reliable, and flexible stage color specification with RGB values.

The value of the `stageColor` property ranges from 0 to 255 for 8-bit index color, and from 0 to 15 for 4-bit color. You can click a color in the color palette to see that color's index number in the lower left corner of the window. Setting the `stageColor` property in a Lingo script is equivalent to choosing the Stage color from the pop-up palette in the panel window.

Note: For compatibility when playing back as a Java applet, use the bgColor property to define the color as an RGB value.

Example This statement sets the variable `oldColor` to the index number of the current Stage color:

Related Lingo
```
oldColor = the stageColor
```

This statement sets the Stage color to the color assigned to chip 249 on the current palette:

Related Functions
```
the stageColor = 249
```

See also **backColor**, **bgColor**, **foreColor**, **color()**

stageLeft

Syntax `the stageLeft`

Description Function; along with `stageRight`, `stageTop`, and `stageBottom`, indicates where the Stage is positioned on the desktop. It returns the left horizontal coordinate of the Stage relative to the upper left corner of the main screen. When the Stage is flush with the left side of the main screen, this coordinate is 0.

When the movie plays back as an applet, the `stageLeft` property is 0, which is the location of the left side of the applet.

This property can be tested but not set.

Sprite coordinates are expressed relative to the upper left corner of the Stage.

Example This statement checks whether the left edge of the Stage is beyond the left edge of the screen and calls the handler `leftMonitorProcedure` if it is:

```
if the stageLeft < 0 then leftMonitorProcedure
```

See also **stageBottom**, **stageRight**, **stageTop**, **locH**, **locV**

stageRight

Syntax `the stageRight`

Description Function; along with `stageLeft`, `stageTop`, and `stageBottom`, indicates where the Stage is positioned on the desktop. It returns the right horizontal coordinate of the Stage relative to the upper left corner of the main screen's desktop. The width of the Stage in pixels is determined by `the stageRight - the stageLeft`.

When the movie plays back as an applet, the `stageRight` property is the width of the applet in pixels.

This function can be tested but not set.

Sprite coordinates are expressed relative to the upper left corner of the Stage.

Example These two statements position sprite 3 a distance of 50 pixels from the right edge of the Stage:

```
stageWidth = the stageRight - the stageLeft
  sprite(3).locH = stageWidth - 50
```

See also **stageBottom**, **stageLeft**, **stageTop**, **locH**, **locV**

stageToFlash()

Syntax

```
sprite(whichFlashSprite).stageToFlash(pointOnDirecto rStage)
stageToFlash (sprite whichFlashSprite, pointOnDirectorStage)
```

Description

Function; returns the coordinate in a Flash movie sprite that corresponds to a specified coordinate on the Director Stage. The function both accepts the Director Stage coordinate and returns the Flash movie coordinate as Director point values: for example, point (300,300).

Flash movie coordinates are measured in Flash movie pixels, which are determined by the original size of the movie when it was created in Flash. Point (0,0) of a Flash movie is always at its upper left corner. (The cast member's `originPoint` property is not used to calculate movie coordinates; it is used only for rotation and scaling.)

The `stageToFlash()` function and the corresponding `flashToStage()` function are helpful for determining which Flash movie coordinate is directly over a Director Stage coordinate. For both Flash and Director, point (0,0) is the upper left corner of the Flash Stage or Director Stage. These coordinates may not match on the Director Stage if a Flash sprite is stretched, scaled, or rotated.

Example

This handler checks to see if the mouse (whose location is tracked in Director Stage coordinates) is over a specific coordinate (130,10) in a Flash movie sprite in channel 5. If the mouse is over that Flash movie coordinate, the script stops the Flash movie.

```
on checkFlashRollover
  if sprite(5).stageToFlash(point(the mouseH,the mouseV)) =
point(130,10) then
    sprite(5).stop()
  end if
end
```

See also

flashToStage()

stageTop

Syntax `the stageTop`

Description Function; along with `stageBottom`, `stageLeft`, and `stageRight`, indicates where the Stage is positioned on the desktop. It returns the top vertical coordinate of the Stage relative to the upper left corner of the main screen's desktop. If the Stage is in the upper left corner of the main screen, this coordinate is 0.

When the movie plays back as an applet, the `stageTop` property is always 0, which is the location of the left side of the applet.

This function can be tested but not set.

Sprite coordinates are expressed relative to the upper left corner of the Stage.

Example This statement checks whether the top of the Stage is beyond the top of the screen and calls the handler `upperMonitorProcedure` if it is:

```
if the stageTop < 0 then upperMonitorProcedure
```

See also **stageBottom**, **stageLeft**, **stageRight**, **locH**, **locV**

startFrame

Syntax `sprite(whichSprite).startFrame`

Description Function; returns the frame number of the starting frame of the sprite span.

This function is useful in determining the span in the Score that a particular sprite covers. It is available only in a frame that contains the sprite, and cannot be applied to sprites in different frames of the movie, nor is it possible to set this value.

Example This statement displays the starting frame of the sprite in channel 5 in the Message window:

```
put sprite(5).startFrame
```

See also **endFrame()**

on startMovie

Syntax

```
on startMovie
  statement(s)
end
```

Description

System message and event handler; contains statements that run just before the playback head enters the first frame of the movie. The `startMovie` event occurs after the `prepareFrame` event and before the `enterFrame` event.

An `on startMovie` handler is a good place to put Lingo that initializes sprites in the first frame of the movie.

Example

This handler makes sprites invisible when the movie starts:

```
on startMovie
  repeat with counter = 10 to 50
    sprite(counter).visible = 0
  end repeat
end startMovie
```

See also

on prepareMovie

starts

Syntax `string1 starts string2`

Description Operator; compares to determines whether *`string1`* starts with *`string2`* (`TRUE` or 1) or not (`FALSE` or 0).

The string comparison is not sensitive to case or diacritical marks; *a* and *Å* are considered to be the same.

This is a comparison operator with a precedence level of 1.

Example This statement reports in the Message window whether the word *Macromedia* starts with the string “Macro”:

```
put "Macromedia" starts "Macro"
```

The result is 1, which is the numerical equivalent of `TRUE`.

See also **contains**

startTime

Syntax

```
sprite(whichSprite).startTime
the startTime of sprite whichSprite
sound(channelNum).startTime
```

Description

Sprite and sound property; for digital video sprites, determines when the specified digital video sprite begins. The value of `startTime` is measured in ticks.

For digital video sprites, this property can be tested and set. Set the `startTime` before the digital video member begins playback.

For sound channels, this property indicates the start time of the currently playing or paused sound as set when the sound was queued. It cannot be set after the sound has been queued. If no value was supplied when the sound was queued, this property returns zero.

Example

This statement starts the digital video sprite in channel 5 at 100 ticks into the digital video:

```
sprite(5).startTime = 100
```

See also

queue(), **setPlayList()**, **play() (sound)**

startTimer

Syntax `startTimer`

Description Command; sets the `timer` property to 0 and resets all the accumulating timers for the `lastClick()`, `lastEvent()`, `lastKey`, and `lastRoll` functions to 0.

If multiple timers are required, you must create and manage your own. These can be properties in a behavior, a global list, or even a parent script. Typically, you use the `ticks` property to track time in this manner.

Example This handler sets the timer to 0 when a key is pressed:

```
on keyDown
  startTimer
end
```

See also **lastClick()**,**lastEvent()**,**lastKey**, **lastRoll**; **timeoutLength**, **timeoutMouse**, **timeoutPlay**, **timeoutScript**, **timer**

state

Syntax

```
member(whichCastMember).state
state of member whichCastMember
```

Description

Cast member property; for Shockwave Audio (SWA) streaming cast members and Flash movie cast members, determines the current state of the streaming file. The properties `streamName`, `URL`, and `preLoadTime` can be changed only when the SWA sound is stopped.

The following properties for the SWA file return meaningful information only after the file begins streaming: `cuePointNames`, `cuePointTimes`, `currentTime`, `duration`, `percentPlayed`, `percentStreamed`, `bitRate`, `sampleRate`, and `numChannels`.

For SWA streaming cast members, the following values are possible:

- 0—Cast streaming has stopped.
- 1—The cast member is reloading.
- 2—Preloading ended successfully.
- 3—The cast member is playing.
- 4—The cast member is paused.
- 5—The cast member has finished streaming.
- 9—An error occurred.
- 10—There is insufficient CPU space.

For Flash movie cast members, this property returns a valid value only when the Director movie is running. The following values are possible:

- 0—The cast member is not in memory.
- 1—The header is currently loading.
- 2—The header has finished loading.
- 3—The cast member's media is currently loading.
- 4—The cast member's media has finished loading.
- -1—An error occurred.

This property can be tested but not set.

Example

This statement issues an alert if an error is detected for the SWA streaming cast member:

```
on mouseDown
  if member("Ella Fitzgerald").state = 9 then
    alert "Sorry, can't find an audio file to stream."
  end if
end
```

Example

This frame script checks to see if a Flash movie cast member named Intro Movie has finished streaming into memory. If it hasn't, the script reports in the Message window the current state of the cast member and keeps the playback head looping in the current frame until the movie finishes loading into memory.

```
on exitFrame
  if member("Intro Movie").percentStreamed < 100 then
    put "Current download state:" && member("Intro Movie").state
    go the frame
  end if
end
```

See also

clearError, **getError()**

static

Syntax

```
sprite(whichFlashSprite).static
the static of sprite whichFlashSprite
member(whichFlashMember).static
the static of member whichFlashMember
```

Description

Cast member property and sprite property; controls playback performance of a Flash movie sprite depending on whether the movie contains animation. If the movie contains animation (`FALSE`, default), the property redraws the sprite for each frame; if the movie doesn't contain animation (`TRUE`), the property redraws the sprite only when it moves or changes size.

This property can be tested and set.

Note: Set the `static` property to `TRUE` only when the Flash movie sprite does not intersect other moving Director sprites. If the Flash movie intersects moving Director sprites, it may not redraw correctly.

Example

This sprite script displays in the Message window the channel number of a Flash movie sprite and indicates whether the Flash movie contains animation:

```
property spriteNum
on beginSprite me
  if sprite(spriteNum).static then
    animationType = "does not have animation."
  else
    animationType = "has animation."
  end if
  put "The Flash movie in channel" && spriteNum && animationType
end
```

staticQuality

Syntax `staticQuality of sprite` *`whichQTVRSprite`*

Description QuickTime VR sprite property; specifies the codec quality used when the panorama image is static. Possible values are `#minQuality`, `#maxQuality`, and `#normalQuality`.

This property can be tested and set.

status

Syntax `soundObject.status`

the status of *soundObject*

Description Read-only property indicates the status of sound channel `channelNum`. Possible values include:

Status	Name	Meaning
0	Idle	No sounds are queued or playing.
1	Loading	A queued sound is being preloaded but is not yet playing.
2	Queued	The sound channel has finished preloading a queued sound but is not yet playing the sound.
3	Playing	A sound is playing.
4	Paused	A sound is paused.

Description

Example This statement displays the current status of sound channel 2 in the Message window:

```
put sound(2).status
```

See also **isBusy()**,**pause() (sound playback)**, **play() (sound)**, **stop() (sound)**

on stepFrame

Syntax

```
on stepFrame
  statement(s)
end
```

Description

System message and event handler; works in script instances in `actorList` because these are the only objects that receive `on stepFrame` messages. This event handler is executed when the playback head enters a frame or the Stage is updated.

An `on stepFrame` handler is a useful location for Lingo that you want to run frequently for a specific set of objects. Assign the objects to `actorList` when you want Lingo in the `on stepFrame` handler to run; remove the objects from `actorList` to prevent Lingo from running. While the objects are in `actorList`, the objects' `on stepFrame` handlers run each time the playback head enters a frame or the `updateStage` command is issued.

The `stepFrame` message is sent before the `prepareFrame` message.

Assign objects to `actorList` so they respond to `stepFrame` messages. Objects must have an `on stepFrame` handler to use this built-in functionality with `actorList`.

The `go`, `play`, and `updateStage` commands are disabled in an `on stepFrame` handler.

Example

If the child object is assigned to `actorList`, the `on stepFrame` handler in this parent script updates the position of the sprite that is stored in the `mySprite` property each time the playback head enters a frame:

```
property mySprite
on new me, theSprite
  mySprite = theSprite
  return me
end
on stepFrame me
  sprite(mySprite).loc = point(random(640),random(480))
end
```

stillDown

Syntax `the stillDown`

Description System property; indicates whether the user is pressing the mouse button (`TRUE`) or not (`FALSE`).

This function is useful within a `mouseDown` script to trigger certain events only after the `mouseUp` function.

Lingo cannot test `stillDown` when it is used inside a repeat loop. Use the `mouseDown` function inside repeat loops instead.

Example This statement checks whether the mouse button is being pressed and calls the handler `dragProcedure` if it is:

```
if the stillDown then dragProcedure
```

See also **the mouseDown (system property)**

stop (Flash)

Syntax

```
sprite(whichFlashSprite).stop()
```

```
stop sprite whichFlashSprite
```

Description Flash command; stops a Flash movie sprite that is playing in the current frame.

Example This frame script stops the Flash movie sprites playing in channels 5 through 10:

```
on enterFrame
  repeat with i = 5 to 10
    sprite(i).stop()
  end repeat
end
```

See also **hold**

stop() (sound)

Syntax

```
sound(channelNum).stop()
stop(sound(channelNum))
```

Description

Command; stops the currently playing sound in sound channel *channelNum*. Issuing a `play()` command begins playing the first sound of those that remain in the queue of the given sound channel.

To see an example of `stop()` used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement stops playback of the sound cast member currently playing in sound channel 1.

```
sound(1).stop()
```

See also

getPlaylist(), **pause() (sound playback)**, **play() (sound)**, **playNext()**, **rewind()**

stopEvent

Syntax `stopEvent`

Description Command; prevents Lingo from passing an event message to subsequent locations in the message hierarchy. Equivalent to the `dontPassEvent` command used in earlier versions of Director, this command also applies to sprite scripts.

Use the `stopEvent` command to stop the message in a primary event handler or a sprite script, thus making the message unavailable for subsequent sprite scripts.

By default, messages are available first to a primary event handler (if one exists) and then to any scripts attached to a sprite involved in the event. If more than one script is attached to the sprite, the message is available to each of the sprite's scripts. If no sprite script responds to the message, the message passes to a cast member script, frame script, and movie script, in that order.

The `stopEvent` command applies only to the current event being handled. It does not affect future events. The `stopEvent` command applies only within primary event handlers, handlers that primary event handlers call, or multiple sprite scripts. It has no effect elsewhere.

Example This statement shows the `mouseUp` event being stopped in a behavior if the global variable `grandTotal` is equal to 500:

```
global grandTotal
on mouseUp me
  if grandTotal = 500 then
    stopEvent
  end if
end
```

Neither subsequent scripts nor other behaviors on the sprite receive the event if it is stopped in this manner.

See also **pass**

stop member

Syntax

```
member (whichCastMember ).stop()
stop member (whichCastMember)
```

Description

Command; stops the playback of a Shockwave Audio (SWA) streaming cast member. When the cast member is stopped, the `state` member property equals 0.

For you to change properties such as `streamName`, `preLoadTime`, and `URL`, the SWA streaming cast member must be stopped.

Example

This statement stops the SWA cast member Big Band from playing:

```
member("Big Band").stop()
```

See also

play member, **pause member**

on stopMovie

Syntax

```
on stopMovie
  statement(s)
end
```

Description

System message and event handler; contains statements that run when the movie stops playing.

An `on stopMovie` handler is a good place to put Lingo that performs cleanup tasks—such as closing resource files, clearing global variables, erasing fields, and disposing of objects—when the movie is finished.

An `on stopMovie` handler in a MIAW is called only when the movie plays through to the end or branches to another movie. It isn't called when the window is closed or when the window is deleted by the `forget window` command.

Example

This handler clears global variables and closes two resource files when the movie stops:

```
global gCurrentScore
on stopMovie
  set gCurrentScore = 0
  closeResFile "Special Fonts"
  closeResFile "Special Cursors"
end
```

See also

on prepareMovie

stopTime

Syntax
```
sprite(whichSprite).stopTime
the stopTime of sprite whichSprite
```

Description
Sprite property; determines when the specified digital video sprite stops. The value of `stopTime` is measured in ticks.

This property can be tested and set.

Example
This statement stops the digital video sprite in channel 5 at 100 ticks into the digital video:

```
sprite(5).stopTime = 100
```

stream

Syntax

```
member(whichFlashSprite).stream(numberOfBytes )
stream(member whichFlashSprite, numberOfBytes)
```

Description

Command; manually streams a portion of a specified Flash movie cast member into memory. You can optionally specify the number of bytes to stream as an integer value. If you omit the *numberOfBytes* parameter, Director tries to stream the number of bytes set by the cast member's `bufferSize` property.

The `stream` command returns the number of bytes actually streamed. Depending on a variety of conditions (such as network speed or the availability of the requested data), the number of bytes actually streamed may be less than the number of bytes requested.

You can always use the `stream` command for a cast member regardless of the cast member's `streamMode` property.

Example

This frame script checks to see if a linked Flash movie cast member has streamed into memory by checking its `percentStreamed` property. If the cast member is not completely loaded into memory, the script tries to stream 32,000 bytes of the movie into memory.

The script also saves the actual number of bytes streamed in a variable called `bytesReceived`. If the number of bytes actually streamed does not match the number of bytes requested, the script updates a text cast member to report the number of bytes actually received. The script keeps the playback head looping in the current frame until the cast member has finished loading into memory.

```
on exitFrame
  if member(10).percentStreamed < 100 then
    bytesReceived = member(10).stream(32000)
    if bytesReceived < 32000 then
      member("Message Line").text = "Received only" &&
bytesReceived \
            && "of 32,000 bytes requested."
      updateStage
    else
      member("Message Line").text = "Received all 32,000 bytes."
    end if
    go the frame
  end if
end
```

streaming

Syntax

```
member(whichMember).streaming
the streaming of member whichMember
```

Description

QuickTime cast member property. When `TRUE`, allows QuickTime playing over the Internet to begin playing immediately while the member downloads to the user's computer. When `FALSE`, the member must download completely before playback will begin.

Defaults to `TRUE` for Director movies made with Director 7.02 and later. Defaults to `FALSE` for movies made with earlier versions of Director.

If the QuickTime member contains a text track with cue points, the text track must be set to preload in order for Director to make use of the cue points. You set the text track to preload using a video editor.

Example

This statement sets the streaming of member SunriseVideo to `FALSE`, causing it to download completely before playing back in Shockwave or ShockMachine:

```
member("SunriseVideo").streaming = 0
```

streamMode

Syntax

```
member(whichFlashMember).streamMode
the streamMode of member whichFlashMember
```

Description

Flash cast member property; controls the way a linked Flash movie cast member is streamed into memory, as follows:

- `#frame` (default)—Streams part of the cast member each time the Director frame advances while the sprite is on the Stage.
- `#idle`—Streams part of the cast member each time an idle event is generated or at least once per Director frame while the sprite is on the Stage.
- `#manual`—Streams part of the cast member into memory only when the `stream` command is issued for that cast member.

This property can be tested and set.

Example

This `startMovie` script searches the internal cast for Flash movie cast members and sets their `streamMode` properties to `#manual`:

```
on startMovie
  repeat with i = 1 to the number of members of castLib 1
    if member(i, 1).type = #flash then
      member(i, 1).streamMode = #manual
    end if
  end repeat
end
```

streamName

Syntax

```
member(whichCastMember).streamName
the streamName of member whichCastMember
```

Description

Shockwave Audio (SWA) cast member property; specifies a URL or file name for a streaming cast member. This property functions the same as the `URL` member property.

This property can be tested and set.

Example

This statement links the file BigBand.swa to an SWA streaming cast member. The linked file is on the disk MyDisk in the folder named Sounds.

```
member("SWAstream").streamName = "MyDisk/sounds/BigBand.swa"
```

streamSize

Syntax

```
member(whichFlashMember).streamSize
the streamSize of member whichFlashMember
```

Description

Cast member property; reports an integer value indicating the total number of bytes in the stream for the specified cast member. The `streamSize` property returns a value only when the Director movie is playing.

This property can be tested but not set.

Example

This frame script checks to see if a Flash movie cast member named Intro Movie has finished streaming into memory. If it hasn't, the script updates a field cast member to indicate the number of bytes that have been streamed (using the `bytesStreamed` member property) and the total number of bytes for the cast member (using the `streamSize` member property). The script keeps the playback head looping in the current frame until the movie finishes loading into memory.

```
on exitFrame
  if member("Intro Movie").percentStreamed < 100 then
    member("Message Line").text = string(member("Intro
Movie").bytesStreamed) \
    && "of" && string(member("Intro Movie").streamSize) \
    &&"bytes have downloaded so far."
    go to the frame
  end if
end
```

on streamStatus

Syntax

```
on streamStatus URL, state, bytesSoFar, bytesTotal, error
  statement(s)
end
```

Description

System message and event handler; called periodically to determine how much of an object has been downloaded from the Internet. The handler is called only if `tellStreamStatus (TRUE)` has been called, and the handler has been added to a movie script.

The `on streamStatus` event handler has the following parameters:

URL	Displays the Internet address of the data being retrieved.
state	Displays the state of the stream being downloaded. Possible values are `Connecting`, `Started`, `InProgress`, `Complete`, and `Error`.
BytesSoFar	Displays the number of bytes retrieved from the network so far.
BytesTotal	Displays the total number of bytes in the stream, if known. The value may be 0 if the HTTP server does not include the content length in the MIME header.
error	Displays an empty string ("") if the download has not finished; OK (*OK*) if the download completed successfully; displays an error code if the download was unsuccessful.

These parameters are automatically filled in by Director with information regarding the progress of the download. The handler is called by Director automatically, and there is no way to control when the next call will be. If information regarding a particular operation is needed, call `getStreamStatus()`.

You can initiate network streams using Lingo commands, by linking media from a URL, or by using an external cast member from a URL. A `streamStatus` handler will be called with information about all network streams.

Place the `streamStatus` handler in a movie script. For an animated demonstration of using the streamStatus event handler, see the **streamStatus** movie.

Example

This handler determines the state of a streamed object and displays the URL of the object:

```
on streamStatus URL, state, bytesSoFar, bytesTotal
  if state = "Complete" then
    put URL && "download finished"
  end if
end streamStatus
```

See also

getStreamStatus(),**tellStreamStatus()**

string()

Syntax `string(expression)`

Description Function; converts an integer, floating-point number, object reference, list, symbol, or other nonstring expression to a string.

Example This statement adds 2.0 + 2.5 and inserts the results in the field cast member Total:

```
member("total").text = string(2.0 + 2.5)
```

Example This statement converts the symbol `#red` to a string and inserts it in the field cast member Color:

```
member("Color").text = string(#red)
```

See also **value()**,**stringP()**,**float()**,**integer()**,**symbol()**

stringP()

Syntax `stringP(expression)`

Description Function; determines whether an expression is a string (`TRUE`) or not (`FALSE`).

The *P* in `stringP` stands for *predicate*.

Example This statement checks whether *3* is a string:

```
put stringP("3")
```

The result is 1, which is the numeric equivalent of `TRUE`.

Example This statement checks whether the floating-point number 3.0 is a string:

```
put stringP(3.0)
```

Because 3.0 is a floating-point number and not a string, the result is 0, which is the numeric equivalent of `FALSE`.

See also **floatP()**,**ilk()**,**integerP()**,**objectP()**,**symbolP()**

strokeColor

Syntax `member(whichCastMember).strokeColor`

Description Vector shape cast member property; indicates the color in RGB of the shape's framing stroke.

To see an example of `strokeColor` used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This sets the strokeColor of cast member "line" to red.

```
member("line").strokeColor = rgb(255, 0, 0)
```

See also **color()**,**fillColor**, **endColor**, **backgroundColor**

strokeWidth

Syntax `member(whichCastMember).strokeWidth`

Description Vector shape cast member property; indicates the width, in pixels, of the shape's framing stroke.

The value is a floating point number between 0 and 100, and can be tested and set.

To see an example of `strokeWidth` used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This sets the strokeWidth of cast member "line" to 10 pixels.

```
member(1).strokeWidth = 10
```

substituteFont

Syntax `TextMemberRef.substituteFont(originalFont, newFont)`

`substituteFont(textMemberRef, originalFont, newFont)`

Description Text cast member command; replaces all instances of *originalFont* with *newFont* in *textMemberRef*.

Example This script checks to see if the font Bonneville is available in a text cast member, and replaces it with Arial if it is not:

```
property spriteNum
on beginSprite me
  currMember = sprite(spriteNum).member
  if currMember.missingFonts contains "Bonneville" then
    currMember.substituteFont("Bonneville", "Arial")
  end if
end
```

See also **missingFonts**

swing()

Syntax
```
WhichQTVRSprite.swing(pan, tilt, fieldOfView, speedToSwing)
swing (whichQTVRSprite, pan, tilt, fieldOfView, speedToSwing)
```

Description QuickTime VR sprite function; swings a QuickTime 3 sprite containing a VR Pano around to the new view settings. The swing is a smooth “camera dolly” effect.

- *whichQTVRSprite*—Sprite number of the sprite with the QuickTime VR member.
- *pan*—New pan position, in degrees.
- *tilt*—New tilt, in degrees.
- *fieldOfView*—New field of view, in degrees.
- *speedToSwing*—Rate at which the swing should take place; specify an integer from 1 to 10 (slow to fast).

The function returns immediately, but the sprite continues to change view until it reaches the final view. The duration required to change to the final settings varies depending on machine type, size of the sprite rectangle, color depth of the screen, and other typical performance constraints.

To check if the swing has finished, check if the `pan` property of the sprite has arrived at the final value.

Example This very gradually adusts the view of QTVR sprite 1 to a pan position of 300 degrees, a tilt of -15 degrees, and a field of view of 40 degrees.

```
sprite(1).swing(300, -15, 40, 1)
```

See also **pan (QTVR property)**

switchColorDepth

Syntax `the switchColorDepth`

Description System property; determines whether Director switches the monitor that the Stage occupies to the color depth of the movie being loaded (`TRUE`) or leaves the color depth of the monitor unchanged when a movie is loaded (`FALSE`). False is the default value.

When `switchColorDepth` is `TRUE`, nothing happens until a new movie is loaded.

Setting the monitor's color depth to that of the movie is good practice.

- When the monitor's color depth is set below that of the movie, resetting it to the color depth of the movie (assuming that the monitor can provide that color depth) helps maintain the movie's original appearance.
- When the monitor's color depth is higher than that of the movie, reducing the monitor's color depth plays the movie using the minimum amount of memory, loads cast members more efficiently, and causes animation to occur more quickly.

This property can be tested and set. The default value comes from the Reset Monitor to Movie's Color Depth option in the General Preferences dialog box.

Example This statement sets the variable named `switcher` to the current setting of `switchColorDepth`:

```
switcher = the switchColorDepth
```

Example This statement checks whether the current color depth is 8-bit and turns the `switchColorDepth` property on if it is:

```
if the colorDepth = 8 then the switchColorDepth = TRUE
```

See also **colorDepth**

symbol()

Syntax
```
stringValue.symbol
symbol(stringValue)
```

Description Function; takes a string, *stringValue*, and returns a symbol.

Example This statement displays the symbol `#hello`:

```
put ("hello").symbol
-- #hello
```

Example This statement displays the symbol `#goodbye`:

```
x = "goodbye"
put x.symbol
-- #goodbye
```

See also **value()**,**string()**

symbolP()

Syntax
```
Expression.symbolP
symbolP(expression)
```

Description Function; determines whether the expression specified by *`expression`* is a symbol (`TRUE`) or not (`FALSE`).

The *P* in `symbolP` stands for *predicate*.

Example This statement checks whether the variable `myVariable` is a symbol:

```
put myVariable.symbolP
```

See also **ilk()**

systemDate

Syntax `the systemDate`

Description System property; returns the current date in a standard date format and can be used in conjunction with other date operations for international and cross- platform date manipulation.

Math operations on the date are performed in days.

This property can be tested but not set.

Example This script displays the current date being retrieved and then determines the date 90 days from the current date:

```
on ShowEndOfTrialPeriodDate
  set today = the systemDate
  set trialEndDate = today + 90
  put "The trial period for this software is over
on"&&trialEndDate
end
```

See also **date() (system clock)**

TAB

Syntax `TAB`

Description Constant; represents the Tab key.

Example This statement checks whether the character typed is the tab character and calls the handler `doNextField` if it is:

```
if the key = TAB then doNextField
```

Example These statements move the playback head forward or backward, depending on whether the user presses Tab or Shift-Tab:

```
if the key = TAB then
  if the shiftDown then
    go the frame -1
  else
    go the frame +1
  end if
end if
```

See also **BACKSPACE**, **EMPTY**, **RETURN (constant)**

tabCount

Syntax `chunkExpression.tabCount`

Description Text cast member property; indicates how many unique tab stops are in the specified chunk expression of the text cast member.

The value is an integer equal to or greater than 0, and may be tested but not set.

tabs

Syntax `member(whichTextMember).tabs`

Description Text cast member property; this property contains a list of all the tab stops set in the text cast member.

Each element of the list contains information regarding that tab for the text cast member. The possible properties in the list are as follows:

#type	Can be #left, #center, #right, or #decimal.
#position	Integer value indicating the position of the tab in points.

You can get and set this property. When `tabs` is set, the `type` property is optional. If `type` is not specified, the tab type defaults to `#left`.

Example This statement retrieves and displays in the Message window all the tabs for the text cast member Intro credits:

```
put member("Intro credits").tabs
-- [[#type: #left, #position: 36], [#type: #Decimal, #position:
141],  \
  [#type: #right, #position: 216]]
```

tan()

Syntax `tan(angle)`

Description Math function; yields the tangent of the specified angle expressed in radians as a floating-point number.

Example The following function yields the tangent of pi/4:

```
tan (PI/4.0) = 1
```

The π symbol cannot be used in a Lingo expression.

See also **PI**

target

Syntax `timeoutObject.target`

Description Timeout object property; indicates the child object that the given *`timeoutObject`* will send its timeout events to. Timeout objects whose target property is `VOID` will send their events to a handler in a movie script.

This property is useful for debugging behaviors and parent scripts that use timeout objects.

Example This statement displays the name of the child object that will receive timeout events from the timeout object `timerOne` in the Message window:

```
put timeout(“timerOne”).target
```

See also **name (timeout property)**, **timeout()**,**timeoutHandler**, **timeoutList**

tell

Syntax

```
tell whichWindow to statement(s)
```

```
tell whichWindow
  statement(s)
end
```

Description

Command; communicates statements to the window specified by *whichWindow*.

The `tell` command is useful for allowing movies to interact. It can be used within a main movie to send a message to a movie playing in a window, or to send a message from a movie playing in a window to the main movie. For example, the `tell` command can let a button in a control panel call a handler in a movie playing in a window. The movie playing in a window may react to the first movie handler by executing the handler. The movie playing in the window may interact with the main movie by sending a value back to the movie.

When you use the `tell` command to send a message to a movie playing in a window, identify the window object by using the full pathname or its number in `windowList`. If you use `windowList`, use the expression `getAt(the windowList, windowNum)`, where *windowNum* is a variable that contains the number of the window's position in the list. Because the opening and closing of windows may change the order of `windowList`, it is a good idea to store the full pathname as a global variable when referencing windows with `getAt` in `windowList`.

Example

A multiple-line `tell` command resembles a handler and requires an `end tell` statement:

```
global childMovie
tell window childMovie
  go to frame "Intro"
  the stageColor = 100
  sprite(4).member = member "Diana Ross"
  updateStage
end tell
```

Example

When a message calls a handler, a value returned from the handler can be found in the global `result` property after the called handler is done. These statements send the `childMovie` window the message `calcBalance` and then return the result:

```
global childMovie
tell window childMovie to calcBalance
-- a handler name
myBalance = result()
-- return value from calcBalance handler
```

Example

When you use the `tell` command to send a message from a movie playing in a window to the main movie, use the `stage` system property as the object name:

```
tell the stage to go frame "Main Menu"
```

When you use the `tell` command to call a handler in another movie, make sure that you do not have a handler by the same name in the same script in the local movie. If you do, the local script will be called. This restriction applies only to handlers in the same script in which you are using the `tell` command.

Example

This statement causes the Control Panel window to instruct the Simulation movie to branch to another frame:

```
tell window "Simulation" to go frame "Save"
```

tellStreamStatus()

Syntax `tellStreamStatus(`*`onOrOffBoolean`*`)`

Description Function; turns the stream status handler on (`TRUE`) or off (`FALSE`).

The form `tellStreamStatus()` determines the status of the handler.

When the `streamStatusHandler` is `TRUE`, Internet streaming activity causes periodic calls to the movie script, triggering streamStatusHandler. The handler is executed, with Director automatically filling in the parameters with information regarding the progress of the downloads.

Example This `on prepareMovie` handler turns the `on streamStatus` handler on when the movie starts:

```
on prepareMovie
  tellStreamStatus(TRUE)
end
```

Example This statement determines the status of the stream status handler:

```
on mouseDown
  put tellStreamStatus()
end
```

See also **on streamStatus**

text

Syntax

```
member(whichCastMember).text
the text of member whichCastMember
```

Description

Text cast member property; determines the character string in the field cast member specified by *whichCastMember*.

The `text` cast member property is useful for displaying messages and recording what the user types.

This property can be tested and set.

When you use Lingo to change the entire text of a cast member you remove any special formatting you have applied to individual words or lines. Altering the `text` cast member property reapplies global formatting. To change particular portions of the text, refer to lines, words, or items in the text.

When the movie plays back as an applet, this property's value is `""` (an empty string) for a field cast member whose text has not yet streamed in.

To see an example of `text` used in a completed movie, see the Forms and Post movie in the Learning\Lingo Examples folder inside the Director application folder.

To see an example of `text` used in a completed movie, see the Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement places the phrase "Thank you." in the empty cast member Response:

```
if member("Response").text = EMPTY then
     member("Response").text = "Thank You."
```

This statement sets the content of cast member Notice to "You have made the right decision!"

```
member("Notice").text = "You have made the right decision!"
```

See also

selEnd, **selStart**

the

Syntax `the property`

Description Keyword; must precede many functions and all Lingo properties written in verbose syntax. This keyword also distinguishes the property or function from a variable or object name.

Earlier versions of Director required you to use the `the` keyword to express cast member and sprite properties. This syntax is still supported as alternate form.

Properties are globally available to handlers even if you don't declare them globally. Like global variables, Lingo system properties are available between different movies in the same presentation. Sprite properties change when a new movie is loaded.

thumbnail

Syntax

```
member(whichMember).thumbnail
the thumbnail of member whichMember
```

Description

Cast member property; contains the image used to preview a cast member in the Cast window. This image can be customized for any cast member.

This property can be tested and set only during authoring.

Example

This statement shows how to use a placeholder cast member to display another thumbnail on the Stage. The placeholder cast member is placed on the Stage, then the picture of that member is set to the thumbnail of member 10. This makes it possible to show a reduced image without having to scale or otherwise manipulate a graphic:

```
member("Placeholder").picture = member(10).thumbnail
```

See also

picture (cast member property)

ticks

Syntax `the ticks`

Description System property; returns the current time in ticks (1 tick = 1/60 of a second). Counting ticks begins from the time the computer is started.

Example This statement converts ticks to seconds and minutes by dividing the number of ticks by 60 twice and then sets the variable `minutesOn` to the result:

```
currentSeconds = the ticks/60
currentMinutes = currentSeconds/60
```

See also **time()**,**timer**, **milliseconds**

tilt

Syntax `tilt of sprite (whichQTVRSprite)`

Description QuickTime VR sprite property; the current tilt, in degrees, of the QuickTime VR movie. This property can be tested and set.

time()

Syntax

```
the time
the short time
the long time
the abbreviated time
the abbrev time
the abbr time
```

Description

Function; returns the current time in the system clock as a string in one of three formats: `short`, `long`, or `abbreviated`. If you don't specify a format, the default format is `short`. The `abbreviated` format can also be referred to as `abbrev` and `abbr`. In the United States, the `short` and `abbreviated` formats are the same.

Example

These statements display the time in different formats in the Message window. Possible results appear below each statement.

```
put the short time
--"1:30 PM"
put the long time
--"1:30:24 PM"
put the abbreviated time
--"1:30 PM"
```

The three time formats vary, depending on the individual computer's time format. The preceding examples are for the United States.

See also

date() (system clock)

time (timeout object property)

Syntax `timeoutObject.time`

Description Timeout object property; the system time, in milliseconds, when the next timeout event will be sent by the given *`timeoutObject`*.

Note that this is not the time until the next event, but the absolute time of the next timeout event.

Example This handler determines the time remaining until the next timeout event will be sent by the timout object `Update` by calculating the difference between its `time` property and the current value of the milliseconds and displaying the result in the field Time Until:

```
on prepareFrame
  msBeforeUpdate = timeout("Update").time - the milliseconds
  secondsBeforeUpdate = msBeforeUpdate / 1000
  minutesBeforeUpdate = secondsBeforeUpdate / 60
  member("Time Until").text = string(minutesBeforeUpdate) &&
"minutes before next \
  timeout”
end
```

See also **milliseconds**, **period**, **persistent**, **target**, **timeout()**,**timeoutHandler**

timeout()

Syntax `timeout("timeoutName")`

Description Function; returns the timeout object named *timeoutName*. Use `timeout(“name”).new` to add a new timeout object to `the timeoutList`. See `new()`.

Example This handler deletes the timeout object named Random Lightning:

```
on exitFrame
  timeout("Random Lightning”).forget()
end
```

See also **forget()**,**new()**,**timeoutHandler**, **timeoutList**

on timeOut

Syntax

```
on timeOut
  statement(s)
end
```

Description

System message and event handler; contains statements that run when the keyboard or mouse is not used for the time period specified in `timeOutLength`. Always place an `on timeOut` handler in a movie script.

To have a timeout produce the same response throughout a movie, use `the timeoutScript` to centrally control timeout behavior.

Example

This handler plays the movie Attract Loop after users do nothing for the time set in the `timeoutLength` property. It can be used to respond when users leave the computer.

```
on timeOut
  play movie "Attract Loop"
end timeOut
```

See also

timeoutScript, **timeoutLength**

timeoutHandler

Syntax *timeoutObject*.timeoutHandler

Description System property; represents the name of the handler that will receive timeout messages from the given *timeoutObject*. Its value is a symbol, such as #timeExpiredHandler. The timeoutHandler is always a handler within the timeout object's target object, or in a movie script if the timeout object has no target specified.

This property can be tested and set.

Example This statement displays the timeoutHandler of the timeout object Quiz Timer in the Message window:

```
put timeout("Quiz Timer").timeoutHandler
```

See also **target**, **timeout()**,**timeoutList**

timeoutKeyDown

Syntax `the timeoutKeyDown`

Description System property; determines whether `keyDown` events set the `timeoutLapsed` property to 0 (`TRUE`, default) or not (`FALSE`). This property is useful for restarting the countdown for a timeout each time a key is pressed.

This property can be tested and set.

Example This statement sets the variable timing to the value of the `timeoutKeyDown` property:

```
timing = timeoutKeyDown
```

This statement turns off the `timeoutKeyDown` property:

```
timeoutKeyDown = FALSE
```

See also **keyDownScript**

timeoutLapsed

Syntax `the timeoutLapsed`

Description System property; indicates how many of ticks have elapsed since the last timeout. A timeout event occurs when the `timeoutLapsed` property reaches the time specified by the `timeoutLength` property.

The `timeoutLapsed` property can be tested and set.

Example This statement sets the Countdown member field to the value of the `timeoutLapsed` property. Dividing `timeoutLapsed` by 60 converts the value to seconds.

```
member("Countdown").text = string(the timeoutLapsed / 60)
```

timeoutLength

Syntax `the timeoutLength`

Description System property; determines how many ticks must elapse before a timeout event occurs. A timeout occurs when the `timeoutLapsed` property reaches the time specified by `timeoutLength`.

This property can be tested and set. The default value is 10,800 ticks or 3 minutes.

Example This statement sets `timeoutLength` to 10 seconds:

```
set the timeoutLength to 10 * 60
```

or

```
the timeoutLength = 10 * 60
```

timeoutList

Syntax `the timeoutList`

Description System property; a linear list containing all currently active timeout objects. Use the `forget()` function to delete a timeout object.

Timeout objects are added to the `timeoutList` with the `new()` funtion.

Example This statement deletes the third timeout object from the timeout list:

```
the timeoutList[3].forget()
```

See also **forget()**, **new()**, **timeout()**, **target**, **timeoutHandler**

timeoutMouse

Syntax `the timeoutMouse`

Description System property; determines whether `mouseDown` events reset the `timeoutLapsed` property to 0 (`TRUE`, default) or not (`FALSE`).

This property can be tested and set.

Example This statement records the current setting of `timeoutMouse` by setting the variable named `timing` to `the timeoutMouse`:

```
timing = the timeoutMouse
```

Example This statement sets the `timeoutMouse` property to `FALSE`. The result is that the `timeoutLapsed` property keeps its current value when the mouse button is pressed.

```
the timeoutMouse = FALSE
```

See also **mouseDownScript**, **mouseUpScript**

timeoutPlay

Syntax `the timeoutPlay`

Description System property; determines whether the `timeoutLapsed` property will be set to `TRUE` when the movie is paused with the `pause` command. When `TRUE`, timeouts will occur when the movie is paused. When `FALSE`, timouts will not occur when the movie is paused. The default value is `FALSE`.

This property can be tested and set.

Example This statement sets `timeoutPlay` to `TRUE`, which tells Lingo to reset the `timeoutLapsed` property to 0 after a movie is played:

```
set the timeoutPlay to TRUE
```

or

```
the timeoutPlay = TRUE
```

timeoutScript

Syntax `the timeoutScript`

Description System property; determines the Lingo that Director executes as a primary event handler when a timeout occurs. The Lingo is written as a string, surrounded by quotation marks. The default value is `EMPTY`.

To define a primary event handler for timeouts, set `timeoutScript` to a string of the appropriate Lingo: either a simple statement or a calling statement for a handler. When the assigned event script is no longer appropriate, turn it off with the statement `set the timeoutScript to EMPTY`.

This property can be tested and set.

Example This statement sets `timeoutScript` to a calling script for the handler `timeoutProcedure`:

```
the timeoutScript = "timeoutProcedure"
```

See also **on timeOut**

timer

Syntax `the timer`

Description System property; a free running timer that counts time in ticks (1 tick = 1/60 second). It has nothing to do with the `timeoutScript` property. It is used only for convenience in timing certain events. The `startTimer` command sets timer to 0.

The `timer` property is useful for determining the amount of time passed since the `startTimer` command was issued. For example, you can use `timer` to synchronize pictures with a soundtrack by inserting a delay that makes the movie wait until a certain amount of time has elapsed.

Example This behavior for a frame script creates a 2-second delay:

```
on beginSprite
  startTimer
end
on exitFrame
  if (the timer < 60 * 2) then go the frame
end
```

Example This statement sets the variable `startTicks` to the current `timer` value:

```
startTicks = the timer
```

See also **lastClick()**,**lastEvent()**,**lastKey**, **lastRoll**; **startTimer**

timeScale

Syntax `member(whichCastMember).timeScale`

`the timeScale of member whichCastMember`

Description Cast member property; returns the time unit per second on which the digital video's frames are based. For example, a time unit in a QuickTime digital video is 1/600 of a second.

This property can be tested but not set.

See also **digitalVideoTimeScale**

title

Syntax
```
window (whichWindow.title)
the title of window whichWindow
```

Description Window property; assigns a title to the window specified by *whichWindow*.

This property can be tested and set for windows other than the Stage.

Example This statement makes Action View the title of window X:

```
window("X").title = "Action View"
```

titleVisible

Syntax

```
window (whichWindow.titleVisible)
the titleVisible of window whichWindow
```

Description

Window property; specifies whether the window specified by *whichWindow* displays the window title in the window's title bar.

This property can be tested and set for windows other than the Stage.

Example

This statement displays the title of the Control Panel window by setting the window's `titleVisible` property to `TRUE`:

```
window("Control Panel").titleVisible = TRUE
```

to

The word `to` occurs in a number of Lingo constructs.

See also **char...of**, **item...of**, **line...of**, **word...of**, **repeat with**, **set...to, set...=**

top

Syntax
```
sprite(whichSprite).top
the top of sprite whichSprite
```

Description
Sprite property; returns the top vertical coordinate of the bounding rectangle of the sprite specified by *whichSprite* as the number of pixels from the upper left corner of the Stage.

When the movie plays back as an applet, this property's value is relative to the upper left corner of the applet.

This property can be tested and set.

Example
This statement checks whether the top of sprite 3 is above the top of the Stage and calls the handler `offTopEdge` if it is:

```
if sprite(3).top < 0 then offTopEdge
```

See also
bottom, **height**, **locH**, **left**, **locV**, **right**, **width**

topSpacing

Syntax `chunkExpression.topSpacing`

Description Text cast member property; allows you to specify additional spacing applied to the top of each paragraph in the *`chunkExpression`* portion of the text cast member.

The value itself is an integer, with less than 0 indicating less spacing between paragraphs and greater than 0 indicating more spacing between paragraphs.

The default value is 0, which results in default spacing between paragraphs.

Example This statement sets the `topSpacing` of the second paragraph in text cast member `"myText"` to 20:

```
member(1).paragraph[2].topSpacing = 20
```

See also **bottomSpacing**

trace

Syntax `the trace`

Description Movie property; specifies whether the movie's trace function is on (`TRUE`) or off (`FALSE`). When the trace function is on, the Message window displays each line of Lingo that is being executed.

This property can be tested and set.

Example This statement turns the `trace` property on:

```
the trace = TRUE
```

See also **traceLogFile**

traceLoad

Syntax `the traceLoad`

Description Movie property; specifies the amount of information that is displayed about cast members as they load:

- 0—Displays no information.
- 1—Displays cast members' names.
- 2—Displays cast members' names, the number of the current frame, the movie name, and the file seek offset (the relative amount the drive had to move to load the media).

The default value for the `traceLoad` property is 0. This property can be tested and set.

Example This statement causes the movie to display the names of cast members as they are loaded:

```
the traceLoad = 1
```

traceLogFile

Syntax `the traceLogFile`

Description System property; specifies the name of the file in which the Message window display is written. You can close the file by setting the `traceLogFile` property to `EMPTY` (""). Any output that would appear in the Message window is written into this file. You can use this property for debugging when running a movie in a projector and when authoring.

Example This statement instructs Lingo to write the contents of the Message window in the file "Messages.txt in the same folder as the current movie:

```
the traceLogFile = the moviePath & "Messages.txt"
```

Example This statement closes the file that the Message window display is being written to:

```
the traceLogFile = ""
```

trackCount (cast member property)

Syntax

```
member(whichCastMember).trackCount()
trackCount(member whichCastMember)
```

Description

Digital video cast member property; returns the number of tracks in the specified digital video cast member.

This property can be tested but not set.

Example

This statement determines the number of tracks in the digital video cast member Jazz Chronicle and displays the result in the Message window:

```
put member("Jazz Chronicle").trackCount()
```

trackCount (sprite property)

Syntax

```
sprite(whichDigitalVideoSprite).trackCount()
trackCount(sprite whichSprite)
```

Description

Digital video sprite property; returns the number of tracks in the specified digital video sprite.

This property can be tested but not set.

Example

This statement determines the number of tracks in the digital video sprite assigned to channel 10 and displays the result in the Message window:

```
put sprite(10).trackCount()
```

trackEnabled

Syntax `sprite(whichDigitalVideoSprite).trackEnabled(whichTr ack)`

`trackEnabled(sprite` *`whichSprite`*`,` *`whichTrack`*`)`

Description Digital video sprite property; indicates the status of the specified track of a digital video. This property is `TRUE` if the track is enabled and playing. This property is `FALSE` if the track is disabled and no longer playing or is not updating.

This property cannot be set. Use the `setTrackEnabled` property instead.

Example This statement checks whether track 2 of digital video sprite 1 is enabled:

```
put sprite(1).trackEnabled(2)
```

See also **setTrackEnabled**

trackNextKeyTime

Syntax

```
sprite(whichDigitalVideoSprite).trackNextKeyTime(whi chTrack)
trackNextKeyTime(sprite whichSprite, whichTrack)
```

Description

Digital video sprite property; indicates the time of the keyframe that follows the current time in the specified digital video track.

This property can be tested but not set.

Example

This statement determines the time of the keyframe that follows the current time in track 5 of the digital video assigned to sprite channel 15 and displays the result in the Message window:

```
put sprite(15).trackNextKeyTime(5)
```

trackNextSampleTime

Syntax `sprite(whichDigitalVideoSprite).trackNextSampleTime( whichTrack)`

trackNextSampleTime(sprite *whichSprite*, *whichTrack*)

Description Digital video sprite property; indicates the time of the next sample that follows the digital video's current time. This property is useful for locating text tracks in a digital video.

This property can be tested but not set.

Example This statement determines the time of the next sample that follows the current time in track 5 of the digital video assigned to sprite 15:

```
put sprite(15).trackNextSampleTime(5)
```

trackPreviousKeyTime

Syntax

```
sprite(whichDigitalVideoSprite).trackPreviousKeyTime (whichTrack)
trackPreviousKeyTime(sprite whichSprite, whichTrack)
```

Description

Digital video sprite property; reports the time of the keyframe that precedes the current time.

This property can be tested but not set.

Example

This statement determines the time of the keyframe in track 5 that precedes the current time in the digital video sprite assigned to channel 15 and displays the result in the Message window:

```
put sprite(2).trackPreviousKeyTime(1)
```

trackPreviousSampleTime

Syntax

```
sprite(whichDigitalVideoSprite).trackPreviousSampleT
ime(whichTrack)
```

```
trackPreviousSampleTime(sprite whichSprite, whichTrack)
```

Description

Digital video sprite property; indicates the time of the sample preceding the digital video's current time. This property is useful for locating text tracks in a digital video.

This property can be tested but not set.

Example

This statement determines the time of the sample in track 5 that precedes the current time in the digital video sprite assigned to channel 15 and displays the result in the Message window:

```
put sprite(15).trackPreviousSampleTime(5)
```

trackStartTime (cast member property)

Syntax

```
member(whichDigitalVideoCastmember).trackStartTime(w hichTrack)
trackStartTime(member whichCastMember, whichTrack)
```

Description

Digital video cast member property; returns the start time of the specified track of the specified digital video cast member.

This property can be tested but not set.

Example

This statement determines the start time of track 5 in the digital video cast member Jazz Chronicle and displays the result in the Message window:

```
put member("Jazz Chronicle").trackStartTime(5)
```

trackStartTime (sprite property)

Syntax

```
sprite(whichDigitalVideoSprite).trackStartTime(which Track)
trackStartTime(sprite whichSprite, whichTrack)
```

Description

Digital video sprite property; sets the starting time of a digital video movie in the specified sprite channel. The value of `trackStartTime` is measured in ticks.

This property can be tested but not set.

Example

In the Message window, this statement reports when track 5 in sprite channel 10 starts playing. The starting time is 120 ticks (2 seconds) into the track.

```
put sprite(10).trackStartTime(5)
-- 120
```

See also

duration, **movieRate**, **movieTime**

trackStopTime (cast member property)

Syntax
```
member(whichDigitalVideoCastmember).trackStopTime(wh ichTrack)
trackStopTime(member whichCastMember, whichTrack)
```

Description Digital video cast member property; returns the stop time of the specified track of the specified digital video cast member. It can be tested but not set.

Example This statement determines the stop time of track 5 in the digital video cast member Jazz Chronicle and displays the result in the Message window:

```
put member("Jazz Chronicle").trackStopTime(5)
```

trackStopTime (sprite property)

Syntax

```
sprite(whichDigitalVideoSprite).trackStopTime(whichT rack)
trackStopTime(sprite, whichSprite, whichTrack)
```

Description

Digital video sprite property; returns the stop time of the specified track of the specified digital video sprite.

When a digital video movie is played, `trackStopTime` is when playback halts or loops if the `loop` property is turned on.

This property can be tested but not set.

Example

This statement determines the stop time of track 5 in the digital video assigned to sprite 6 and displays the result in the Message window:

```
put sprite(6).trackStopTime(5)
```

See also

movieRate, **movieTime**, **trackStartTime (cast member property)**

trackText

Syntax

```
sprite(whichDigitalVideoSprite).trackText(whichTrack )
trackText(sprite whichSprite, whichTrack)
```

Description

Digital video sprite property; provides the text that is in the specified track of the digital video at the current time. The result is a string value, which can be up to 32K characters long. This property applies to text tracks only.

This property can be tested but not set.

Example

This statement assigns the text in track 5 of the digital video assigned at the current time to sprite 20 to the field cast member Archives:

```
member("Archives").text = string(sprite(20).trackText(5))
```

trackType (cast member property)

Syntax

```
member(whichDigitalVideoCastmember).trackType(whichT rack)
trackType(member whichCastMember, whichTrack)
```

Description

Digital video cast member property; indicates which type of media is in the specified track of the specified cast member. Possible values are `#video`, `#sound`, `#text`, and `#music`.

This property can be tested but not set.

Example

The following handler checks whether track 5 of the digital video cast member Today's News is a text track and then runs the handler `textFormat` if it is:

```
on checkForText
  if member("Today's News").trackType(5) = #text then textFormat
end
```

trackType (sprite property)

Syntax

```
sprite(whichDigitalVideoSprite).trackType(whichTrack )
trackType(sprite whichSprite, whichTrack)
```

Description

Digital video sprite property; returns the type of media in the specified track of the specified sprite. Possible values are `#video`, `#sound`, `#text`, and `#music`.

This property can be tested but not set.

Example

The following handler checks whether track 5 of the digital video sprite assigned to channel 10 is a text track and runs the handler `textFormat` if it is:

```
on checkForText
  if sprite(10).trackType(5) = #text then textFormat
end
```

trails

Syntax
```
sprite(whichSprite).trails
the trails of sprite whichSprite
```

Description
Sprite property; for the sprite specified by *whichSprite*, turns the trails ink effect on (1 or TRUE) or off (0 or FALSE). For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet.

To erase trails, animate another sprite across these pixels or use a transition.

Example
This statement turns on trails for sprite 7:

```
sprite(7).trails = 1
```

See also
directToStage

transitionType

Syntax

```
member(whichCastMember).transitionType
the transitionType of member whichCastMember
```

Description

Transition cast member property; determines a transition's type, which is specified as a number. The possible values are the same as the codes assigned to transitions for the `puppetTransition` command.

Example

This statement sets the type of transition cast member 3 to 51, which is a pixel dissolve cast member:

```
member(3).transitionType = 51
```

translation

Syntax

```
member(whichQuickTimeMember).translation
the translation of member whichQuickTimeMember
sprite(whichQuickTimeSprite).translation
the translation of sprite whichQuickTimeSprite
```

Description

QuickTime cast member and sprite property; controls the offset of a QuickTime sprite's image within the sprite's bounding box.

This offset is expressed in relation to the sprite's default location as set by its `center` property. When `center` is set to `TRUE`, the sprite is offset relative to the center of the bounding rectangle; when `center` is set to `FALSE`, the sprite is offset relative to the upper left corner of the bounding rectangle.

The offset, specified in pixels as positive or negative integers, is set as a Director list: [*xTrans*, *yTrans*]. The *xTrans* parameter specifies the horizontal offset from the sprite's default location; the *yTrans* parameter specifies the vertical offset. The default setting is [0,0].

When the sprite's `crop` property is set to `TRUE`, the `translation` property can be used to mask portions of the QuickTime movie by moving them outside the bounding rectangle. When the `crop` property is set to `FALSE`, the `translation` property is ignored, and the sprite is always positioned at the upper left corner of the sprite's rectangle.

This property can be tested and set.

Example

This frame script assumes that the center property of the cast member of a 320- pixel-wide QuickTime sprite in channel 5 is set to `FALSE`, and its crop property is set to `TRUE` . It keeps the playback head in the current frame until the movie's horizontal translation point has moved to the right edge of the sprite, in 10-pixel increments. This has a wipe right effect, moving the sprite out of view to the right. When the sprite is out of view, the playback head continues to the next frame.

```
on exitFrame
  horizontalPosition = sprite(5).translation[1]
  if horizontalPosition < 320 then
    sprite(5).translation = sprite(5).translation + [10, 0]
    go the frame
  end if
end
```

triggerCallback

Syntax

```
sprite(whichQTVRSprite).triggerCallback
triggerCallback of sprite whichQTVRSprite
```

Description

QuickTime VR sprite property; contains the name of the handler that runs when the user clicks a hotspot in a QuickTime VR movie. The handler is sent two arguments: the `me` parameter and the ID of the hotspot that the user clicked.

The value that the handler returns determines how the movie processes the hotspot. If the handler returns `#continue`, the QuickTime VR sprite continues to process the hotspot normally. If the handler returns `#cancel`, the default behavior for the hotspot is canceled.

Set this property to 0 to clear the callback.

The QuickTime VR sprite receives the message first.

To avoid a decrease in performance, set the `triggerCallback` property only when necessary.

This property can be tested and set.

Example

This statement sets the callback handler for a QuickTime VR sprite to the handler named `MyHotSpotCallback` when the playback head first enters the sprite span. Every time that hotspot is triggered, the `MyHotSpotCallback` handler is executed. When the playback head leaves the sprite span, the callback is canceled.

```
property pMySpriteNum, spriteNum
on beginSprite me
  pMySpriteNum = me.spriteNum
  sprite(pMySpriteNum).triggerCallback = #MyHotSpotCallback
end
on MyHotSpotCallback me, hotSpotID
  put "Hotspot" && hotSpotID && "was just triggered"
end
on endSprite me
  sprite(pMySpriteNum).triggerCallback = 0
end
```

trimWhiteSpace

Syntax `member(whichMember).trimWhiteSpace`

Description Cast member property. Determines whether the white pixels around the edge of a bitmap cast member are removed or left in place. This property is set when the member is imported. It can be changed in Lingo or in the Bitmap tab of the Property Inspector.

trimWhitespace()

Syntax `imageObject.trimWhitespace()`

Description Function; removes any white pixels that lie outside the minimum rectangle and returns the result in a new image object.

Example This statement trims the white space from member Flower and returns the new, trimmed image object in the variable `trimmedImage`.

```
trimmedImage = member("flower").image.trimWhitespace()
```

See also **crop() (member command)**

TRUE

Syntax `TRUE`

Description Constant; represents the value of a logically true expression, such as `2 < 3`. It has a traditional numerical value of 1, but any nonzero integer evaluates to `TRUE` in a comparison.

Example This statement turns on the `soundEnabled` property by setting it to `TRUE`:

```
the soundEnabled = TRUE
```

See also **FALSE**, **if**

tweened

Syntax

```
sprite(whichSprite).tweened
the tweened of sprite whichSprite
```

Description

Sprite property; determines whether only the first frame in a new sprite is created as a keyframe (`TRUE`), or whether all frames in the new sprite are created as keyframes (`FALSE`).

This property does not affect playback and is useful only during Score recording.

This property can be tested and set.

Example

When this statement is issued, newly created sprites in channel 25 have a keyframe only in the first frame of the sprite span:

```
sprite(25).tweened = 1
```

type (cast member property)

Syntax

```
member(whichCastMember).type
the type of member whichCastMember
member( whichCastMember, which castLib). type
member whichCastMember of castLib whichCast.type
the type of member whichCastMember of castLib whichCast
```

Description

Cast member property; indicates the specified cast member's type. This property replaces the `castType` property used in previous versions of Director.

The `type` member property can be one of the following values:

```
#animgif        #ole
#bitmap         #palette
#button         #picture
#cursor         #QuickTimeMedia
#digitalVideo   #script
#empty          #shape
#field          #sound
#filmLoop       #swa
#flash          #text (#richText is now obsolete)
#font           #transition
#movie          #vectorShape
```

This list includes those types of cast members that are available in Director and the Xtras that come with it. You can also define custom cast member types for custom cast members.

When a movie plays back as an applet, the `type` member property is valid only for cast member types that the player supports.

For movies created in Director 5 and 6, the `type` member property returns `#field` for field cast members and `#richText` for text cast members. However, field cast members originally created in Director 4 return `#text` for the member type, providing backward compatibility for movies that were created in Director 4.

This property can be tested but not set.

Example

The following handler checks whether the cast member Today's News is a field cast member and displays an alert if it is not:

```
on checkFormat
  if member("Today's News").type <> #field then alert \
  "Sorry, this cast member must be a field."
end
```

type (sprite property)

Syntax

```
sprite(whichSprite).type
the type of sprite whichSprite
```

Description

Sprite property; clears sprite channels during Score recording by setting the `type` sprite property value for that channel to 0.

Note: Switch the member of a sprite only to another member of the same type to avoid changing the sprite's properties when the member type is switched.

This property can be tested and set.

Example

This statement clears sprite channel 1 when issued during a Score recording session:

```
sprite(1).type = 0
```

union()

Syntax

```
rect(1).union(rect(2))
union (rect1, rect2)
```

Description Function; returns the smallest rectangle that encloses the two rectangles *rect1* and *rect2*.

Example This statement returns the rectangle that encloses the specified rectangles:

```
put union (rect (0, 0, 10, 10), rect (15, 15, 20, 20))
-- rect (0, 0, 20, 20)
```

or

```
put rect(0, 0, 10, 10).union(rect(15, 15, 20, 20))
--rect (0, 0, 20, 20)
```

See also **map()**,**rect()**

unLoad

Syntax
```
unLoad
unLoad theFrameNum
unLoad fromFrameNum, toFrameNum
```

Description Command; forces Director to clear the cast members used in a specified frame from memory. Director automatically unloads the least recently used cast members to accommodate `preLoad` commands or normal cast loading.

- When used without an argument, the `unLoad` command clears from memory the cast members in all the frames of a movie.
- When used with one argument, *`theFrameNum`*, the `unLoad` command clears from memory the cast members in that frame.
- When used with two arguments, *`fromFrameNum`* and *`toFrameNum`*, the `unLoad` command unloads all cast members in the range specified. You can specify a range of frames by frame numbers or frame labels.

Example This statement clears the cast members used in frame 10 from memory:

```
unLoad 10
```

Example This statement clears the cast members used from the frame labeled first to the frame labeled last:

```
unLoad "first", "last"
```

See also **preLoad (command)**, **preLoadMember**, **unLoadMember**, **purgePriority**

unLoadMember

Syntax

```
unLoadMember
member(whichCastMember). unLoad()
unLoadMember member whichCastMember
member(whichCastMember, whichCastLib). unLoad()
unLoadMember member whichCastMember of castLib whichCast
member(firstCastmember). unLoad(lastCastMember)
unLoadMember member firstCastMember, lastCastMember
```

Description

Command; forces Director to clear the specified cast members from memory. Director automatically unloads the least recently used cast members to accommodate `preLoad` commands or normal cast loading.

- When used without an argument, `unLoadMember` clears from memory the cast members in all the frames of a movie.
- When used with the arguments *`whichCastMember`* and *`whichCast`*, the `unLoadMember` command clears from memory the cast member name or number that you specify.
- When used with the arguments *`firstCastMember`* and *`lastCastMember`*, the `unLoadMember` command unloads all cast members in the range specified.

When used in a new movie with no loaded cast members, this command returns an error.

Cast members that you have modified during authoring or by setting `picture`, `pasteClipBoadInto`, and so on, cannot be unloaded.

Example

This statement clears from memory the cast member Screen1:

```
unLoadMember member "Screen1"
```

or

```
member("Screen1").unload()
```

Example

This statement clears from memory all cast members from cast member 1 to cast member Big Movie:

```
unLoadMember 1, member "Big Movie"
```

or

```
member(1).unload("Big Movie")
```

See also

preLoad (command), **preLoadMember**, **purgePriority**

unloadMovie

Syntax `unloadMovie *whichMovie*`

Description Command; removes the specified preloaded movie from memory. This command is useful in forcing movies to unload when memory is low.

You can use a URL as the file reference.

If the movie isn't already in RAM, the result is -1.

Example This statement checks whether the largest contiguous block of free memory is less than 100K and unloads the movie Parsifal if it is:

```
if (the freeBlock < (100 * 1024)) then unLoadMovie "Parsifal"
```

Example This statement unloads the movie at http://www.cbDemille.com/ SunsetBlvd.dir:

```
unLoadMovie "http://www.cbDemille.com/SunsetBlvd.dir"
```

updateFrame

Syntax `updateFrame`

Description Command; during Score generation only, enters the changes to the current frame that have been made during Score recording and moves to the next frame. Any objects that were already in the frame when the update session started remain in the frame. You must issue an `updateFrame` command for each frame that you are updating.

Example When used in the following handler, the `updateFrame` command enters the changes that have been made to the current frame and moves to the next frame each time Lingo reaches the end of the repeat loop. The number of frames is determined by the argument `numberOfFrames`.

```
on animBall numberOfFrames
  beginRecording
    horizontal = 0
    vertical = 300
    repeat with i = 1 to numberOfFrames
      go to frame i
      sprite(20).memberNum = member("Ball").number
      sprite(20).locH = horizontal
      sprite(20).locV = vertical
      sprite(20).type = 1
      sprite(20).foreColor = 255
      horizontal = horizontal + 3
      vertical = vertical + 2
      updateFrame
    end repeat
  endRecording
end
```

See also **beginRecording**, **endRecording**, **scriptNum**, **tweened**

updateLock

Syntax `the updateLock`

Description Movie property; determines whether the Stage is updated during Score recording (`FALSE`) or not (`TRUE`).

You can keep the Stage display constant during a Score recording session by setting `updateLock` to `TRUE` before Lingo updates the Score. If `updateLock` is `FALSE`, the Stage updates to show a new frame each time the frame is entered.

You can also use `updateLock` to prevent unintentional Score updating when leaving a frame, such as when you temporarily leave a frame to examine properties in another frame.

Although this property can be used to mask changes to a frame during run time, be aware that changes to field cast members appear immediately when the content is modified, unlike changes to location or members with other sprites, which are not updated until this property is turned off.

updateMovieEnabled

Syntax `the updateMovieEnabled`

Description System property; specifies whether changes made to the current movie are automatically saved (`TRUE`) or not saved (`FALSE`, default) when the movie branches to another movie.

This property can be tested and set.

Example This statement instructs Director to save changes to the current movie whenever the movie branches to another movie:

```
the updateMovieEnabled = TRUE
```

updateStage

Syntax `updateStage`

Description Command; redraws the Stage immediately instead of only between frames.

The `updateStage` command redraws sprites, performs transitions, plays sounds, sends a `prepareFrame` message (affecting movie and behavior scripts), and sends a `stepFrame` message (which affects `actorList`).

Example This handler changes the sprite's horizontal and vertical locations and redraws the Stage so that the sprite appears in the new location without having to wait for the playback head to move:

```
on moveRight whichSprite, howFar
  sprite(whichSprite).locH = sprite(whichSprite).locH + howFar
  updateStage
end moveRight
```

URL

Syntax

```
member(whichCastMember).URL
```

```
the URL of member whichCastMember
```

Description

Cast member property; specifies the URL for Shockwave Audio (SWA) and Flash movie cast members.

For Flash movie members, this property is synonymous with the `pathName` member property.

The `URL` property can be tested and set. For SWA members, it can be set only when the SWA streaming cast member is stopped.

Example

This statement makes a file on an Internet server the URL for SWA cast member Benny Goodman:

```
on mouseDown
  member("Benny Goodman").URL =
"http://audio.macromedia.com/samples/classic.swa"
end
```

URLEncode

Syntax `URLEncode(proplist_or_string {, serverOSString} {, characterSet})`

Description Function; returns the URL-encoded string for its first argument. Allows CGI parameters to be used in other commands. The same translation is done as for `postNetText` and `getNetText()` when they are given a property list.

Use the optional parameter *`serverOSString`* to encode any return characters in *`proplist_or_string`*. The value defaults to `"Unix"` but may be set to `"Win"` or `"Mac"` and translates any carriage returns in the *`proplist_or_string`* argument into those used on the server. For most applications, this setting is unnecessary because line breaks are usually not used in form responses.

The optional parameter *`characterSet`* applies only if the user is running on a Shift-JIS (Japanese) system. Its possible settings are `"JIS"`, `"EUC"`, `"ASCII"`, and `"AUTO"`. Retrieved data is converted from Shift-JIS to the named character set. Returned data is handled exactly as by `getNetText()` (converted from the named character set to Shift-JIS). If you use `"AUTO"`, the posted data from the local character set is not translated; the results sent back by the server are translated as they are for `getNetText()`. `"ASCII"` is the default if *`characterSet`* is omitted. `"ASCII"` provides no translation for posting or results.

Example In the following example, `URLEncode` supplies the URL-encoded string to a CGI query at the specified location.

```
URL = "http://aserver/cgi-bin/echoquery.cgi"
gotonetpage URL & "?" & URLEncode( [#name: "Ken", #hobby: "What?"]
)
```

See also **getNetText()**,**postNetText**

useAlpha

Syntax

```
member(whichCastMember).useAlpha
imageObject.useAlpha
```

Description

Bitmap cast member and image object property; for 32-bit cast members and image objects with alpha channel information, determines whether Director uses the alpha information when drawing the image onto the Stage (`TRUE`), or whether Director ignores the alpha information when drawing to the Stage (`FALSE`).

Example

This toggles the alpha channel of cast member "foreground" on and off.

```
member("foreground").useAlpha=not member("foreground").useAlpha
```

useFastQuads

Syntax `the useFastQuads`

Description Global property; when set to `TRUE`, Director uses a faster, less precise method for calculating quad operations. Fast quads calculations are good for simple rotation and skew sprite effects. Director's slower, default quad calculation method provides more visually pleasing results when using quads for distortion and other arbitrary effects. Defaults to `FALSE`.

Simple sprite rotation and skew operations always use the fast quad calculation method, regardless of this setting. Setting `the useFastQuads` to TRUE will not result in an increase in the speed of these simple operations.

Example This statement tells Director to use its faster quad calculation code for all quad operations in the movie:

```
the useFastQuads = TRUE
```

See also **quad**

useHypertextStyles

Syntax `member(whichTextMember).useHypertextStyles`

Description Text cast member property; controls the display of hypertext links in the text cast member.

When `useHypertextStyles` is `TRUE`, all links are automatically colored blue with underlines, and the cursor changes to a pointing finger when it is over a link.

Setting this property to `FALSE` turns off the automatic formatting and cursor change.

Example This behavior toggles the formatting of hypertext on and off in text cast member "myText":

```
on mouseUp
  member("myText").usehypertextStyles = not
member("myText").usehypertextStyles
end
```

userName

Syntax `the userName`

Description Movie property; a string containing the user name entered when Director was installed.

This property is available in the authoring environment only. It could be used in a movie in a window (MIAW) tool that is personalized to show the user's information.

Example This handler places the user's name and serial number in a display field when the window is opened. (A movie script in the MIAW is a good location for this handler.)

```
on prepareMovie
  displayString = the userName
  put RETURN&the organizationName after displayString
  put RETURN&the serialNumber after displayString
  member("User Info").text = displayString
end
```

See also **organizationName**, **serialNumber**, **window**

value()

Syntax `value(stringExpression)`

Description Function; returns the value of a string. The string can be any expression that Lingo can understand. When `value()` is called, Lingo parses through the *`stringExpression`* provided and returns its logical value.

Any Lingo expression that can be `put` in the Message window or set as the value of a variable can also be used with `value()`.

These two Lingo statements are equivalent:

```
put sprite(2).member.duration * 5
put value("sprite(2).member.duration * 5")
```

These two Lingo statements are also equivalent:

```
x = (the mouseH - 10) / (the mouseV + 10)
x = value("(the mouseH - 10) / (the mouseV + 10)")
```

Expressions that Lingo cannot parse will produce unexpected results, but will not produce Lingo errors. The result is the value of the inititial portion of the expression up to the first syntax error found in the string.

The `value()` function can be useful for parsing expressions input into text fields by end-users, string expressions passed to Lingo by Xtras, or any other expression you need to convert from a string to a Lingo value.

Keep in mind that there may be some situations where using `value()` with user-input can be dangerous, such as when the user enters the name of a custom handler into the field. This will cause the handler to be executed when it is passed to `value()`.

Do not confuse the actions of the value function with the `integer()` and `float()` functions.

Example This statement displays the numerical value of the string `"the sqrt of" && "2.0"`:

```
put value("the sqrt of" && "2.0")
```

The result is 1.4142.

Example This statement displays the numerical value of the string “penny”:

```
put value("penny")
```

The resulting display in the Message window is `VOID`, because the word *penny* has no numerical value.

Example You can convert a string that is formatted as a list into a true list by using this syntax:

```
myString = "[" & QUOTE & "cat" & QUOTE & ", " & QUOTE & "dog" &
QUOTE & "]"
myList = value(myString)
put myList
-- ["cat", "dog"]
```

This allows a list to be placed in a field or text cast member and then extracted and easily reformatted as a list.

Example This statement parses the string "3 5" and returns the value of the portion of the string that Lingo understands:

```
put value("3 5")
-- 3
```

See also **string()**, integer(), float()

version

Syntax `version`

Description Keyword; system variable that contains the version string for Director. The same string appears in the Macintosh Finder's Info window.

Example This statement displays the version of Director in the Message window:

```
put version
```

vertex

Syntax `member(whichVectorShapeMember).vertex[whichVertexPos ition]`

Description Chunk expression; enables direct access to parts of a vertex list of a vector shape cast member.

Use this chunk expression to avoid parsing different chunks of the vertex list. It's possible to both test and set values of the vertex list using this type of chunk expression.

Example The following code shows how to determine the number of vertex points in a member:

```
put member("Archie").vertex.count
-- 2
```

Example To obtain the second vertex for the member, you can use code like this:

```
put member("Archie").vertex[2]
-- point(66.0000, -5.0000)
```

Example You can also set the value in a control handle:

```
member("Archie").vertex[2].handle1 = point(-63.0000, -16.0000)
```

See also **vertexList**

vertexList

Syntax `member(whichVectorShapeMember).vertexList`

Description Cast member property; returns a linear list containing property lists, one for each vertex of a vector shape. The property list contains the location of the vertex and the control handle. There are no control handles if the location is (0,0).

Each vertex can have two control handles that determine the curve between this vertex and the adjacent vertexes. In `vertexList`, the coordinates of the control handles for a vertex are kept relative to that vertex, rather than absolute in the coordinate system of the shape. If the first control handle of a vertex is located 10 pixels to the left of that vertex, its location is stored as (-10, 0). Thus, when the location of a vertex is changed with Lingo, the control handles move with the vertex and do not need to be updated (unless the user specifically wants to change the location or size of the handle).

When modifying this property, be aware that you must reset the list contents after changing any of the values. This is because when you set a variable to the value of the property, you are placing a copy of the list, not the list itself, in the variable. To effect a change, use code like this:

```
- Get the current property contents
currVertList = member(1).vertexList
-- Add 25 pixels to the horizontal and vertical positions of the
first vertex in the list
currVertList[1] .vertex = currVertList[1] .vertex + point(25, 25)
-- Reset the actual property to the newly computed position
member(1).vertexList = currVertList
```

Example This statement displays the `vertextList` value for an arched line with two vertexes:

```
put member("Archie").vertexList
-- [[#vertex: point(-66.0000, 37.0000), #handle1: point(-70.0000,
-36.0000),  \
  #handle2: point(-62.0000, 110.0000)], [#vertex: point(66.0000,
-5.0000),  \
  #handle1: point(121.0000, 56.0000), #handle2: point(11.0000,
-66.0000)]]
```

See also **addVertex**, **count()**,**deleteVertex()**,**moveVertex()**,**moveVertexHandle()**,**originMode**, **vertex**

video

Syntax

```
member(whichCastMember).video
the video of member whichCastMember
```

Description

Digital video cast member property; determines whether the graphic image of the specified digital video cast member plays (`TRUE` or 1) or not (`FALSE` or 0).

Only the visual element of the digital video cast member is affected. For example, when `video` is set to `FALSE`, the digital video's soundtrack, if present, continues to play.

Example

This statement turns off the video associated with the cast member Interview:

```
member("Interview").video = FALSE
```

See also

setTrackEnabled, **trackEnabled**

videoForWindowsPresent

Syntax `the videoForWindowsPresent`

Description System property; indicates whether Video for Windows (AVI) is present on the computer.

This property can be tested but not set.

Example This statement checks whether Video for Windows is missing and branches the playback head to the Alternate Scene marker if it isn't:

```
if the videoForWindowsPresent= FALSE then go to "Alternate Scene"
```

See also **quickTimeVersion()**

viewH

Syntax

```
sprite(whichVectorOrFlashSprite).viewH
the viewH of sprite whichVectorOrFlashSprite
member(whichVectorOrFlashMember).viewH
the viewH of member whichVectorOrFlashMember
```

Description

Cast member and sprite property; controls the horizontal coordinate of a Flash movie and vector shape's view point, specified in pixel units. The values can be floating-point numbers. The default value is 0.

A Flash movie's view point is set relative to its origin point.

Setting a positive value for `viewH` shifts the movie to the left inside the sprite; setting a negative value shifts the movie to the right. Therefore, changing the `viewH` property can have the effect of cropping the movie or even of removing the movie from view entirely.

This property can be tested and set.

Note: This property must be set to the default value if the scaleMode property is set to #autoSize, or the sprite will not display correctly.

Example

This handler accepts a sprite reference as a parameter and moves the view of a Flash movie sprite from left to right within the sprite's bounding rectangle:

```
on panRight whichSprite
  repeat with i = 120 down to -120
    sprite(whichSprite).viewH = i
    updateStage
  end repeat
end
```

See also

scaleMode, **viewV**, **viewPoint**, **viewScale**

viewPoint

Syntax

```
sprite(whichVectorOrFlashSprite).viewPoint
the viewPoint of sprite whichVectorOrFlashSprite
member(whichVectorOrFlashMember).viewPoint
the viewPoint of member whichVectorOrFlashMember
```

Description

Cast member property and sprite property; controls the point within a Flash movie or vector shape that is displayed at the center of the sprite's bounding rectangle in pixel units. The values are integers.

Changing the view point of a cast member changes only the view of a movie in the sprite's bounding rectangle, not the location of the sprite on the Stage. The view point is the coordinate within a cast member that is displayed at the center of the sprite's bounding rectangle and is always expressed relative to the movie's origin (as set by the `originPoint`, `originH`, and `originV` properties). For example, if you set a Flash movie's view point at point (100,100), the center of the sprite is the point within the Flash movie that is 100 Flash movie pixel units to the right and 100 Flash movie pixel units down from the origin point, regardless of where you move the origin point.

The `viewPoint` property is specified as a Director point value: for example, point (100,200). Setting a Flash movie's view point with the `viewPoint` property is the same as setting the `viewH` and `viewV` properties separately. For example, setting the `viewPoint` property to point (50,75) is the same as setting the `viewH` property to 50 and the `viewV` property to 75.

Director point values specified for the `viewPoint` property are restricted to integers, whereas `viewH` and `viewV` can be specified with floating-point numbers. When you test the `viewPoint` property, the point values are truncated to integers. As a general guideline, use the `viewH` and `viewV` properties for precision; use the `originPoint` property for speed and convenience.

This property can be tested and set. The default value is point (0,0).

Note: This property must be set to the default value if the scaleMode property is set to #autoSize, or the sprite will not display correctly.

Example

This handler makes a specified Flash movie sprite move down and to the right in increments of five Flash movie pixel units:

```
on panAcross whichSprite
  repeat with i = 1 to 10
    sprite(whichSprite).viewPoint = sprite(whichSprite).viewPoint
+ point(i * -5, i * -5)
    updateStage
  end repeat
end
```

See also

scaleMode, **viewV**, **viewH**, **viewScale**

viewScale

Syntax

```
sprite(whichVectorOrFlashSprite).viewScale
the viewScale of sprite whichVectorOrFlashSprite
member(whichVectorOrFlashMember).viewScale
the viewScale of member whichVectorOrFlashMember
```

Description

Cast member property and sprite property; sets the overall amount to scale the view of a Flash movie or vector shape sprite within the sprite's bounding rectangle. You specify the amount as a percentage using a floating-point number. The default value is 100.

The sprite rectangle itself is not scaled; only the view of the cast member within the rectangle is scaled. Setting the `viewScale` property of a sprite is like choosing a lens for a camera. As the `viewScale` value decreases, the apparent size of the movie within the sprite increases, and vice versa. For example, setting `viewScale` to 200% means the view inside the sprite will show twice the area it once did, and the cast member inside the sprite will appear at half its original size.

One significant difference between the `viewScale` and `scale` properties is that `viewScale` always scales from the center of the sprite's bounding rectangle, whereas `scale` scales from a point determined by the Flash movie's `originMode` property.

This property can be tested and set.

Note: This property must be set to the default value if the scaleMode property is set to #autoSize, or the sprite will not display correctly.

Example

This sprite script sets up a Flash movie sprite and doubles its view scale:

```
on beginSprite me
  sprite(spriteNum of me).viewScale = 200
end
```

See also

scaleMode, **viewV**, **viewPoint**, **viewH**

viewV

Syntax

```
sprite(whichVectorOrFlashSprite).viewV
the viewV of sprite whichVectorOrFlashSprite
member(whichVectorOrFlashMember).viewV
the viewV of member whichVectorOrFlashMember
```

Description

Cast member and sprite property; controls the vertical coordinate of a Flash movie and vector shape's view point, specified in pixel units. The values can be floating-point numbers. The default value is 0.

A Flash movie's view point is set relative to its origin point.

Setting a positive value for `viewV` shifts the movie up inside the sprite; setting a negative value shifts the movie down. Therefore, changing the `viewV` property can have the effect of cropping the movie or even of removing the movie from view entirely.

This property can be tested and set.

Note: This property must be set to the default value if the scaleMode property is set to #autoSize, or the sprite will not display correctly.

Example

This handler accepts a sprite reference as a parameter and moves the view of a Flash movie sprite from the top to the bottom within the sprite's bounding: rectangle.

```
on panDown whichSprite
  repeat with i = 120 down to -120
    sprite(whichSprite).viewV = i
    updateStage
  end repeat
end
```

See also

scaleMode, **viewV**, **viewPoint**, **viewH**

visible (sprite property)

Syntax

```
sprite(whichSprite).visible
the visible of sprite whichSprite
```

Description

Sprite property; determines whether the sprite specified by *whichSprite* is visible (TRUE) or not (FALSE). This property affects all sprites in the channel, regardless of their position in the Score.

Note: Setting the visible property of a sprite channel to FALSE makes the sprite invisible and prevents only the mouse-related events from being sent to that channel. The beginSprite, endSprite, prepareFrame, enterFrame, and exitFrame events continue to be sent regardless of the sprite's visibility setting. Clicking the Mute button on that channel in the Score, however, will set the visible property to FALSE and prevent all events from being sent to that channel. Muting disables a channel, while setting a sprite's visible property to FALSE merely affects a graphic property.

This property can be tested and set. If set to FALSE, this property will not automatically reset to TRUE when the sprite ends. You must set the visible property of the sprite to TRUE in order to see any other members using that channel.

Example

This statement makes sprite 8 visible:

```
sprite(8).visible = TRUE
```

visible (window property)

Syntax

```
window whichWindow.visible
the visible of window whichWindow
```

Description

Window property; determines whether the window specified by *whichWindow* is visible (`TRUE`) or not (`FALSE`).

This property can be tested and set.

Example

This statement makes the Control Panel window visible:

```
window("Control Panel").visible = TRUE
```

VOID

Syntax `VOID`

Description Constant; indicates the value `VOID`.

Example This statement checks whether the value in the variable `currentVariable` is `VOID`:

```
if currentVariable = VOID then
  put "This variable has no value"
end if
```

See also **voidP()**

voidP()

Syntax `voidP(variableName)`

Description Function; determines whether the variable specified by *`variableName`* has any value. If the variable has no value or is `VOID`, this function returns `TRUE`. If the variable has a value other than `VOID`, this function returns `FALSE`.

Example This statement checks whether the variable `answer` has an initial value:

```
put voidP(answer)
```

See also **ilk()**,**VOID**

volume (cast member property)

Syntax

```
member(whichCastMember).volume
the volume of member whichCastMember
```

Description

Shockwave Audio (SWA) cast member property; determines the volume of the specified SWA streaming cast member. Values range from 0 to 255.

This property can be tested and set.

Example

This statement sets the volume of an SWA streaming cast member to half the possible volume:

```
member("SWAfile").volume = 128
```

volume (sound channel)

Syntax `the volume of sound` *`channelNum`*

Description Sound channel property; determines the volume of the sound channel specified by *`channelNum`*. Sound channels are numbered 1, 2, 3, and so on. Channels 1 and 2 are the channels that appear in the Score.

The value of the `volume` sound property ranges from 0 (mute) to 255 (maximum volume). A value of 255 indicates the full volume set for the machine, as controlled by the `soundLevel` property, and lower values are scaled to that total volume. This property allows several channels to have independent settings within the available range.

The lower the value of the `volume` sound property, the more static or noise you're likely to hear. Using `soundLevel` may produce less noise, although this property offers less control.

This property does not support dot syntax. Use the syntax exactly as shown here.

To see an example of `volume (sound channel)` used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement sets the volume of sound channel 2 to 130, which is a medium sound level setting:

```
sound(2).volume = 130
```

See also **fadeIn()**, **fadeOut()**,**soundEnabled**, **soundLevel**, **fadeTo()**, **pan (sound property)**

volume (sprite property)

Syntax

```
sprite(whichSprite).volume
the volume of sprite whichSprite
```

Description

Sprite property; controls the volume of a digital video movie cast member specified by name or number. The values range from 0 to 256. Values of 0 or less mute the sound. Values exceeding 256 are loud and introduce considerable distortion.

Example

This statement sets the volume of the QuickTime movie playing in sprite channel 7 to 256, which is the maximum sound volume:

```
sprite(7).volume = 256
```

See also

soundLevel

warpMode

Syntax

```
sprite(whichQTVRSprite).warpMode
warpMode of sprite whichQTVRSprite
```

Description

QuickTime VR sprite property; specifies the type of warping performed on a panorama.

Possible values are `#full`, `#partial`, and `#none`.

This property can be tested and set. When tested, if the values for the static and motion modes differ, the property's value is the value set for the current mode. When set, the property determines the warping for both the static and motion modes.

Example

This sets the warpMode of sprite 1 to #full:

```
sprite(1).warpMode = #full
```

width

Syntax

```
member(whichCastMember).width
the width of member whichCastMember
imageObject.width
sprite(whichSprite).width
the width of sprite whichSprite
```

Description

Cast member, image object and sprite property; for vector shape, Flash, animated GIF, bitmap, and shape cast members, determines the width, in pixels, of the cast member specified by *whichCastMember*, *imageObject* or *whichSprite*. Field and button cast members are not affected.

For cast members and image objects, this property can be tested; for sprites, this property can be both tested and set.

Setting this property on bitmap sprites automatically sets the sprite's `stretch of sprite` property to `TRUE`.

Example

This statement assigns the width of member 50 to the variable `height`:

```
height = member(50).width
```

Example

This statement sets the width of sprite 10 to 26 pixels:

```
sprite(10).width = 26
```

Example

This statement assigns the width of sprite number i + 1 to the variable `howWide`:

```
howWide = sprite(i + 1).width
```

See also

height

window

Syntax `window whichWindow`

Description Keyword; refers to the movie window—a window that contains a Director movie—specified by *whichWindow*.

Windows that play movies are useful for creating floating palettes, separate control panels, and windows of different shapes. Using windows that play movies, you can have several movies open at once and allow them to interact.

Example This statement opens a window named Navigation:

```
open window "Navigation"
```

Example This statement moves the Navigation window to the front:

```
moveToFront window "Navigation"
```

or

```
window("Navigation").moveToFront()
```

See also **close window**, **moveToBack**, **moveToFront**, **open window**

windowList

Syntax `the windowList`

Description System property; displays a list of references to all known movie windows.

Example This statement displays a list of all known movie windows in the Message window:

```
put the windowList
```

Example This statement clears `windowList`:

```
the windowList = [ ]
```

See also **windowPresent()**

windowPresent()

Syntax `windowPresent(windowName)`

Description Function; indicates whether the object specified by *windowName* is running as a movie in a window (`TRUE`) or not (`FALSE`). If a window had been opened, `windowPresent` remains `TRUE` for the window until the window has been removed from the `windowList` property.

The *windowName* argument must be the window's name as it appears in the `windowList` property.

Example This statement tests whether the object myWindow is a movie in a window (MIAW) and then displays the result in the Message window:

```
put windowPresent(myWindow)
```

See also the windowList

windowType

Syntax
```
window whichWindow.windowType
the windowType of window whichWindow
```

Description Window property; controls the display style of the window specified by *whichWindow*, as follows:

- 0—Movable, sizable window without zoom box
- 1—Alert box or modal dialog box
- 2—Plain box, no title bar
- 3—Plain box with shadow, no title bar
- 4—Movable window without size box or zoom box
- 5—Movable modal dialog box
- 8—Standard document window
- 12—Zoomable, nonresizable window
- 16—Rounded-corner window
- 49—Floating palette, during authoring (in Macintosh projectors, the value 49 specifies a stationary window)

You can set this property before opening the window. Numbers 6, 7, 9, 10, 11, 13, 14, 15, and 17 through 48 have no effect when used as the `windowType` value.

You can change the `windowType` setting after a window has been opened, but a delay may occur while the window is redrawn.

If no `windowType` setting is specified, a value of 0 is used.

In Microsoft Windows, these numbers create windows with the same functionality jsut described, but with a Windows appearance. Other values for `windowType` are possible, but use them with caution, because some modal windows can be exited only when you restart the computer.

Example This statement sets the value of the display style of the window named Control Panel to 8:

```
window("Control Panel").windowType = 8
```

word...of

Syntax

```
member(whichCastMember).word[whichWord]
textMemberExpression.word[whichWord]
chunkExpression.word[whichWord]
word whichWord of fieldOrStringVariable
fieldOrStringVariable. word[whichWord]
textMemberExpression.word[firstWord..lastWord]
member(whichCastMember).word[firstWord..lastWord]
word firstWord to lastWord of chunkExpression
chunkExpression.word[whichWord..lastWord]
```

Description

Chunk expression; specifies a word or a range of words in a chunk expression. A word chunk is any sequence of characters delimited by spaces. (Any nonvisible character, such as a tab or carriage return, is considered a space.)

The expressions *whichWord*, *firstWord*, and *lastWord* must evaluate to integers that specify a word in the chunk.

Chunk expressions refer to any character, word, item, or line in any source of characters. Sources of characters include field and text cast members and variables that hold strings.

To see an example of `word...of` used in a completed movie, see the Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

These statements set the variable named `animalList` to the string "fox dog cat" and then insert the word *elk* before the second word of the list:

```
animalList = "fox dog cat"
put "elk" before animalList.word[2]
```

The result is the string "fox elk dog cat".

Example

This statement tells Director to display the fifth word of the same string in the Message window:

```
put "fox elk dog cat".word[5]
```

Because there is no fifth word in this string, the Message window displays two quotation marks (""), which indicate an empty string.

See also

char...of, **line...of**, **item...of**, **count()**,**number (words)**

wordWrap

Syntax
```
member(whichCastMember).wordWrap
the wordWrap of member whichCastMember
```

Description
Cast member property; determines whether line wrapping is allowed (`TRUE`) or not (`FALSE`).

Example
This statement turns line wrapping off for the field cast member Rokujo:

```
member("Rokujo").wordWrap = FALSE
```

xtra

Syntax `xtra whichXtra`

Description Function; returns an instance of the scripting Xtra specified by *whichXtra*.

To see an example of `xtra` used in a completed movie, see the Read and Write Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement uses the `new()` function to create a new instance of the Multiuser Xtra and assigns it to the variable `tool`:

```
tool = new(xtra "Multiuser")
```

See also **new()**

xtraList

Syntax `the xtraList`

Description System property; displays a linear property list of all available Xtras and their file versions. This property is useful when the functionality of a movie depends on a certain version of an Xtra.

#name	Specifies the file name of the Xtra on the current platform. It is possible to have a list without a `#name` entry, such as when the Xtra exists only on one platform.
#version	Specifies the same version number that appears in the Properties dialog box (Windows) or Info window (Macintosh) when the file is selected on the desktop. An Xtra may not necessarily have a version number.

There are two possible properties that can appear in `XtraList`:

Example This statement displays the `xtraList` in the Message window:

```
put the xtraList
```

Example This handler checks the version of a given Xtra:

```
-- This handler checks all the available Xtras to return the
version of the requested Xtra
-- The XtraFileName may be only a partial match if desired
-- Use the full file name for more exact usage
-- The string returned is either the version property for the Xtra
or an empty string
-- It may be empty if either the version property doesn't exist or
the Xtra is not found in the available list
on GetXtraVersion XtraFileName
  --  Get the entire list of Xtras and their information
  listOfXtras = the xtraList
  --  Initialize the local variable to contain the version
  theVersion = ""
  --  Iterate through all the Xtras listed
  repeat with currentXtra in listOfXtras
    --    If the current Xtra's name contains the Xtra passed in,
then check for the version
    if currentXtra.name contains XtraFileName then
      --      First determine if the version property exists for
that Xtra
      versionFlag = getaProp(currentXtra, #version)
      --      If the version property is not VOID, then set the
local variable to
    -- the string contained in that property value
      if not voidP(versionFlag) then
           theVersion = currentXtra.version
      end if
    end if
  end repeat
--  Return the version information found or an empty string
  return theVersion
end
```

See also **movieXtraList**, **getaProp**

xtras

See **number of xtras**

zoomBox

Syntax `zoomBox startSprite, endSprite {,delayTicks}`

Description Command; creates a zooming effect, like the expanding windows in the Macintosh Finder. The zoom effect starts at the bounding rectangle of *`startSprite`* and finishes at the bounding rectangle of *`endSprite`*. The `zoomBox` command uses the following logic when executing:

1. Look for *`endSprite`* in the current frame: otherwise,
2. Look for *`endSprite`* in the next frame.

Note, however, that the `zoomBox` command does not work for *`endSprite`* if it is in the same channel as *`startSprite`*.

The *`delayTicks`* variable is the delay in ticks between each movement of the zoom rectangles. If *`delayTicks`* is not specified, the delay is 1.

Example This statement creates a zoom effect between sprites 7 and 3:

```
zoomBox 7, 3
```

zoomWindow

Syntax

```
on zoomWindow
  statement(s)
end
```

Description

System message and event handler; contains statements that execute whenever a movie running as a movie in a window (MIAW) is resized. This happens when the user clicks the Minimize or Maximize button (Windows) or the Zoom button (Macintosh). The operating system determines the dimensions after resizing the window.

An `on zoomWindow` event handler is a good place to put Lingo that rearranges sprites when window dimensions change.

Example

This handler moves sprite 3 to the coordinates stored in the variable `centerPlace` when the window that the movie is playing in is resized:

```
on zoomWindow
  centerPlace = point(10, 10)
  sprite(3).loc = centerPlace
end
```

See also

drawRect, **sourceRect**, **on resizeWindow**

Lingo by feature: Overview

This appendix lists various Director features and the corresponding Lingo elements that you can use to implement those features. Click an element to go to that element's *Lingo Dictionary* entry.

For information about Lingo used for the Multiuser server Xtra, see **Using the Shockwave Mulituser Server** in the Director Support Center. For information about Lingo used for XML parsing, see **Using the XML Parser** in the Director Support Center.

Animated GIFs

These terms are useful for working with animated GIFs:

directToStage
frameRate
linked
moviePath
pause (movie playback)
playBackMode
resume sprite
rewind sprite

Animation

These terms are useful for creating animation with Lingo:

blend
ink
loc
locH
locV
member (sprite property)
regPoint
tweened

Behaviors

These terms are useful for authoring behaviors and using behaviors while the movie plays.

Authoring behaviors

Use these terms to set up behaviors and the behavior's Parameters dialog box:

ancestor
on getBehaviorDescription
on runPropertyDialog
on getPropertyDescriptionList
on getBehaviorTooltip
property
on isOKToAttach

Sending messages to behaviors

Use these commands to send messages to behaviors attached to sprites:

call
sendSprite
callAncestor
sendAllSprites

Identifying behaviors

Use these terms to identify the behaviors attached to sprites:

currentSpriteNum
scriptInstanceList
me
spriteNum

Bitmaps

These terms are useful for working with bitmaps.

Bitmap properties

Use these terms to check and set bitmap properties:

alphaThreshold
backColor
blend
depth
dither
trimWhiteSpace
imageQuality
movieImageQuality
foreColor
palette
picture (cast member property)
pictureP()
rect (member)
imageCompression
movieImageCompression

Alpha channel

Use these terms to control alpha channel effects:

alphaThreshold
depth
createMask()
extractAlpha()
dither
useAlpha
createMatte()
setAlpha()

Image objects

Use these terms to create and control image objects:

copyPixels()
crop() (image object command)
draw
duplicate() (image function)
getPixel()
fill()
image
image()
rect (image)
setPixel()

Cast members

These terms are useful for working with cast members.

Creating cast members

Use **importFileInto** and **new()**to create cast members.

Authoring

Use **duplicate member**, **erase member**, and **pasteClipBoardInto** to work with cast members during authoring.

Graphic cast members

Use these terms to check and set the images assigned to graphic cast members:

center
crop (cast member property)
depth
media
palette
picture (cast member property)
pictureP()
regPoint

General cast member properties

Use these terms to check and set cast member properties:

fileName (cast member property)
media
modified
name (cast member property)
number (cast member property)
preLoadMode
type (cast member property)
URL

Graphic cast member dimensions

Use **height**, **rect (member)**, and **width** to check and set dimensions for graphic cast members.

Casts

Theses terms are useful for working with casts.

Loading casts

Use **preLoadMode** to check and set when Director preloads a cast.

Cast properties

Use these terms to specify cast properties:

castLib

fileName (cast property)

name (cast property)

number (cast property)

number (system property)

Cast management

Use these terms to manage casts:

activeCastLib

duplicate member

erase member

findEmpty()

move member

number of members

pasteClipBoardInto

save castLib

selection (cast property)

Computer and operating system

Use these terms to check and control the computer:

beep
beepOn
cpuHogTicks
emulateMultiButtonMouse
floatPrecision
freeBlock()
freeBytes()
maxInteger
multiSound
romanLingo

Operating system control

Use **restart** and **shutDown** to control the operating system:

Data types

These terms are useful for specifying data types:

 # (symbol)
float()
floatP()
integer()
integerP()
objectP()
string()
stringP()
symbol()
symbolP()
VOID
voidP()

Digital video

These terms are useful for working with AVI and QuickTime digital video.

controller
digitalVideoTimeScale
digitalVideoType
directToStage
duration
frameRate
loop (cast member property)
movieRate
movieTime
pausedAtStart
quickTimeVersion()
timeScale
trackEnabled
trackNextKeyTime
trackNextSampleTime
trackPreviousKeyTime
trackPreviousSampleTime
trackStartTime (sprite property)
trackStartTime (cast member property)
trackStopTime (sprite property)
trackStopTime (cast member property)
trackText
trackType (cast member property)
trackType (sprite property)
trackCount (cast member property)
trackCount (sprite property)
video
videoForWindowsPresent

QuickTime

Use these terms to work with QuickTime:

enableHotSpot
fieldOfView
getHotSpotRect()
hotSpotExitCallback
hotSpotEnterCallback
invertMask
isVRMovie
loopBounds
mask
motionQuality
mouseLevel
node
nodeEnterCallback
nodeType
nudge
pan (QTVR property)
ptToHotSpotID()
quickTimeVersion()
rotation
scale
swing()
staticQuality
tilt
translation
triggerCallback
warpMode

nodeExitCallback

Events

Use these event handlers for Lingo that runs when a specific event occurs:

on activateWindow
close window
on cuePassed
on deactivateWindow
on enterFrame
on EvalScript
on exitFrame
on idle
on keyDown
on keyUp
on mouseDown (event handler)
on mouseEnter
on mouseLeave
on mouseUp (event handler)
on stopMovie
on endSprite
on moveWindow
on mouseWithin
open window
on prepareFrame
on prepareMovie
on resizeWindow
on mouseUpOutside
on rightMouseDown (event handler)
on rightMouseUp (event handler)
on startMovie
on stepFrame
on streamStatus
on timeOut
on **zoomWindow**
on beginSprite
on hyperlinkClicked

Use the **pass** and **stopEvent** commands to override the way that Director passes messages along the message hierarchy.

External files

These terms are useful for working with external files.

Path and file names

Use these terms to check and set path and file names:

@ (pathname)
applicationPath
fileName (cast property)
fileName (cast member property)
getNthFileNameInFolder()
moviePath
searchCurrentFolder
URL

Obtaining external media

Use these terms to obtain external media:

downloadNetThing
importFileInto
preloadNetThing()

Managing external files

Use these terms to manage external files:

closeXlib
open
openXlib
showXlib
sound playFile

Flash

These terms are useful for working with Flash cast members:

actionsEnabled
bufferSize
bytesStreamed
clearError
defaultRect
directToStage
findLabel()
flashRect
frame (sprite property)
frameCount
frameReady()
getFlashProperty()
getVariable()
hitTest()
imageEnabled
loop (keyword)
obeyScoreRotation
originMode
originV
pausedAtStart
playBackMode
play
quality
rotation
scale
setVariable()
sound
state
stop (Flash)
streamSize
the soundMixMedia
broadcastProps
buttonsEnabled
centerRegPoint
clickMode
defaultRectMode
eventPassMode
fixedRate
flashToStage()
frame() (function)
frameRate
getError()
getFrameLabel
goToFrame
hold
linked
mouseOverButton
originH
originPoint
pathName (movie property)
percentStreamed
playing
posterFrame
rewind sprite
scaleMode
setFlashProperty()
showProps()
stageToFlash()
static
streamMode
stream
URL

viewH

viewPoint

viewScale

viewV

volume (cast member property)

Frames

These Lingo terms let you work with frames.

Frame events

Use the **on enterFrame**, **on exitFrame**, and **on prepareFrame** event handlers to contain Lingo that runs at specific events during the course of a frame:

Frame properties

Use these Lingo terms to check and set frame properties:

frameLabel
framePalette
frameScript
frameSound1
frameSound2
the_markerList
frameTempo
frameTransition
label()
labelList
marker()

Interface elements

These terms are useful for working with interface elements.

Menus

Use these terms to create menus:

enabled
installMenu
menu
name (menu property)
name (menu item property)
number (menu items)
number (menus)
script

Buttons and check boxes

Use these terms to specify buttons and check boxes:

alert
buttonStyle
buttonType
checkBoxAccess
checkBoxType
checkMark
hilite (cast member property)

Keys

These terms are useful for Lingo related to using the keyboard.

Identifying keys

Use these terms to identify keys:

charToNum()
commandDown
controlDown
key()
keyCode()
keyPressed()
mouseChar
numToChar()
optionDown
shiftDown

Keyboard interaction

Use **keyPressed()**,**lastEvent()**, and **lastKey** to detect what the user types at the keyboard.

Keyboard events

Use these terms to set up handlers that respond to pressing keys:

on keyDown
on keyUp
flushInputEvents
keyDownScript
keyUpScript

Lingo

These terms are important language elements that you use to construct scripts.

Booleans

Use these terms to test whether a condition exists:

- **FALSE** (0 is the numerical equivalent of FALSE.)
- **TRUE** (1 is the numerical equivalent of TRUE.)
- **not**
- **or**

Script control

Use these terms to control how a script executes:

abort	**pass**
do	**result**
exit	**scriptsEnabled**
halt	**scriptText**
nothing	**stopEvent**

Code structures

Use **if** to create if..then statements.

Use **case**, **end case**, and **otherwise** in case statements.

Use these terms for repeat loops:

end repeat	**repeat with**
exit repeat	**repeat with...down to**
next repeat	**repeat with...in list**
repeat while	

Syntax elements

Use these terms as part of Lingo's syntax:

# (symbol)	**member (keyword)**
" (string)	**of**
/ (continuation)	**or**
-- (comment)	**property**
() (parentheses)	**sprite**

castLib

end

global

the

window

Lists

These terms are useful for working with lists.

Creating lists

Use **[ ] (list)**, **duplicate() (list function)**, or **list()**to create a list.

Adding list items

Use these terms to add items to a list:

[ ] (bracket access)
add
addAt
addProp
append

Deleting list items

Use these terms to delete items from a list:

deleteAll
deleteAt
deleteOne
deleteProp

Retrieving values from a list

Use these terms to retrieve values from a list:

[ ] (bracket access)
deleteProp
deleteProp
getLast()
getOne()
getPos()
getProp()
getPropAt()

Getting information about lists

Use these terms to get information about lists:

count()
findPos
findPosNear
ilk()
listP()
max()
min
param()
paramCount()

Setting values in a list

Use these terms to set values in a list:

[ ] (bracket access)

setAt

setaProp

setProp

Media synchronization

Use these terms to synchronize animation and sound:

cuePointNames

cuePointTimes

mostRecentCuePoint

on cuePassed

isPastCuePoint()

Memory management

These terms are useful for determining memory requirements and controlling when the movie loads and unloads cast members.

Use the **on idle** event handler for Lingo that runs when the movie is idle.

Idle loading

Use these terms to control idle loading:

cancelIdleLoad
finishIdleLoad
idleHandlerPeriod
idleLoadDone()
idleLoadPeriod
idleLoadTag
idleReadChunkSize
netThrottleTicks

Preloading and querying media

Use these terms to load media into memory and check whether media is available:

frameReady()
loaded
mediaReady
preLoad (command)
preLoad (cast member property)
preLoadBuffer member
preLoadEventAbort
preLoadMode
preloadNetThing()
preLoadMember
preLoadMovie
preLoadRAM
purgePriority
unLoad
unLoadMember
unloadMovie

Available memory

Use these terms to check how much memory is available:

freeBlock()
freeBytes()
memorySize
movieFileFreeSize
movieFileSize

Memory requirements

Use **ramNeeded()**and **size** to determine how much memory required for a cast member or a range of frames.

Message window

Use these terms to work in the Message window:

put

showXlib

trace

traceLoad

traceLogFile

appMinimize

Monitor

Use **colorDepth**, **deskTopRectList**, and **switchColorDepth** to check and control the monitor.

Mouse interaction

These terms are useful for Lingo related to using the mouse.

Mouse clicks

Use these terms to detect what the user does with the mouse:

clickOn
doubleClick
emulateMultiButtonMouse
lastClick()
lastEvent()
lastRoll
mouseChar
on mouseDown (event handler)
mouseH
mouseItem
mouseLevel
mouseLine
mouseLoc
mouseMember
mouseOverButton
on mouseUp (event handler)
mouseV
mouseWord
on rightMouseDown (event handler)
on rightMouseUp (event handler)
rollOver()
stillDown

Mouse events

Use these terms to set up handlers that respond to mouse events:

mouseDownScript
mouseUpScript
on mouseDown (event handler)
on mouseEnter
on mouseLeave
on mouseUp (event handler)
on mouseUpOutside
on mouseWithin
on rightMouseDown (event handler)
on rightMouseUp (event handler)

Cursor control

Use **cursor (command)**, **cursor (sprite property)**, and **cursorSize** to control the cursor.

Movie in a window

These terms are useful for working with movies in a window.

Movie in a window events

Use these event handlers to contain Lingo that you want to run in response to events in a movie in a window:

on activateWindow
on closeWindow
on moveWindow
on openWindow
on resizeWindow
zoomWindow

Opening and closing movies in a window

Use these terms for opening and closing windows:

close window
forget window
open window
windowList

Window appearance

Use these terms to check and set the appearance of a movie's window:

drawRect
fileName (window property)
frontWindow
modal
moveToBack
moveToFront
name (window property)
rect (window)
sourceRect
tell
title
titleVisible
visible (window property)
windowPresent()
windowType
appMinimize

Communication between movies

Use the **tell** command to send messages between movies.

Movies

These terms are useful for managing movies.

Stopping movies

Use these terms to stop or quit the movie or projector:

exitLock
halt
pauseState
quit
restart
shutDown

Movie information

Use these terms to obtain information about the movie and the movie's environment:

environment
lastFrame
movie
movieFileFreeSize
movieFileSize
movieName
moviePath
number (system property)
runMode
safePlayer
version
movieFileVersion

Source control

Use these terms to manage Director projects with more than one person working on them:

comments
modifiedBy
linkAs()
creationdate
modifiedDate
seconds

Saving movies

Use **saveMovie** and **updateMovieEnabled** to save changes to a movie.

Error checking

Use the `alertHook` event to post alerts that describe errors in a projector.

Movie events

Use the **on prepareMovie**, **on startMovie**, and **on stopMovie** event handlers for Lingo that

responds to movie events.

Shockwave Multiuser Server

For information about Lingo for use with the Shockwave Multiuser Server, see **Using the Shockwave Mulituser Server** in the Director Support Center.

Navigation

Use these terms to jump to different locations:

delay
go
go loop
go next
go previous
goToFrame
gotoNetMovie
gotoNetPage
play
play done

Network Lingo

These terms are useful for working with the network.

Downloading and streaming media

Use these terms to obtain or stream media from the network:

downloadNetThing (For projectors and authoring only)

getNetText()

gotoNetMovie

gotoNetPage

postNetText

preloadNetThing()

Checking availability

Use **frameReady()**and **mediaReady** to check whether specific media is completely downloaded.

Using network operations

Use these terms to check the progress of a network operation or get information regarding network media:

getStreamStatus()

getLatestNetID

netAbort

netDone()

netError()

netPresent

netLastModDate()

netMIME()

netTextResult()

on streamStatus

proxyServer

tellStreamStatus()

Working with the local computer

Use these terms to work with the user's computer:

browserName()

cacheDocVerify() (For projectors and authoring only)

cacheSize() (For projectors and authoring only)

clearCache (For projectors and authoring only)

getPref()

setPref

Browsers

Use **on EvalScript**, **externalEvent**, and **netStatus** to interact with browsers.

Accessing EMBED and OBJECT tag parameters

Use **externalParamCount()**,**externalParamName()**, and **externalParamValue()**to access EMBED and OBJECT parameter tags:

Operators

These terms are operators available in Lingo.

Math operators

Use these terms for math statements:

*** (multiplication)**

/ (division)

+ (addition)

- (minus)

= (equals)

<> (not equal)

> (greater than)

>= (greater than or equal to)

< (less than)

<= (less than or equal to)

Comparison operators

Use **and**, **not**, and **or** to compare expressions:

Palettes and color

Use these terms to check and set palettes for movies and for cast members:

color()
depth
palette
paletteMapping
puppetPalette

Parent scripts

Use these terms to work with parent scripts and child objects:

actorList
ancestor
new()
handlers()
property
on stepFrame
handler()
rawNew()

Points and rects

These terms are useful for checking and setting points and rectangles.

inflate

inside()

intersect()

map()

offset() (rectangle function)

point()

quad

rect()

rect (sprite)

sourceRect

union()

For Lingo that controls a sprite's bounding rectangle, see **Sprite dimensions** in this appendix.

Projectors

These terms are useful for working with projectors:

alertHook

environment

editShortCutsEnabled

platform

runMode

Puppets

Use this Lingo to control the `puppet` of sprites and effects channels:

puppet
puppetPalette
puppetSound
puppetSprite
puppetTempo
puppetTransition
updateStage

Score

The following terms let you work with the Score.

Score properties

Use **lastFrame**, **score**, and **scoreSelection** to work with the movie's Score.

Score generation

Use these terms to generate score from Lingo:

beginRecording
clearFrame
deleteFrame
duplicateFrame
endRecording
insertFrame
scoreColor
scoreSelection
scriptNum
scriptType
tweened
updateFrame
updateLock

Shapes

Use these Lingo terms to work with shapes:

filled

lineDirection

lineSize

pattern

shapeType

Shockwave audio

Use these terms to check, stream, and play Shockwave audio sounds:

bitRate

bitsPerSample

copyrightInfo

duration

getError()

getErrorString()

numChannels

pause (movie playback)

percentPlayed

percentStreamed

play member

preLoadBuffer member

preLoadTime

sampleRate

soundChannel

state

stop member

streamName

URL

volume (cast member property)

Sound

These terms are useful for playing sounds.

Sound information

Use these terms for information about a sound:

channelCount
isBusy()
sampleCount
soundBusy()
soundEnabled
sound
status

Playing sound

Use these terms to control how sound plays:

breakLoop()
elapsedTime
endTime
fadeIn()
fadeOut()
fadeTo()
getPlayList()
loopCount
loopEndTime
loopsRemaining
loopStartTime
member (sound property)
pan
pause (sound playback)
playNext()
puppetSound
queue()
rewind()
setPlayList()
sound close
sound fadeIn
sound fadeOut
sound playFile
sound stop
stop() (sound)

Sprites

These Lingo terms are for sprites.

Sprite events

Use the **on beginSprite** and **on endSprite** event handlers to contain Lingo that you want to run when a sprite begins or ends.

Assigning cast members to sprites

Use **castLibNum**, **member (sprite property)**, or **memberNum** to specify a sprite's cast member.

Sprite rotation

Use the **rotation** sprite property to rotate sprites.

Dragging sprites

Use these terms to set how the user can drag sprites:

constrainH()
constrainV()
constraint
moveableSprite
sprite...intersects
sprite...within

Sprites and Lingo

Use these terms to manage how Lingo controls sprites:

puppetSprite
puppet
scriptNum
scriptInstanceList
spriteNum
sendSprite
sendAllSprites

Drawing sprites on the Stage

Use these terms to control how Director draws a sprite on the Stage:

blend
flipH
flipV
ink
quad
skew
trails
tweened
updateStage
visible (sprite property)

rotation

Sprite dimensions

Use these terms to check and set the size of a sprite's bounding rectangle:

bottom
height
left
quad
right
top
width
zoomBox

You can also manipulate a sprite's bounding rectangle with Lingo for rectangles. See **Points and rects**.

Sprite locations

Use the **loc**, **locH**, and **locV** sprite properties to check and set sprite locations.

Sprite color

Use these terms to check and set a sprite's color:

backColor
bgColor
color (sprite property)
foreColor

Stage

These terms are useful for controlling the Stage and determining its size and location:

centerStage

fixStageSize

picture (window property)

stage

stageBottom

stageColor

stageLeft

stageRight

stageTop

updateStage

Tempo

Use the **puppetTempo** command to control a movie's tempo.

Text

These terms are useful for working with text, strings, and fields.

Manipulating strings

Use these terms to manipulate strings:

& (concatenator)
&& (concatenator)
delete
hilite (cast member property)
put...after
put...before
put...into
string()
stringP()
text

Chunk expressions

Use these terms to identify chunks of text:

char...of
chars()
contains
EMPTY
item...of
itemDelimiter
last()
length()
line...of
number (characters)
number of items
number of lines
number (words)
offset() (string function)
paragraph
ref
selection (text cast member property)
selectedText
selEnd (fields only)
selStart (fields only)
string()
stringP()
value()
word...of

Editable text

Use the **editable** property to specify whether text is editable.

Shocked fonts

Use these terms to include Shocked fonts with downloaded text.

RecordFont
bitMapSizes

originalFont characterSet

Character formatting

Use these terms to format text:

backColor
bgColor
charSpacing
color()
dropShadow
font
fontSize
fontStyle
foreColor

Paragraph formatting

Use these terms to format paragraphs:

alignment
bottomSpacing
firstIndent
fixedLineSpace
leftIndent
margin
rightIndent
tabCount
tabs
topSpacing
wordWrap

Text cast member properties

Use these terms to work with the entire text content of a text cast member:

antiAlias
antiAliasThreshold
autoTab
HTML
kerning
kerningThreshold
picture (cast member property)
RTF

Lingo that applies to chunk expressions is also available to the text within a text cast member.

Mouse pointer position in text

Use these terms to detect where the mouse pointer is within text:

pointInHyperlink()
pointToChar()
pointToParagraph()
pointToWord()

pointToItem()

Text boxes for field cast members

Use these terms to set up the box for a field cast member:

border
boxType
lineCount
lineHeight() (function)
lineHeight (cast member property)
pageHeight

Scrolling text

Use these terms to work with scrolling text:

linePosToLocV()
locToCharPos()
locVToLinePos()
scrollByLine
scrollByPage
scrollTop

Constants

Use these terms to work with constants:

BACKSPACE
EMPTY
ENTER
RETURN (constant)
VOID

Time

These terms are useful for working with time.

Current date and time

Use these terms to determine the current date and time:

abbr, abbrev, abbreviated
date() (system clock)
long
short
systemDate

Measuring a length of time

Use these terms to measure time in a movie:

framesToHMS()
HMStoFrames()
milliseconds
startTimer
ticks
time()
timer

Timeouts

Use these terms to handle timeouts:

timeoutKeyDown
timeoutLapsed
timeoutLength
name (timeout property)
persistent
time()
timeoutHandler
timeoutMouse
timeoutPlay
timeoutScript
period
target
timeout()
timeoutList

Transitions

Use these terms to work with transitions:

changeArea

chunkSize

duration

puppetTransition

transitionType

Variables

These terms are useful for creating and changing variables:

Creating variables

Use these terms to create variables:

= (equals)
global
property

Testing and changing variables

Use these terms to check and change the values assigned to variables:

= (equals)
clearGlobals
globals
ilk()
put
set...to, set...=
showGlobals
showLocals

Vector Shapes

Use these Lingo terms to work with vector shapes:

addVertex
antiAlias
backgroundColor
broadcastProps
centerRegPoint
closed
curve
defaultRect
defaultRectMode
deleteVertex()
directToStage
endColor
fillColor
fillCycles
fillDirection
fillMode
fillOffset
fillScale
flashRect
flipH
flipV
gradientType
imageEnabled
moveVertex()
moveVertexHandle()
newCurve()
originH
originMode
originPoint
originV
regPointVertex
rotation
scaleMode
scale
showProps()
skew
static
strokeColor
strokeWidth
vertexList
viewPoint
viewScale
viewV

XML parsing

For information about using Lingo for XML parsing, see **Using the XML Parser** in the Director Support Center.

Xtras

Use these terms to work with Xtras:

movieXtraList

name (system property)

number of xtras

xtra

xtraList

xtras

Miscellaneous

Use **random()**and **randomSeed** to generate random numbers.

Operators

- # (symbol)
- . (dot operator)
- - (minus)
- -- (comment)
- & (concatenator)
- && (concatenator)
- () (parentheses)
- * (multiplication)
- + (addition)
- / (division)
- < (less than)
- <= (less than or equal to)
- <> (not equal)
- = (equals)
- > (greater than)
- >= (greater than or equal to)
- [ ] (bracket access)
- [ ] (list)
- " (string)
- \ (continuation)
- @ (pathname)

A

- abbr, abbrev, abbreviated
- abort
- abs()
- actionsEnabled
- activateApplication
- on activateWindow
- activeCastLib
- activeWindow
- actorList
- add
- addAt
- addProp
- addVertex
- after
- alert
- alertHook
- alignment
- allowCustomCaching
- allowGraphicMenu
- allowSaveLocal
- allowTransportControl
- allowVolumeControl
- allowZooming
- alphaThreshold
- ancestor
- and
- antiAlias
- antiAliasThreshold
- append
- applicationPath
- appMinimize
- atan()
- autoMask
- autoTab

B

C

D

E

F

G

H

halt
handler()
handlers()
height
hilite (command)
hilite (cast member property)
hitTest()
HMStoFrames()
hold
hotSpot
hotSpotEnterCallback
hotSpotExitCallback
HTML
hyperlink
on hyperlinkClicked
hyperlinkRange
hyperlinks
hyperlinkState

I

on idle
idleHandlerPeriod
idleLoadDone()
idleLoadMode
idleLoadPeriod
idleLoadTag
idleReadChunkSize
if
ilk()
image
image()
imageCompression
imageEnabled
imageQuality
importFileInto
in
INF
inflate
ink
inlineImeEnabled
insertFrame
inside()
installMenu
integer()
integerP()
interface()
intersect()
interval
into
invertMask
isBusy()
on isOKToAttach
isPastCuePoint()
isVRMovie
item...of
itemDelimiter

J

No Lingo terms begin with the letter J.

K

kerning
kerningThreshold
key()
keyboardFocusSprite
keyCode()
on keyDown
keyDownScript
keyPressed()
on keyUp
keyUpScript

L

label()
labelList
last()
lastChannel
lastClick()
lastEvent()
lastFrame
lastKey
lastRoll
left
leftIndent
length()
line...of
lineCount
lineDirection
lineHeight() (function)
lineHeight (cast member property)
linePosToLocV()
lineSize
linkAs()
linked
list()
listP()
loaded
loc
locH
locToCharPos()
locV
locVToLinePos()
locZ of sprite
log()
long
loop (keyword)
loop (cast member property)
loop (Flash property)
loopBounds
loopCount
loopEndTime
loopsRemaining
loopStartTime

M

map()
mapMemberToStage()
mapStageToMember()
margin
marker()
the markerList
mask
max()
maxInteger
mci

N

O

P

Q

R

S

T

U

V

W

X

Y

No Lingo terms begin with the letter Y.

Z

Help for this dialog box or inspector

An error occurred finding the help topic for this dialog box or inspector. To find information about it, note the name of the dialog box as it appears in its title bar and search for the name using the Help index or the Find feature.