Director 6.0 Help

Last revised May 8, 1997

The Director 6.0 Help system was created by many talented individuals.

Online help development by Jeff Swartz

Online help project management by Karen Olsen-Dunn

Help content written by Ben Melnick, Joe Schmitz, and Corinne Chandel

Show Me movie production by James Khazar, Zippy Lehnus, Kathleen Craig, and Dave Benman

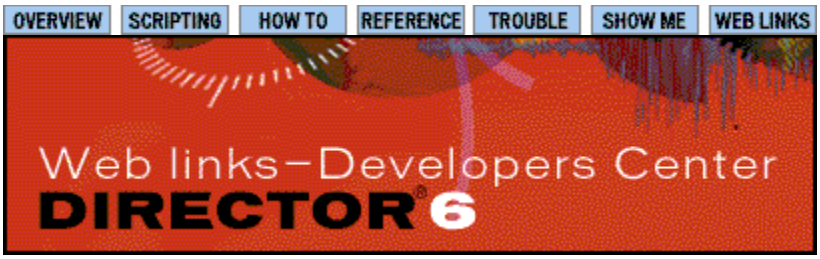
Show Me instructional design by Ben Melnick, James Khazar, Karen Olsen-Dunn, and Joe Schmitz

Art design by Ilene Sandler

Production art by Noah Zilberberg

Special thanks to: Director engineering, Tech support, and QA review teams; Lee Swearingen and Cathy Clarke at DXM Productions for developing the "Streaming Shockwave" movie; Karin Arrigoni for indexing the help system; Krzysztof Rogala for Show Me DLL development; Landra Tankha for testing help links; the makers of chocolate of any kind, which was devoured at a phenomenal rate during the project.

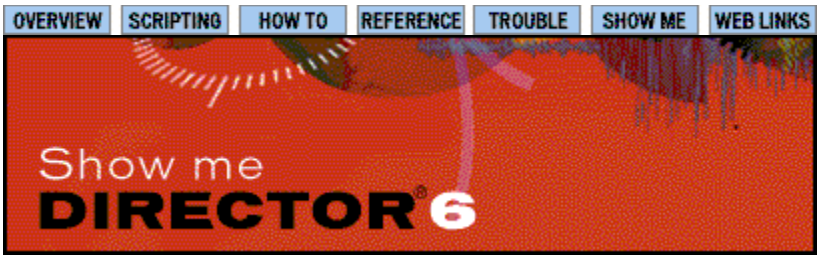
Macromedia, Inc.
600 Townsend St.
San Francisco, CA 94103



The Macromedia Director Developers Center web site is located at the following address: **www.macromedia.com/support/director/**. (Note: In Windows NT you cannot launch the URL by clicking here; you must type the URL directly into your web browser.)

The Director Developers Center includes these areas:

- **How do I...?** - The Experts Speak ... Working With Shockwave ... Tips & Tricks ... Show Me ... Submission Guide ... Made With Macromedia
- **Troubleshooting** - Frequently Asked Questions ... Recently Asked Questions ... Technotes
- **Updates & Downloads** - Product Updates ... Xtras ... Goodies
- **Interact** - Feedback
- **Doc Stuff** - Code Examples ... Glossary ... New & Improved ... Oops!
- **Resources** - Books & Macromedia Press ... User Groups ... Web Sites ... Events ... Programs & Services



Tutorial

- Learning Director

Director feature demos

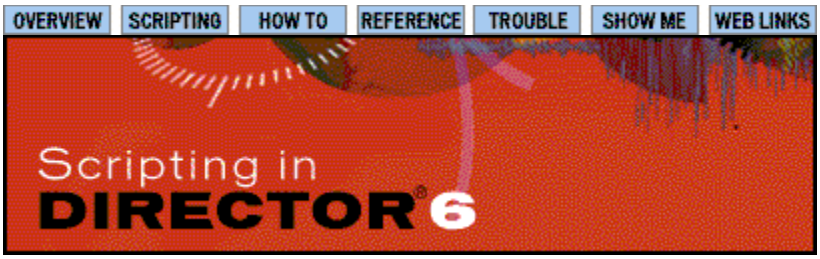
- Behaviors
- Cast Members and Sprites
- Color Palettes
- Film Loops
- Ink Effects
- Onion Skinning
- Streaming Shockwave
- Tweening

Examples about Lingo

Click to play the movie:	Click to open inside Director and see how it works:
Keyboard Lingo	--
Lists Lingo	Lists Lingo
Field Lingo	Field Lingo
Navigation Lingo	Navigation Lingo
--	Simple Child Object
--	Multiple Child Objects
Synchronized Media	Synchronized Media
Scriptable Authoring	Scriptable Authoring

More on the web

Check the Director Developers Center Show Me page to see more examples.



Director scripting includes the following categories:

Lingo

Browser scripts

Browser scripts

Shockwave movies and other objects within a browser-including Shockwave movies-interact by:

- **Sending messages to a movie:** More information
  - The scripting environment controls the Shockwave movie by using the `EvalScript` function to send Lingo commands to the movie. This is especially useful for allowing the scripting environment to control and synchronize movie playback.
  - Commands a browser can send the movie provide basic functionality for controlling the movie. For more information, see the **Commands a browser can send a Shockwave movie** help topic.
- **Sending messages to the scripting environment:** More information
  - The movie interacts with the browser and other Shockwave movies by using the `externalEvent` command to send instructions to the scripting environment.

Sending messages to a movie

The `EvalScript` function provides fairly complete access to and control of the movie without requiring a detailed definition of all available properties and functions.

This function calls an `on EvalScript` handler in the movie, which must be present for `EvalScript` to work.

The `on EvalScript` handler can process the parameter string however you choose to specify it.

Specify return values by placing a return string statement at the end of the `on EvalScript` handler or executing a Lingo statement that returns a value.

For security, avoid using the `do` command, which could allow arbitrary Lingo to be executed. Instead, use a handler such as the following, in which possible Lingo statements are already defined:

```
on EvalScript param
  case: param of
    "Horn": playHorn
    "Drum": bangDrum
    "Bongos": reallyBeat
  end case
end
```

The following commands can control a movie when issued from a browser:

To do this	Issue this command from the browser

Control the player.	`Play`
	`Stop`
	`Rewind`
	`GotoFrame`
	`GotoMovie`

Determine the Shockwave movie's current frame.	`GetCurrent Frame`
Send a string, such as a Lingo statement, to the Shockwave movie.	`EvalScript`
Determine whether the movie plays on loading or rewinding.	`AutoStart`

Commands a browser can send to a Shockwave movie

- **Play-**This command starts the movie if it is stopped. When the movie starts, its on prepareMovie handler runs first.

  This command has no effect on a movie that is already playing. For example, it doesn't restart a movie or run its `on prepareMovie` handler.

- **Stop**-This command completely stops the movie. However, the movie still redraws the Stage if another window passes over the movie.

  Except for `Rewind`, `GotoFrame`, and `GetCurrentFrame`, the movie doesn't generate or respond to events in this state until the movie receives the `Play` command from the scripting environment.

  This command runs the movie's `on stopMovie` handler when the movie stops. Lingo globals aren't preserved.

- **Rewind**-This command rewinds the movie.
  - If the `AutoStart` property is `true`, the movie then plays.
  - If the `AutoStart` property is `false`, the movie remains stopped.

- **GetCurrentFrame-**This command returns the number of the current frame. It works whether or not the

movie is stopped.

- **GotoFrame-**This command instructs the movie to go to a specified frame. It works whether or not the movie is stopped.
- **GotoMovie-**This command instructs the movie to go the specified URL, which may be relative. It doesn't work if the movie is stopped.

Properties a browser can set in a Shockwave movie

AutoStart-This property determines whether or not a movie starts automatically when it loads on the web page and when `Rewind` is called. Set the `AutoStart` property in the EMBED or OBJECT tag as an attribute; possible values are `true` or `false`.

- If `AutoStart` is `true`, movies play immediately upon loading. This is the default for tags that do not specify this attribute.
- If `AutoStart` is `false`, movies load and then stop.

The `AutoStart` property can also be set after the movie has been loaded on the page.

Sending messages to the scripting environment

Shockwave makes many methods and properties available to a variety of scripting languages.

- The `externalEvent` command communicates with the browser. For more information, see "Communicating with the browser," below.
- The `externalParamCount`, `externalParamName`, and `externalParamValue` functions access external parameters in an OBJECT or EMBED tag. For more information, see the **External parameter access from Lingo** help topic.

Communicating with the browser

Use the `externalEvent` command from within a movie to control external objects in the browser, including other Shockwave movies. This command has the form:

```
externalEvent "string"
```

Where *string* represents the browser instructions that the Shockwave movie sends.

This command doesn't return a value.

If you use `externalEvent` when authoring outside a browser, Director indicates in the Message window that `externalEvent` was called, but the command has no other effect.

How the scripting environment interprets the string included with the `externalEvent` command depends on the scripting environment.

Using `externalEvent` in LiveConnect

In LiveConnect, the string is evaluated as a function call. For JavaScript, simply define a function called whatever you choose in the HTML header. In the movie, the chosen function name and parameters should be passed as the string parameter to externalEvent. Because the browser must interpret parameters as strings, you must enclose them in single quotation marks.

In HTML, use a statement similar to the following:

```
function AnyFunctionName(parm1, parm2) {
// script here
}
```

In the movie, use a statement similar to the following:

```
externalEvent "AnyFunctionName('parm1', 'parm2')"
```

Using externalEvent in ActiveX

ActiveX handles externalEvent as an event. You can process this event and its string parameter the same as other events, such as an onClick event in a button object. Use a statement similar to the following:

```
externalEvent ("string")
```

In HTML, you can define a function to receive the event in the HTML header. Use a statement similar to this example in VBScript:

```
Sub
  MovieName_ExternalEvent (aString)
  REM process the string
End Sub
```

External parameter access from Lingo

When you use an EMBED or OBJECT tag to invoke a Shockwave movie from an HTML page, additional parameters can be included in these tags. These parameters and their values can be accessed from Lingo and are used to control a wide variety of aspects of the movie. Create a single movie and change its appearance or behavior by altering the parameter values on the EMBED or OBJECT tag.

Note: External parameters are accessed with new Lingo functions. Shockwave obtains the parameters from the HTML tag used to specify and run the movie. Shockwave provides the parameters to the movie through the functions described below.

The OBJECT tag supports only specific named parameters that were defined at the time the ActiveX control was created. Although Netscape Navigator supports user-defined parameters, to ensure that your movie runs properly on all browsers, use only those parameters that both Netscape Navigator and Internet Explorer recognize.

- Use the `externalParamCount` function to determine how many external parameters are passed from an HTML EMBED or OBJECT tag, u.
- Use the `externalParamName` function to determine the names of external parameters passed from an HTML EMBED or OBJECT tag.
- Use the `externalParamValue` function to obtain a specific value from the list of external parameters.

For more information about parameters that are accessible from Lingo, see Chapter 14, "Shockwave, the Internet, and Lingo," in Learning Lingo.

Lingo

Lingo is Director's scripting language. For general information on using **Lingo**, see the **Lingo Basics** topic.

Click a letter to view Lingo elements alphabetically:

A • B • C • D • E • F • G • H • I • J • K • L • M • N • O • P • Q • R • S • T • U • V • W • X • Y • Z

Lingo updates for Director 6

Click a category to see a list of Lingo elements that have changed since Director 5:

New Lingo elements in Director 6

Lingo that has changed in Director 6

Lingo that is outdated in Director 6

Lingo by feature

Click a feature below to see a list of Lingo elements that support that feature:

Behaviors
Cast members
Casts
Code structures & syntax
Computer & monitor
Digital video
Events
External files
Fields
Frames
Functions
Interface elements
Lists
Media Synchronization
Memory management
Movie control
Movie in a window
Navigation
Net Lingo
Operators
Parent scripts
Points and rects
Puppets
Score generation
Shockwave audio
Sound
Sprites
Strings
Text and Keys
Time
User interaction
Variables
Xtras

Lingo is Director's scripting language. Some of the advantages of using Lingo in a movie include:

- Internet operation and Shockwave support
- Navigation features that let users play and explore movies in the way they prefer
- Ability to play additional movies in windows
- Communication with users by receiving and sending information
- Ability to play animation and sound in ways that the Score alone can't
- Control of fields, sound, and digital video
- Creation of child objects
- Automation of authoring by duplicating manual tasks done by using the interface

Lingo Basics

Lingo is Director's scripting language. Some of the advantages of using Lingo in a movie include:

- Internet operation and Shockwave support
- Navigation features that let users play and explore movies in the way they prefer
- Communication with users by receiving and sending information
- Ability to play animation and sound in ways that the Score alone can't
- Control of fields, sound, and digital video
- Creation of child objects
- Automation of authoring by duplicating manual tasks done by using the interface

Lingo Menu

The Lingo menu appears when you click and hold the Lingo button in the Script window. This menu displays the complete set of Lingo elements that you can use to write scripts.

Choose an element from the Lingo menu to enter it into a script at the insertion point. This avoids typing the command and inserting typos.

Writing Scripts

For information about writing scripts, see:

Writing scripts
Types of scripts
Using variables
Event message hierarchy
Working with casts
Puppeting
Sprite properties
Handling text
Controlling sound
Managing memory
Using movie in a window
Child-parent scripts
Using XCMDs
Generating score
Creating dialog boxes

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

New Lingo elements in Director 6

See also: New Lingo elements in Director 5

The following elements are new in Director 6 or have had functionality added since Director 5:

activeCastLib
alertHook
applicationPath
behavesLikeToggle of member
behavesLikeToggle of sprite
bitRate of member
browserName
cacheDocVerify
cacheSize
call
callAncestor
netTextResult
numChannels of member
on alertHook
on beginSprite
on cuePassed
on endSprite
on EvalScript
on getBehaviorDescription
on getPropertyDescriptionList
on mouseEnter
on mouseLeave

Lingo that is outdated in Director 6

See also: Lingo that became outdated in Director 5

The following elements are obsolete and no longer supported:

castNum of sprite

colorQD

setCallBack

spriteBox

Lingo that has changed in Director 6

See also: Lingo that changed in Director 5

The following elements have been revised. The older elements are still supported, but they will become obsolete and should be avoided:

Director 5 Element	Director 6 Element
continue	go to the frame + 1
dontPassEvent	stopEvent
mouseCast	mouseMember
pause	go to the frame

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

New Lingo elements in Director 5

The following elements were new in Director 5 or had functionality added since Director 4. If you upgraded from Director 4 to Director 6, update Lingo in your movies to replace these elements:

activeWindow
autoTab of member
beginRecording
border of member
boxDropShadow
boxType of member
buttonType
cancelIdleLoad
case
castLibNum of sprite
center of member
changeArea of member
channelCoun of member
charPosToLoc
chunkSize of member
clearFrame
crop of member
deleteFrame
desktopRectList
digitalVideoTimeScale
digitalVideoType of member
dropShadow of member
duplicate(list)
duplicateFrame
duration of member
name of CastLib
new
number of CastLib
number of castLibs
number of members of castLib
on activateWindow
on closeWindow
on moveWindow
on resizeWindow
on rightMouseDown
on rightMouseUp
on zoomWindow
openWindow
otherwise
pageHeight of member
paletteMapping
pattern
paletteRef
the platform
preLoadMode of CastLib
preLoadMovie
rect of member
rightMouseDown, the
rightMouseUp, the
sampleRate

editable of member
emulateMultiButtonMouse
end case
endRecording
fileName of castLib
filled of member
finishIdleLoad
frameLabel
framePalette
frameScript
frameSound1
frameSound2
frameTempo
frameTransition
frontWindow
height of member
idleHandlerPeriod
idleLoadDone
idleLoadMode
idleLoadPeriod
idleLoadTag
idleReadChunkSize
insertFrame
keyPressed
lineCount of member
linePosToLocV
lineSize of member
loc of sprite
locToCharPos
locVToLinePos
loop of member
margin of member
media of member
member
memberNum of sprite
sampleSize of member
save castLib
score
scoreSelection
scriptsEnabled
scriptType
scrollByLine
scrollByPage
scrollTop of member
selection of castLib
setCallBack
shapeType
sound of member
wordWrap of member
timeScale of member
trackCount(member)
trackCount(sprite)
trackEnabled
trackNextKeyTime
trackNextSampleTime
trackPreviousKeyTime
trackPreviousSampleTime
trackStartTime(member)
trackStartTime(sprite)
trackStopTime(member)
trackStopTime(sprite)
trackText
trackType (member)
trackType (sprite)
transitionType of member
type of member
unloadMovie
updateFrame
updateLock
windowPresent

Lingo that became outdated in Director 5

The following elements became obsolete in Director 5 and are no longer supported. If you upgraded from Director 4 to Director 6, update the Lingo in your movies to replace these elements:

`birth` (use `new` instead)

`closeDA`

`factory`

`instance`

`openDA`

`when...then` constructs

Lingo that changed in Director 5

The following elements were revised to support multiple casts in Director 5. If you upgraded from Director 4 to Director 6, update Lingo in your movies to use the new elements:

Director 4 Element	Director 5 Element
backColor of cast	backColor of member
cast	member
castmembers	number of members
castNum of sprite	memberNum of sprite
castType of cast	type of member
center of cast	center of member
controller of cast	controller of member
crop of cast	crop of member
depth of cast	depth of member
duplicate cast	duplicate member
duration of cast	duration of member
erase cast	erase member
fileName of cast	fileName of member
foreColor of cast	foreColor of member
frameRate of cast	frameRate of member
height of cast	height of member
hilite of cast	hilite of member
loaded of cast	loaded of member
loop of cast	loop of member
modified of cast	modified of member
move cast	move member
name of cast	name of member
number of cast	number of member
number of castmembers	number of members
palette of cast	palette of member
picture of cast	picture of member
preLoad of cast	preLoad of member
preLoadCast	preLoadMember
purgePriority of cast	purgePriority of member
scriptText of cast	scriptText of member
size of cast	size of member
sound of cast	sound of member
text of cast	text of member
textAlign of field	alignment of member
textFont of field	font of member

textHeight of field	**lineHeight of member**
textSize of field	**fontSize of member**
textStyle of field	**fontStyle of member**
video of cast	**video of member**
width of cast	**width of member**

Child-parent scripts

A child object is a an occurrence of a parent script. Each child object of the same parent script shares the parent script's handlers but maintains individual values for properties:

A parent script contains three types of Lingo:

- An optional `on new` handler, which creates a new child object and sets its initial values when the handler is called. (The term `me` serves as a local variable that contains the child object itself and provides a placeholder for the child object in Lingo statements.)
- Optional additional handlers that control the child object's behavior and properties after the child object is created.
- An optional statement that declares which variables are property variables-variables for which each child object can maintain individual values regardless of the values for other child objects.

The `new` function creates a new child object when it uses the name of a parent script, as in the following syntax:

```
new(script "scriptName", argument1, argument2, argument3...)
```

The `new` function can be issued from anywhere in the movie. Customize the child object by changing the variable name and values of the arguments in the `new` statement.

An ancestor is an additional parent script whose handlers are available to a child object. A parent script makes another parent script its ancestor by assigning the script's name to the ancestor property. For example, the following statement makes the script Ancestor Ball Script an ancestor:

```
set ancestor to new(script"Ancestor Ball Script", ¬
listPosition)
```

For more information, see "Parent Scripts and Child Objects" in *Learning Lingo* . For a demonstration of parent scripts, see the "Single Child Object" and "Multiple Child Objects" sample movies.

{button See also,AL(`Child_parent_scripts')}

Event message hierarchy

Each message has a set series of scripts that it goes to after the message is sent. Different messages are sent to different types of scripts. The following table lists the order of objects that each type of message is sent to:

This message:	Is sent to this series of scripts:
mouseDown, mouseUp	Primary event handler, sprite, cast member, frame, and then movie scripts
mouseEnter, mouseLeave, mouseUpOutside, mouseWithin	Sprite scripts and then cast member scripts
keyDown, keyUp	Primary event handler, sprite, cast member, frame, and then movie scripts
enterFrame, exitFrame	Frame and then movie scripts
beginSprite, endSprite	Sprite scripts only
prepareFrame	Sprite, cast member, frame, and then movie scripts
idle	Primary event handler and then movie scripts
prepareMovie, startMovie and stopMovie handler	Movie scripts
activateWindow, closeWindow, openWindow, resizeWindow, zoomWindow	Movie scripts
Custom handler calling statements	Handlers in the script that sent the message and then to movie scripts

For more information about strategies for placing handlers, see Chapter 2, "Script Basics," in *Learning Lingo*.

{button See also,AL(`Lingo_event_hierarchy')}

Puppeting

Making a Score channel a puppet has the movie ignore the Score's settings for that channel and control the channel directly from Lingo.

- For sprite channels that are puppets, changes made by Lingo last beyond the life of the sprite that the change was first made to. Use `puppetSprite` or `puppet of sprite` to make a sprite channel a puppet.
- For sound channels, the channel remains under Lingo's control until you use a statement to return control to the Score. Use the `puppetSound` command to make a sound channel a puppet.
- For tempos and palette channels, the channel remains under Lingo's control until the playback head enters a frame that has a new palette or tempo setting. Use the `puppetTempo` command to make the tempo channel a puppet. Use the `puppetPalette` command to make the palette channel a puppet.
- For transition channels, the channel is under Lingo's control only for the specific instance in which the puppet transition is used. Use the `puppetTransition` command to make the transition channel a puppet.

For details about using one of these commands, see that command's entry in the online help. For more information about puppets, see Chapter 5, "Controlling Score Channels from Lingo," in *Learning Lingo*.

{button See also,AL(`Lingo_puppeting')}

Handling text

Director has two types of cast members dedicated to text:

- Text cast members can be edited and formatted in the Text window but not on the Stage or from Lingo. This type of text is a graphic after the movie is converted to a projector. Although text cast members are useful for displaying text that has been previously formatted as rich text, they can't receive user input or change formatting after the movie is distributed.
- Field cast members can be edited on the Stage and from Lingo while the movie plays. This lets you make movies in which the user can type characters and have Director manage them. Strings in a field cast member can be revised as the movie plays, and Lingo can format characters at any time.

A variety of field cast member properties determine the format of characters in the field cast member. For a list of properties that you can set, choose Fields from the categorized Lingo menu in the Script window.

Lingo duplicates the ability to make a field editable by setting the `editable of sprite` and the `editable of member` properties. To turn editable fields on or off independent of the interface, set the sprite or field cast member property to either TRUE or FALSE.

For more information about handling text and fields, see Chapter 7, "Working with Text and User Input," in *Learning Lingo*.

Controlling sound

Lingo can control many aspects of how sound plays in a movie. Using the **puppetSound** command, you can override the Score and play a sound cast member from Lingo. However, Lingo can also control how sound plays. The following table lists additional ways that Lingo can control sound and the elements that you use to achieve it. For details about an element, see its entry in the online help:

To do this:	Use these elements:
Play an external sound file	`sound playFile`
Check whether a sound channel is currently playing a sound and make the movie respond accordingly	`the soundBusy`
Turn sound off	`the soundEnabled`
Control sound volume from the movie	`the soundLevel`
Control how sound fades in and out	`sound fadeIn` and `sound fadeOut`

{button See also,AL(`Lingo_controlling_sound')}

Using movie in a window

Use Lingo to create a window by specifying the screen rectangle for the window and then specifying the movie assigned to the window. You can also make the window visible, change its type, set its title, or set the window's size and location.

Besides specifying which movie plays in the window and when the window opens and closes, you can control the behavior of the window itself and how movies in windows interact with the movie on the stage.

The following lists the Lingo elements you use to achieve different tasks for playing a movie in a window.

To:	Use:
Set up a rectangle for the movie	`set the rect of window "`*whichWindow*`" to rect(`*coordinates*`)`
Assign a movie to the window	`set the fileName of window "`*whichWindow*`" to "`*fileName*`"`
Specify whether the window title is visible	`set the titleVisible of window "`*whichWindow*`" to `*trueOrFalse*
Open the window that contains the movie	`open window "`*whichWindow*`"`
Specify the window type	`set the windowType of window "`*whichWindow*`"`
Move the window to the front	`moveToFront window "`*whichWindow*`"`
Move the window to the back	`moveToBack window "`*whichWindow*`"`
Make the window visible	`set the visible of window "`*whichWindow*`"`
Pass Lingo statements between windows	`tell "`*instructions*`"`
Close the window	`close window "`*whichWindow*`"`

For an example of how these elements are used together to create a movie in a window, see Chapter 11, "Movies in a window," in *Learning Lingo*.

{button See also,AL(`Lingo_mov_in_window')}

Using variables

Director remembers and updates values by using variables. 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 information the user enters or record whether a specific event has happened.

The value assigned to the variable can be a whole number, a decimal number (such as 1.56), a character string (such as "xyz" or a person's name), a symbol, or the result of a calculation.

Assigning values to variables

Assign a value-such as a number or a character string-to a variable with the `set...=` or `set...to` command. For example, the statement

```
set theName to "Mary"
```

assigns the string "Mary" to the variable `vName`.

Some possible sources for values assigned to variables are strings that the user types, the result of an arithmetic operation, and the result of clicking a particular sprite.

Creating variables

A variable is created the first time you assign a value to it, which is also called initializing a variable. You can then use the variable in other expressions or change its value based on whatever criteria you want. A variable can be a global variable or a local variable.

Global variables

Global variables can be shared among handlers and movies. The global variable exists and retains its value for as long as Director is running or until you issue the `clearGlobals` command.

You make a variable global by using the term `global` before the variable name in every handler that uses the global variable. Alternatively, make a variable global in every handler in a script by inserting the term `global` followed by the names of variables that are to be global at the top of the Script window.

Every handler that declares that a variable is global can use the variable's current value; if the handler changes the variable's value, the new value is available throughout the movie. Variables that you declare in the Message window are automatically global.

For example, to use someone's name several times in a movie, you could establish a global variable that contains a name entered by the user at the beginning of the movie.

The following statements makes `theName` a global variable and give it the value Mary:

```
global gName

set gName = "Mary"
```

Later in the movie, this handler could change the name assigned to this variable to "John." The variable can be changed from two different handlers because each handler treats the variable as global:

```
on nameChange

   global gName

   set gName to "John"

end
```

It is a good habit to start the names of all global variables with a small letter "g". This helps identify which variables are global when you examine Lingo code.

Display all current global variables and their current values by using the `showGlobals` command in the Message window.

Local variables

A local variable exists only as long as the handler in which it is defined is running. You can use a local variable in any handler, but it is available only while that handler is running.

Unless the handler uses the term `global` to declare that a variable is global, the variable is automatically a local variable.

Display all current local variables in the handler by using the `showLocals` command. This command can be used in the Message window or in handlers to help with debugging. The result appears in the Message window.

Treating variables as local is a good idea when you only want to use the variable temporarily in that one handler. This helps avoid unintentionally changing the value in another handler that uses the same variable name.

{button See also,AL(`Lingo_using_vars')}

Types of scripts

A script's type is determined by where it is attached in the movie. The type of script in which you place Lingo can affect the script's behavior.

There are four types of scripts:

- *Score scripts* are assigned to sprites and frames in the Score. A Score script assigned to a sprite is called a *sprite script*. A Score script assigned to a frame's script channel is called a *frame script*.

  Sprite scripts can respond to `beginSprite`, `endSprite`, `mouseDown`, `mouseUp`, and rollover events. If the sprite is a field, the script can also respond to `keyDown` and `keyUp` events.

  Frame scripts can respond to rollover, `prepareFrame`, `enterFrame`, `exitFrame`, `mouseDown`, `mouseUp`, `keyDown` events. If the frame contains field sprites, the script can also respond to `keyDown` and `keyUp` events.

  Director automatically assigns numbers to new Score scripts. These numbers appear in the Script pop-up menu and in the cells that the script is assigned to. When you revise a Score script, the changes show up everywhere the script is assigned in the Score.

  Attach score scripts by selecting sprite and frames to attach them to and then selecting the script from the Script pop-up, or by dragging them from the Cast window to sprites or frames.

- *Scripts of cast members* are attached directly to a cast member independent of the Score. Scripts assigned to cast members can respond to `prepareFrame`, `mouseDown`, and `mouseUp` events and to keyDown and keyUp events if the cast member is a field.

- *Movie scripts* aren't attached to a specific object but are available to the entire movie. Scripts assigned to the movie can respond to rollover, `mouseDown`, `mouseUp`, `keyDown`, `keyUp`, `enterFrame`, `exitFrame`, `idle`, `prepareFrame`, `startMovie`, and `stopMovie` events.

- *Parent scripts* are a special type of script that contains Lingo used to create child objects. For information about parent scripts, see **Child-parent scripts**.

The title bar at the top of the Script window tells the script's type.

Score, movie, and parent scripts exist as full-fledged cast members in the Cast window. You can change one of these script's type to one of the others by choosing from the Type pop-up menu in the Script Cast Member Properties dialog box.

{button See also,AL(`Lingo_script_types')}

Managing memory

Loading cast members that require a large amount of memory can cause undesired pauses in a movie.

Lingo helps you minimize these pauses by controlling when specific cast members are loaded, setting how many bytes Director attempts to load at one time, and prioritizing when Director unloads them.

For more information, see the entry for individual Lingo elements that can manage memory. For a list of elements related to memory management, choose Memory Management from the features menu in the Script window.

{button See also,AL(`Lingo_managing_memory')}

Working with casts

Lingo uses the `castLib` keyword, followed by a cast's name, to identify a cast. For example, in the statement `set the text of member "Title" of castLib "News" to "Calendar"` it uses `castLib` to identify the cast News.

When you want to change a movie's content by switching casts, you can change the cast assigned to a sprite by changing the sprite's `castLibNum` property. The sprite then uses the cast member that has the same cast member number in the new cast.

When you want to replace a movie's content by replacing its casts, you can make the casts external casts and change the cast file assigned to the cast.

The following Lingo elements are useful for obtaining information about casts:

- **castLib**
- **fileName of castLib**
- **findEmpty**
- **name of castLib**
- **number of castLib**
- **number of castLibs**
- **number of members of castLib**
- **preLoadMode of castLib**
- **save castLib**

{button See also,AL(`Lingo_work_with_casts')}

Sprite properties

Using Lingo's sprite properties, you can check a sprite's current conditions and change many of them. For a complete list of sprite properties, choose Sprites from the Lingo features menu in the Script window. For more information about a specific property, such as whether it can be set from Lingo, see the property's entry in the online help.

The following properties can be set from Lingo provided that the sprite channel has been put under Lingo's control by the `puppetSprite` command:

backColor
blend
castLibNum
constraint
editable
foreColor
height
ink
lineSize
loc
locH
locV
member
memberNum
moveableSprite
pattern
stretch
trails
width

{button See also,AL(`Lingo_sprite_properties')}

Generating Score

Lingo duplicates manual tasks that you perform in the Score-such as selecting frames, specifying what's in each channel, and animating sprites over a series of frames-by creating a new frame and then specifying each channel's content. It repeats this, frame by frame, until the entire sequence of frames is set up.

You can add new frames, edit frames, or delete frames.

To start recording Score, you must issue the `beginRecording` keyword. When you are done recording Score, you must issue the `endRecording` keyword.

Lingo can specify each channel's content during a Score recording session. The following lists what Lingo can set for each channel:

Channel	Lingo that can set the channel's content
Label	`the frameLabel`
Tempo	`the frameTempo`
Palette	`the framePalette`
Transition	`the frameTransition`
Sound channel 1	`the frameSound1`
Sound channel 2	`the frameSound2`
Script	`the frameScript`
Sprite channels	Sprite properties such as `memberNum of sprite`, `locH` and `locV`, and `moveable of sprite`. (Assign a sprite script by setting `the scriptNum of sprite`.)

When the frame's content is complete, use the `updateFrame` command to enter the new content. The `updateFrame` command makes a copy of the current frame, inserts it as the next frame, and then advances to the new frame. After Director has entered the new frame, you can specify that frame's content.

Several commands are available to add or delete frames. The following table lists these commands and their result:

Command	Result
`clearFrame`	Deletes everything in the current frame, but remains in the frame.
`deleteFrame`	Deletes the current frame. The next frame then becomes the current frame.
`duplicateFrame`	Duplicates the current frame and its content. The duplicate frame is inserted after the current frame and then becomes the current frame.
`insertFrame`	Inserts a copy of the current frame following the current frame. The new frame then becomes the current frame.
`updateFrame`	Enters the changes made to the current frame. The command then makes a copy of the current frame, inserts it as the next frame, and then advances to the new frame.

For more information about generating Score from Lingo, see Chapter 12, "Creating movies from Lingo," in *Learning Lingo*.

{button See also,AL(`Lingo_generating_score')}

Writing scripts

You write scripts in the Script window. The following are common ways to perform basic tasks for creating, assigning, and opening scripts:

To:	Do this:
Open a new Score script	Choose New Script from the Script pop-up menu; or double-click a frame in the script channel. When you open a new Score script, the script receives the number of the first available location in the Cast window.
Open a new movie script	Click the Add button in the Script window.
Attach a Score script to one or more cells in the Score	Drag the script from the Cast window to the score, OR select locations in the Score and then choose the Score script number from the Script pop-up menu.
Remove a Score script from the cell	Select the cell and then choose Clear Script from the Script pop-up.
Change a Script window's type	Open the script's Cast Member Properties and then choose a type from the Type pop-up menu.
Cycle through the scripts in the Script window	Use the Next Cast Member and Previous Cast Member arrows at the top of the Script window to advance or back up to the script.
Open the script assigned to a cast member	Click Script in the Cast Member Properties dialog box; or select the script in the Cast window and then click the script button at the top of the Cast window.
Open a Score, movie, or parent script	Double-click the script in the Cast window.
Duplicate a script	Select the script and choose Duplicate from the Edit menu

{button See also,AL(`Lingo_writing_scripts')}

Dialog boxes from the MUI Xtra

The MUI Xtra provides fully functional dialog boxes set up the way that you specify. The dialog boxes respond to user action and return information about the user's choice.

These dialog boxes don't require the memory or disk footprint of a MIAW that simulates a dialog box. They also draw the appropriate Windows controls, follow preferences for dialog box appearance set by the user in Windows 95, and draw with gray-scale controls on the Macintosh.

See **Creating dialog boxes** for an explanation of how to create dialog boxes. For examples of scripts that use this Lingo, see the movie Dialogs.dir in the Goodies folder.

Dialog boxes created from the MUI Xtra also provide:

- Horizontal scrolling within editable text fields
- Clipboard operations for editable text
- Activation of a default item when the user presses Enter or Return
- Activation of a cancel item when the user presses Esc-period or Command-period.

The MUI Xtra is used for some of Director's user interface. Each dialog box is an instance of the MUI Xtra.

The MUI Xtra supports these dialog box types:

- General purpose dialog boxes that you can set up with any arrangement of buttons, editable fields, labels, sliders, and pop-ups
- Alerts that offer several possible button and icon combinations
- Standard open file dialog boxes
- Standard save file dialog boxes
- A dialog box for entering a URL

What the MUI Xtra contains

The MUI Xtra contains the following that you can use to create dialog boxes:

- These Lingo elements:

  Alert
  FileOpen
  FileSave
  GetItemPropList
  GetUrl
  GetWidgetList
  GetWindowPropList
  Initialize
  ItemUpdate
  New
  Run
  Stop
  WindowOperation

- A predefined list of window properties that you can use as default values for a general purpose dialog box
- A predefined list of properties for components of a general purpose dialog box

Creating dialog boxes

To create dialog boxes from the MUI Xtra:

1. Plan and design the dialog box.

Determine what users need the dialog box to achieve, the type of interface components that best accomplish this, and where the data in the dialog box comes from. Sketch a preliminary dialog box to get a better idea of where options need to appear and how well they fit into the dialog box.

2. Decide which type of dialog box to use.

The MUI Xtra provides predefined dialog boxes for entering alerts, opening and saving files, and for entering URLs. These are relatively easy to create. If none of these types are appropriate, you can define your own general purpose dialog box.

3. Use the statement `new (xtra "MUI")` **to create an instance of the MUI Xtra.**

4. Set up the dialog box.

There are two general categories of dialog boxes. The procedure for creating a dialog box is different for each category:

- Dialog boxes for alerts, opening and saving files, and opening URLs are created by calling the function that creates the specific dialog box. When the user clicks a button, the dialog box responds accordingly and sends the appropriate message back to Director. To set up one of these dialog boxes, use the following functions:

To customize this type of dialog box	Set parameters for this function
Alert	**Alert**
Open file dialog box	**FileOpen**
Save file dialog box	**FileSave**
Dialog box for entering a URL	**GetUrl**

- General purpose dialog boxes are created by defining each attribute and component in the dialog box, and then explicitly initializing and running the dialog box. See **Using a general purpose dialog box** for more information.

Using a general purpose dialog box

When you use a general purpose dialog box, first set up the dialog box's overall attributes and components. After the dialog box's attributes are set up, use the `Initialize`, `Run`, and `Stop` commands to control the dialog box.

To define a general purpose dialog box:

1. Create two lists of definitions:

- One list is a property list that contains definitions of overall dialog box attributes such as the name that appears in the title bar, the box's size and location, and the handler that runs when an event occurs in the dialog box. See **Specifying overall dialog box properties**
- The other list is a linear list of definitions for each component in the dialog box. Each item in this linear list is a property list that defines one component of the dialog box. See **Specifying dialog box content**

2. Use the **Initialize** command to specify that these lists are the ones Director should use for definitions of the general purpose dialog box.

Note: A Lingo statement can't contain more than 256 characters. Since lists are typically long, it might be necessary to assign some values to variables and then use the variables in the list.

To use a general purpose dialog box:

After the dialog box is defined and initialized, Lingo can open, respond to, and close the dialog box.

- Open a modal dialog box by using the **Run** command. Open a non-modal dialog box by using the **WindowOperation** command with the `#show` option.
- Use the **ItemUpdate** command to update the dialog box in response to user actions.
- Respond to user actions by sending events in the dialog box to the handler set up to handle callbacks. See **Sending dialog box events to Director**
- Close a modal dialog box by using the **Stop** command. Close a non-modal dialog box by using the **WindowOperation** command with the `#hide` option.

Alert

Syntax: `Alert(`*MUIObject*`, `*alertPropertiesList*`)`

This command displays an alert dialog box created from an instance of the MUI Xtra. This feature is in addition to the simple alerts generated by the `alert` command.

The MUI Xtra provides modal alerts. The alert can be moveable or non-moveable.

To create the alert, create a MUI Xtra object, and then issue the alert command with a list containing definitions of alert properties as the second parameter.

The following are properties that you must specify and their possible values:

Property	Possible values	Specifies

`#buttons`	`#Ok`
	`#OkCancel`
	`#AbortRetryIgnore`
	`#YesNoCancel`
	`#YesNo`
	`#RetryCancel`
		The set of buttons that appear in the alert. The buttons appear in the order that they are named in each symbol.

`#default`	The ordinal number of the button that becomes the default. For example, if the alert's buttons are OK and Cancel, 2 specifies the Cancel button. Specify 0 for no default.	Which button is the default.

`#icon`	`#stop`
	`#note`
	`#caution`
	`#question`
	`#error`
		The type of icon that appears in the alert. Specify 0 for no icon.

`#message`	A string	The message that appears in the alert
`#movable`	`TRUE` `FALSE`	Whether the alert is moveable
`#title`	A string	The alert's title

You must explicitly specify each of the alert's properties. The MUI Xtra doesn't provide a list of default alert properties.

Lingo returns a value for the button that the user clicks.

Alerts can be almost as big as the screen; you can display a lengthy description if appropriate.

Example:

The following statements create and display an alert dialog box.

- The first statement creates an instance of the MUI Xtra, which is the object used as the dialog box.
- The second statement sets up a list of the alert's properties.
- The final statements use the `Alert` command to display the alert and report which buttons the user clicks.

```
set alertObj = new(xtra "MUI")


set alertInitList = Â
[ #buttons : #YesNo, Â
#title : "Alert Test", Â
#message : "This shows Yes and No buttons",Â
```

#movable : TRUE]

```
if objectP ( alertObj ) then
  set result = Alert( alertObj, alertInitList )
  case result of
    1 : -- the user clicked yes
    2 : -- the user clicked no
    otherwise : -- something is seriously wrong
  end case
end if
```

FileOpen

Syntax: `FileOpen(`*MUIObject*`,` *string* `)`

This function displays a standard file open dialog box provided by an instance of the MUI Xtra.

The second parameter specifies a string that appears in the editable field when the dialog box opens. The user can specify which file to open by entering the file name in the editable field. When the user clicks a button, the text is returned.

- If the user clicks Cancel, the returned text is the same as the value that was passed in.
- If the user clicks OK, the returned text is a platform-specific path.

Example:

These statements create and display a standard file open dialog box.

- The first statement creates an instance of the MUI Xtra, which is the object used as the dialog box.
- The second statement assigns a string to the variable `fileString`, which is used later as the second parameter of the `FileOpen` command.
- The third statement uses the `FileOpen` command to generate the open file dialog box.
- The final statements check whether the original string sent with the `FileOpen` command is the same as the string that was returned when the user clicked a button. If the values are different, the user selected a file to open.

```
set aMuiObj = new (xtra "MUI")

set fileString = "Open this file"

set result = fileOpen(aMuiObj, fileString)

-- Check to see if the dialog was canceled
if (result <> fileString) then
   -- The dialog wasn't canceled, do something with
-- the new path data.
else
   put "ERROR - fileOpen requires a valid aMuiObj"
end if
```

FileSave

Syntax: `FileSave( MUIObject, string, message )`

This function displays a standard file saving dialog box that saves the current file. The dialog box is created from an instance of the MUI Xtra.

- The *string* parameter specifies the string that appears in the dialog box's file name field. The user can use this field to enter a new file name for the file. When the user clicks a button, Lingo returns a value for *string* that contains the field's content. If the user clicks Cancel, the returned string is the same as the original string.
- The *message* parameter is the string that appears above the dialog box's editable field.

Example:

These statements create and display a file save dialog box.

- The first statement creates an instance of the MUI Xtra, which is the object used as the dialog box.
- The second statement assigns a string to the variable `fileString`, which is used later as the second parameter of the `FileSave` command.
- The third statement uses the `FileSave` command to generate the save file dialog box.
- The final statements check whether the result after the user clicks a button is the same as the string sent when the dialog box opened. If it differs, the user clicked something other than Cancel.

```
set aMuiObj = new (Xtra "MUI")


set fileString = "save this file"

set result = fileSave( aMuiObj, fileString, "with this prompt" )

if (result <> fileString) then
   -- The dialog wasn't canceled, do something with the new      -- path data.
end if
```

GetItemPropList

Syntax: `GetItemPropList(`*MUIObject*`)`

This function returns a list of the MUI Xtra's predefined properties for components in a general purpose dialog box. It is useful for defining new components in a general purpose dialog box. Use `GetItemPropList` to obtain a comprehensive list of properties and values and then edit individual properties as necessary.

The list of properties and values are the following:

Property	Default value
`#value`	`0`
`#type`	`#checkBox`
`#attributes`	`[],`
`#title`	`"title"`
`#tip`	`"tip"` (This is not supported at the release of Director 6. It is reserved for possible use in later versions of the MUI Xtra.)
`#locH`	`20`
`#locV`	`24`
`#width`	`200`
`#height`	`210`
`#enabled`	`1`

Example:

These statements define the beginning of a dialog box window.

- The first statement creates an instance of the MUI Xtra, which is the object used as the dialog box.
- The second statement assigns a list of default dialog component settings to the variable `tempItemProps`.
- The third statement makes the component the dialog box's beginning by changing its type to `#windowBegin`.

```
set aMuiObj = new (Xtra "MUI")

set tempItemProps = GetItemPropList(aMuiObj)

set the type of tempItemProps = #windowBegin
```

GetUrl

Syntax: `GetUrl(`*MUIObject*`, `*message*`, `*MovableOrNot*`)`

This function displays a dialog box for entering a URL and returns the URL that the user enters.

- *message* specifies the message that appears in the field for entering a URL. When the dialog box is first opened, this string is sent as a predefined value. When the user clicks a button, Lingo returns the string that the user entered. If the user clicks Cancel, the returned string is the same as the original value.
- On the Macintosh, *MovableOrNot* specifies whether the dialog box is movable. TRUE makes the dialog box movable. FALSE makes the dialog box not movable. The GetURL dialog box is always movable in Windows.

Example:

These statements display a dialog box for entering a URL.

- The first statement creates an instance of the MUI Xtra, which is the object used as the dialog box.
- The second statement uses the `GetUrl` function to display a moveable dialog box for entering URLs and assigns the dialog box to the variable `result`. The message "Enter a URL here" appears in the dialog box's field for entering a URL.
- The final statements check whether the result after the user clicks a button is the same as the string sent when the dialog box opened. If it differs, the user entered a URL and clicked OK.

```
set MUIObj = new (xtra "Mui")

set result = GetUrl(MUIObj, "Enter a URL", TRUE )

if objectP ( MUIObj) then

  set result = GetUrl( MUIObj, "Enter a URL", TRUE )

  if ( result <> "Enter a URL" ) then

    goToNetPage result

  end if

end if
```

GetWidgetList

Syntax: `GetWidgetList(`*MUIObject*`)`

This function returns a linear list of symbols for types of general purpose dialog box components supported for an instance of the MUI Xtra.

Example:

This statement displays a list of widgets supported by MUIObject, which is an instance of the MUI Xtra:

```
put GetWidgetList(MUIObject)

-- [#dividerV, #dividerH, #bitmap, #checkBox, #radioButton, #PopupList,
#editText, #WindowBegin, #WindowEnd, #GroupHBegin, #GroupHEnd,
#GroupVBegin, #GroupVEnd, #label, #IntegerSliderH, #FloatSliderH,
#defaultPushButton, #cancelPushButton, #pushButton, #toggleButton]
```

GetWindowPropList

Syntax: `GetWindowPropList(`*MUIObject*`)`

This function returns a list of the MUI Xtra's predefined settings for a general purpose dialog box's window.

When defining a new general purpose dialog box, use `GetWindowPropList` function to obtain a comprehensive list of dialog box properties and values and then edit individual properties as necessary. Besides being more convenient, this technique ensures compatibility with future versions of the MUI Xtra that may have additional properties.

These are the window properties and predefined values that `GetWindowPropList` returns:

Property	Predefined value
`#type`	`#normal`
`#name`	`"window"`
`#callback`	`"nothing"`
`#mode`	`#data`
`#xPosition`	`100`
`#yPosition`	`120`
`#width`	`200`
`#height`	`210`
`#modal`	`1`
`#toolTips`	`0`
`#closeBox`	`1`
`#canZoom`	`0`

Example:

These statements define a new general purpose dialog box. The first statement assigns a list of predefined properties to the variable `thePropList`. Subsequent statements customize the dialog box by modifying these settings:

```
set thePropList = GetWindowPropList(muiObject)

set the name     of thePropList = "Picture Window"
set the callback of thePropList = "theWindowCallback"
set the mode     of thePropList = #data
set the modal    of thePropList = TRUE
set the closeBox of thePropList = FALSE
```

Initialize

Syntax: `Initialize (`*MUIObject*`, `*initialPropertyList*`)`

This command sets up a general purpose dialog box from an instance of the MUI Xtra.

initialPropertyList is a property list that specifies where Director obtains definitions for the dialog box's attributes.

- The property list associated with the `#windowPropList` property (see **Specifying overall dialog box properties**) is the list Director uses for definitions of the overall dialog box's attributes.
- The linear list associated with the `#windowItemList` property (see **Specifying dialog box content**) is the list Director uses for definitions of individual components. Each item in the list is a property list that defines one component.

Example:

This statement initializes a general purpose dialog box created from MUIObject, which is an instance of the MUI Xtra. The list `aWindowPropList` contains definitions for the overall dialog box. The list `aWindowItemList` contains definitions for the dialog box's individual components:

```
Initialize(MUIObj, Â
[#windowPropList:aWindowPropList, Â
 #windowItemList:aWindowItemList])
```

ItemUpdate

Syntax: `ItemUpdate(`*MUIObject*`, `*itemNumber*`, `*itemInputPropList*`)`

This command updates a component in a general purpose dialog box. It is useful for updating a dialog box in response to user actions while the dialog box is displayed.

- *itemNumber* represents the number of the item being updated.
- *itemInputPropList* represents the list of new properties for the item.

The `ItemUpdate` command can be used for many things; possible uses include enabling or disabling buttons, changing the range of a pop-up, updating a sliders position, and updating editable text items if the user enters an invalid value.

You may want to update individual items in a dialog box depending on user input, user interaction, or to display underlying data. Although you would typically update an item's #value, you can also update everything else about an item, except for its type. Set the `height`, `width`, `locH`, and `locV` properties to -1 to keep their current values.

Example 1:

These statements update the dialog box component that has the number `itemNum`.

- The first statement obtains the component's definitions from the overall list of item definitions.
- The second and third statements modify the component's `type` and `attribute` properties.
- The last statement uses the `ItemUpdate` command to update the component's settings.

```
set baseItemList = getAt ( theItemList, itemNum )

set the type of baseItemList  = #IntegerSliderH

set the attributes  of baseItemList = [#valueRange  :[#min:1, #max:8,
#increment:1, #jump:1, #acceleration:1]

ItemUpdate(MUIObj, itemNum, baseItemList)
```

Example 2:

This handler updates

```
on smileyUpdate
  -- declare globals
  global smileyIndex, gMuiSmileDialObj, itemNumSmile, Â     itemNumSlide,
smileItemList

  -- validate dialog object
  if ( objectP ( gMuiSmileDialObj ) ) then
    -- get a list to put in new/updated values
    set baseItemList = duplicate ( getAt ( smileItemList, Â
    itemNumSmile ) )

    -- metrics can be set to -1, this "keeps them the same"
    -- instead of updating.
    -- could also be set to a new value if you
    -- wanted to resize the item or relocate it.
    set the width of baseItemList  = -1 -- keep previous
    set the height of baseItemList = -1 -- keep previous
    set the locH of baseItemList   = -1 -- keep previous
    set the locV of baseItemList   = -1 -- keep previous

    -- in this particular case, the value is
```

```
    -- the only thing that's changing

    set the value of baseItemList  = string(smileyIndex)
    -- member name

    -- tell the dialog to update the item number
    -- with the new item list
    ItemUpdate(gMuiSmileDialObj, itemNumSmile, baseItemList)
  end if
end
```

new

Syntax: `new(xtra "MUI")`

or

```
set theObject = new(xtra "MUI")
```

This function creates a new instance of the MUI Xtra that you can use as a dialog box. This use is in addition to creating child objects, new cast members, and instances of other Xtras with the `new` function.

You must create a new instance of the MUI Xtra before you can use it to create a dialog box or obtain any of the Xtra's predefined values.

Example:

This statement creates a new instance of the MUI Xtra and assigns it to the variable `MUIObject`:

```
set MUIObject = new(xtra "MUI")
```

run

Syntax: `run(`*MUIObject*`)`

This command displays a general purpose modal dialog box created from an instance of the MUI Xtra.

Before Director can open the dialog box, use the **Initialize** command to define the dialog box. See **Specifying dialog box content** and **Specifying Overall dialog box properties** for more information about specifying the content of a general purpose dialog box.

Note: To open a non-modal dialog box, use the **WindowOperation** command with the `#show` option. This command allows other Lingo to run in the movie while the non-modal dialog box is open.

Example:

This handler checks whether the object MUIObject exists and displays a general purpose dialog box from MUIObject if it is:

```
on runDialog
  global MUIObject
  if objectP(MUIObject) then
     run(MUIObject)
  end if
end
```

stop

Syntax: `stop(`*MUIObject*`, `*stopItem*`)`

This function stops the general purpose dialog created from an instance of the MUI Xtra. After the function is called, Lingo returns the results as the number for the *stopItem* parameter.

The *stopItem* parameter is returned from the `run(`*MUIOject*`)` call. Use this to pass back a parameter indicating how the dialog box was stopped. For example, this could return 1 if the user clicked OK and return 0 if the user clicked Cancel.

Note: To close a non-modal dialog box, use the **WindowOperation** command with the `#hide` option.

Example:

This handler stops the general purpose dialog box created from MUIObject. The second parameter of the stop command is zero, which fulfills the requirement for a value but has no other purpose:

```
on stopDialog
   global MUIObject
   if ( objectP (MUIObject)) then
      stop(MUIObject, 0)
   end if
end stopDialog
```

WindowOperation

Syntax: `WindowOperation(`*MUIObject*`, `*operation*` )`

This command controls the window for a general purpose dialog box.

Replace the *operation* parameter with a value that determines what the window does. Possible values and their result are:

Possible values	Result
`#show`	Displays a non-modal dialog box only. (To open a modal dialog box, use the **Run** command.)
`#hide`	Hides a non-modal dialog box. (To close a modal dialog box, use the **Stop** command.)
`#center`	Centers the window on the monitor screen.
`#zoom`	Sends a message that the user clicked the zoom box on the window. The callback handler must resize the dialog box, if you want the window to resize after the user clicks the zoom box.
`#tipsOn`	Turns tool tips on. (This is not supported in the initial release of Director 6. `#tipsOn` is reserved for future versions of the MUI Xtra.)
`#tipsOff`	Turns tool tips off. (This is not supported in the initial release of Director 6. `#tipsOff` is reserved for future versions of the MUI Xtra.)

Example 1:

This handler checks whether MUIObject exists and displays the dialog box if the does:

```
on showDialog
   global MUIObject
   if objectP( MUIObject ) then
      WindowOperation( MUIObject, #show )
   end if
end showDialog
```

Example 2:

This statement hides the dialog box created from MUIObject:

```
WindowOperation(MUIObject, #hide)
```

Specifying overall dialog box properties

To specify the dialog box's overall properties, create a property list that sets values for each of the window's properties.

The `GetItemPropList` function (see **GetItemPropList**) returns a predefined list of attributes. It's usually easiest to obtain a predefined list from the `GetItemPropList` function and then modify values as needed.

Before opening the dialog box, use the **Initialize** command to specify which list Director uses as the source of definitions for the overall dialog box.

The following are the overall dialog box properties and their possible values:

Property	Possible Values
`#type`	`#alert`, `#normal`, `#palette`
`#name`	String that contains the window name. Use "" for no name.
`#callback`	Handler that processes the result of the callback. See **Sending dialog box events to Director** for more information.
`#mode`	`#data`, `#dialogUnit`, or `#pixel`. These set the way that Director lays out the dialog box.
`#Xposition`	Number of pixels that upper left corner of the dialog box appears from the left of the screen. Specify -1 to have the dialog box appear in the center.
`#Yposition`	Number of pixels that the top of the dialog box appears from the top of the screen. Specify -1 to have the dialog box appear in the center.
`#width`	Width of the window in pixels. Specify 0 to have the dialog box set its width automatically.
`#height`	Height of the window in pixels. Specify 0 to have the dialog box set its height automatically.
`#modal`	TRUE or FALSE. Sets whether the dialog box is modal.
`#toolTips`	TRUE or FALSE. Sets whether to use tooltips initially. (This is not supported in Version 1.0 of the MUI Xtra. It is reserved for future use.)
`#closeBox`	TRUE or FALSE. Specifies whether the dialog box has a close box.
#canZoom	TRUE or FALSE. Specifies whether the dialog box can zoom.

Note: A Lingo statement can't contain more than 256 characters. Because lists are typically long, it might be necessary to assign some values to variables and then use those variables in the list.

Example:

These statements set up a list of overall dialog box properties.

- The first statement creates an instance of the MUI Xtra and assigns it to the variable `theBox`.
- The second statement assigns a list of predefined values to the variable `aWindowPropList`.
- The next three statements modify the name, callback, and width properties that were obtained from the `GetWindowPropList` function.
- The last statement displays the modified list in the Message window. The result appears at the end of the example.

```
set theBox = new(xtra "mui")

set aWindowPropList = GetWindowPropList(theBox)

set the name of     aWindowPropList = "General Settings"

set the callback of aWindowPropList =  "otherCallback"

set the width of    aWindowPropList = 200

put aWindowPropList

 -- [#type: #normal, #name: "General Settings", #callback:
"otherCallback", #mode: #data, #xPosition: 100, #yPosition: 120, #width:
200, #height: 210, #modal: 1, #toolTips: 0, #closeBox: 1, #canZoom: 0]
```

Specifying dialog box content

To specify the content of a general purpose dialog box, create a linear list of definitions for each component of the dialog box. Each definition is a property list that defines one component. Components appear in the order that they are listed.

Some components define the structure of the dialog box, such as the beginning and end of the window and the start and end of horizontal and vertical sets of components. When constructing the dialog box, you must use this overall framework of components:

Window beginning

Additional components as desired. This can include nested sets of additional horizontal and vertical groups, horizontal and vertical dividers, labels, and interface features such as buttons, check boxes, editable fields, and other interface elements.

Window end

To define individual components, create a property list that specifies each component. For convenience, use the `GetItemPropList` function obtain a predefined list of values and then modify properties as needed.

The following are properties for general dialog box components and their possible values:

Property	Possible value
`#value`	Determines the component's value type. Possible types are integer, float, or string. See the following section "Possible #value settings for general purpose dialog box components" for more information about the `#value` property.
`#type`	One of the supported component types (see **Possible component types**)
`#attributes`	A list that specifies the component's attributes. Attributes that you can specify depend on the component's type. See **Possible attribute settings**
`#title`	String used as the title for the component. Specify "" for no title.
`#tip`	String used as the message in a tool tip. . Specify "" for no tool tip. (Tool tips aren't supported in the initial release of the MUI Xtra. This is reserved for possible use in future releases.)
`#locH`	The distance of the component's left edge from the left of the dialog box. There is no need to specify this property if `#data` is specified for the dialog box's `#mode` property.
`#locV`	The distance of the component's top from the top of the dialog box. There is no need to specify this property if `#data` is specified for the dialog box's `#mode` property.
`#width`	The component's width in pixels. Specify 0 to have Director automatically size the component's width. There is no need to specify this property if `#data` is specified for the dialog box's `#mode` property.
`#height`	The component's height in pixels. Specify 0 to have Director automatically size the component's height. There is no need to specify this property if `#data` is

	specified for the dialog box's `#mode` property.
`#enabled`	TRUE or FALSE. Specify whether the item is enabled.

Possible #value settings for general purpose dialog box components

The `#value` settings for general purpose dialog box components are crucial for the correct display and editing of data in the dialog box. The following are possible settings for each type of component:

Component	Possible setting for #value	Examples
`#dividerV`	(Ignored)	(Not applicable)
`#dividerH`	(Ignored)	(Not applicable)
`#bitmap`	Cast member number, name, or reference	`12, "First bitmap castmember", member 12 of castLib "Internal"`
`#checkBox`	Boolean	`TRUE, FALSE`
`#radioButton`	Boolean	`TRUE, FALSE`
`#PopupList`	String, integer, or float	`1, "Fred", 2.3`
`#editText`	String	"Edit This Text"
`#WindowBegin`	(Ignored)	(Not applicable)
`#WindowEnd`	(Ignored)	(Not applicable)
`#GroupHBegin`	(Ignored)	(Not applicable)
`#GroupHEnd`	(Ignored)	(Not applicable)
`#GroupVBegin`	(Ignored)	(Not applicable)
`#GroupVEnd`	(Ignored)	(Not applicable)
`#label`	String	`"label: ", "Long Label Text Here"`
`#IntegerSliderH`	Integer	`1, 100, 0`
`#FloatSliderH`	Float	`1.2, 0.0, 12.345`
`#defaultPushButton`	(Ignored)	(Not applicable)
`#cancelPushButton`	(Ignored)	(Not applicable)
`#pushButton`	(Ignored)	(Not applicable)
`#toggleButton`	Boolean	`TRUE, FALSE`

Before opening the dialog box, use the **Initialize** command to specify which list Director uses as the source of definitions for the overall dialog box.

Note: A Lingo statement can't contain more than 256 characters. Because lists are typically long, it might be necessary to assign some values to variables and then use those variables in the list.

Example:

These statements specify the components for a general purpose dialog box that contains an editable field and a button. The dialog box is created from theBox, which is an instance of the MUI Xtra.

- The first statement creates a new list named aWindowItemList.
- Each subsequent set of statements obtains a predefined list of properties for an individual component, modifies it as necessary, and then adds the modified list to the overall list of dialog box components.
- The final statement displays the new overall list in the Message window. The result appears at the end of the example. For easier reading, the display has been broken out into separate sections for each item in the list.

```
set aWindowItemList = []

-- Set up the beginning of the dialog box
set tempItemPropList = GetItemPropList(theBox)
set the type of tempItemPropList = #windowBegin
append (aWindowItemList, duplicate(tempItemPropList))

-- Set up the beginning of an overall group
set tempItemPropList = GetItemPropList(theBox)
set the type of tempItemPropList = #groupHBegin
append (aWindowItemList, duplicate(tempItemPropList))

-- Set up an editable field
-- This will use default values for text attributes, because no
-- list of values is assigned to the attributes property.
set tempItemPropList = GetItemPropList(theBox)
set the type of tempItemPropList = #editText
set the value of tempItemPropList = "Enter text here"
append (aWindowItemList, duplicate(tempItemPropList))

-- Set up an OK button
set tempItemPropList = GetItemPropList(theBox)
set the type of tempItemPropList = #pushButton
set the title of tempItemPropList = "OK"
append (aWindowItemList, duplicate(tempItemPropList))


-- Set up end of overall group
set tempItemPropList = GetItemPropList(theBox)
set the type of tempItemPropList = #groupHBegin
append (aWindowItemList, duplicate(tempItemPropList))
```

```
	-- Last, set up end of window

	set tempItemPropList = GetItemPropList(theBox)

	set the type of tempItemPropList = #windowEnd

	append (aWindowItemList, duplicate(tempItemPropList))


	put aWindowItemList

	-- [

	[#value: 0, #type: #WindowBegin, #attributes: [], #title: "title", #tip: "tip", #locH: 20, #locV: 24, #width: 200, #height: 210, #enabled: 1],

	[#value: 0, #type: #GroupHBegin, #attributes: [], #title: "title", #tip: "tip", #locH: 20, #locV: 24, #width: 200, #height: 210, #enabled: 1],

	[#value: "Enter text here", #type: #editText, #attributes: [], #title: "title", #tip: "tip", #locH: 20, #locV: 24, #width: 200, #height: 210, #enabled: 1],

	[#value: 0, #type: #pushButton, #attributes: [], #title: "OK", #tip: "tip", #locH: 20, #locV: 24, #width: 200, #height: 210, #enabled: 1],

	[#value: 0, #type: #GroupHBegin, #attributes: [], #title: "title", #tip: "tip", #locH: 20, #locV: 24, #width: 200, #height: 210, #enabled: 1],

[#value: 0, #type: #WindowEnd, #attributes: [], #title: "title", #tip: "tip", #locH: 20, #locV: 24, #width: 200, #height: 210, #enabled: 1]

]
```

Possible attribute settings

To specify the attributes of a general purpose dialog box component, assign a list of attribute specifications to the component's `#attributes` property.

What you can specify depends on the type of component. The following are possible properties and values you can specify for a component's `attribute` property. The available attributes vary depending on the type of component.

Attribute	Possible values	Can be set for
`#textSize`	One of the following: `#large`, `#tiny`, `#normal`(default)	Strings
`#textStyle`	A list that includes any of the following attributes: `#bold`, `#italic`, `#underline`, `#plain` (default), `#inverse` (v2)	Strings
`#textAlign`	One of the following: `#left`, `#right`, `#center` (defaults to system language standard)	Strings
`#popupStyle`	One of the following: `#tiny`, `#cramped`, `#normal` (default)	Pop-ups
`#valueList`	A list of values that appear in a pop-up. All values are coerced to strings. For example, Director treats `['one', #two, 3, 4.0]` as four pop-up items, each of which is treated as a string.	Pop-ups
`#valueRange`	A list of a slider's minimum, maximum, increment, jump, and acceleration values. Available properties to set are `#min`, `#max`, `#increment`, `#jump`, and `#acceleration`. This list is the default setting: `[#min:0.0, #max:1000.0, #increment:1.0, #jump:10.0, #acceleration:0.5]`	Sliders
`#sliderStyle`	A linear list of one or both of `#ticks` or `#value`.	Sliders
`#layoutStyle`	A list of values that includes one or more of the following: `#minimize`, `#lockPosition`, `#lockSize`, `#centerH`, `#right`, `#left`, `#centerV`, `#top`, `#bottom`.	All items that aren't grouped by the `#groupHBegin`, `#groupHEnd`, `#groupVBegin`, or `#groupVEnd` properties.
`#bitmapStyle`	A property list that specifies that one of the bitmap icons in the MUI Xtra is to be used as an icon for a general purpose dialog box. The property in the property list is `#bitmapIcon`. Possible values are `#stop`, `#note`, `#caution`, `#question`, `#error`. For example, the statement `set the attributes of Â tempPropItemList= Â [#bitmapIcon:#caution]` sets the caution symbol as the value for `#bitmapStyle`.	Bitmaps

Example 1:

The following statement sets up attributes for an editable field:

```
set the attributes of tempTextItemList = Â [#textSize:#Normal,
#textStyle:[#Normal]]
```

Example 2:

The following statement sets up attributes for a slider:

```
set the attributes of tempItemList = Â
[#valueRange: [#min:0.0, #max:100.0, Â
 #jump:5.0, #acceleration:0.5], #sliderStyle:[#ticks]]
```

Possible component types

The following are available component types, whether they use a title, and the attributes that you can specify for them. See **Possible attribute settings** for details about setting specific attributes.

Property	Title	Available attributes
`#bitmap`	No	`#layoutStyle`, `#bitmapStyle`
`#cancelPushButton`	Yes	`#textSize`, `#layoutStyle`
`#checkBox`	Yes	`#textSize`, `#layoutStyle`
`#defaultPushButton`	Yes	`#textSize`, `#layoutStyle`
`#dividerH`	No	`#layoutStyle`
`#dividerV`	No	`#layoutStyle`
`#editText,`	No	`#textSize`, `#justification`, `#textStyle`, `#layoutStyle`
`#floatSliderH`	No	`#sliderStyle`, `#valueRange`, `#layoutStyle`
`#groupHBegin`	No	None
`#groupHEnd`	No	None
`#groupVBegin`	No	None
`#groupVEnd`	No	None
`#IntegerSliderH`	No	`#sliderStyle`, `#valueRange`, `#layoutStyle`
`#label`	No	`#textSize`, `#justification`, `#textStyle`, `#layoutStyle`
`#none`	No	`#layoutStyle`
`#popupList`	No	`#popupStyle`, `#valueList`, `#layoutStyle`
`#pushButton`	Yes	`#textSize`, `#layoutStyle`
`#radioButton`	Yes	`#textSize`, `#layoutStyle`
`#toggleButton`	Yes	`#textSize`, `#layoutStyle`
`#windowBegin`	No	None
`#windowEnd`	No	None

The **GetWidgetList** function also returns a list of supported general purpose dialog box component types.

Sending dialog box events to Director

Sending dialog events back to Director lets you determine events such as clicking buttons, changing values, or when the dialog box moves or closes.

When a dialog box event occurs, the callback message includes three arguments:

- The callback event symbol, which identifies the type of event.
- Event-specific information such as the number in the `#windowItemList` of the item involved in the event
- The new item property list for the affected item

To specify how a general purpose dialog box responds, write Lingo that instructs the movie what to do in response to an event and then assign that Lingo to the event's symbol. Specify which handler responds to a dialog box event by assigning the handler's name to the dialog box's `#callback` property in the list assigned to the `#windowPropList`.

The following are possible events that a general purpose dialog box can send back to Director to indicate what happened to the dialog box:

Event	What occurred
`#itemChanged`	The item's value has changed.
`#itemClicked`	The user clicked an item in the dialog box.
`#windowOpening`	The dialog box window opened.
`#windowClosed`	The dialog box window closed.
`#windowZoomed`	The dialog box window was zoomed.
`#windowResized`	The dialog box window was resized.
`#itemEnteringFocus`	An item in the dialog box got focus.
`#itemLosingFocus`	An item in the dialog box lost focus.

These are some tricks for setting up a callback:

- Store item numbers in global variables when building an item list. To check whether an item is important, compare `eventData` to the stored index number.
- Because all buttons have their text in their titles, you can compare the title against text to determine if a particular button was clicked. For portability and easier localization, store text button titles in global string variables when setting up a dialog box and to do comparisons.
- Always make sure there is always a way to close a dialog box. Otherwise, it can be impossible to dismiss it.
- Because you can update items, if text changes you can enable an OK button that is unavailable until the text changes.
- Avoid hanging inside a modal dialog by making sure there is an event that can stop the dialog box.

Example:

The following handler contains a `case` statement that responds to an event in a general purpose dialog box. Each possible event is a condition that Director tests for.

The three arguments are the following:

- `event` is the type of event.
- `eventData` is specific information about the event.
- `itemPropList` is the list of property's for the item.

The handler's name is already assigned to the dialog box's `#callback` property in the list assigned to the `#windowPropList`.

```
on theWindowCallback event, eventData, itemPropList

  global gMuiSmileDialObj, smileyIndex
  if symbolP(event) then -- basic error check
    case event of
      #itemChanged :
        if the type of itemPropList = #IntegerSliderH then
          set smileyIndex = the value of itemPropList
          smileyUpdate  FALSE
        end if

      #itemClicked :
        case ( the type of itemPropList ) of
          #bitmap : beep
          #defaultPushButton :
            if ( the title of itemPropList = "Forward" ) then
              smileyIndexIncrement
              smileyUpdate TRUE
            end if
          #pushButton :
            if ( the title of itemPropList = "Back" ) then
              smileyIndexDecrement
              smileyUpdate TRUE
            end if

          #cancelPushButton :
            if ( objectP ( gMuiSmileDialObj ) ) then
              stop(gMuiSmileDialObj, 1)
            end if
        end case

        #windowOpening:
        #windowClosed:
        #windowZoomed :
        #windowResized :
        otherwise :
      end case
    end if
end theWindowCallback
```

Lingo elements-behaviors

- bitRate of member
- call
- callAncestor
- on getBehaviorDescription
- on getPropertyDescriptionList
- on runPropertyDialog
- scriptInstanceList of sprite
- spriteNum

{button See Also,AL(`Lingo_menu_behaviors')}

Lingo elements-media sychnronization

- cuePointNames of member
- cuePointTimes of member
- isPastCuePont
- on cuePassed

{button See Also,AL(`Lingo_menu_media_synch')}

Lingo elements-net Lingo

Click a Lingo element for more information:

- browserName
- cacheSize
- cacheDocVerify
- clearCache
- downloadNetThing
- externalParamCount
- externalParamName
- externalParamValue
- frameReady
- getLatestNetID
- getNetText
- getPref
- gotoNetMovie
- gotoNetPage
- mediaReady
- netAbort
- netDone
- netError
- netLastModDate
- netMIME
- netPresent
- netStatus
- netTextResult
- on streamStatus
- preloadNetThing
- proxyServer
- setPref
- tellStreamStatus

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-navigation

Click a Lingo element for more information:

- delay
- go
- go loop
- labelList
- go next
- marker
- go previous
- play
- play done

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-movie control

Click a Lingo element for more information:

- abort
- centerStage
- changeArea of member
- do
- exitLock
- halt
- fixStageSize
- labelList
- lastFrame
- movie
- movieName
- name of xtra
- platform
- play
- play done
- printFrom
- saveMovie
- score
- scriptsEnabled
- searchPaths
- stage
- stageBottom
- stageColor
- stageLeft

Lingo elements-user interaction

Click a Lingo element for more information:

Lingo elements-computer & monitor

Click a Lingo element for more information:

desktopRectList
drawRect of window
floatPrecision
machineType
mci
restart
romanLingo
shutDown
version

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-memory management

Click a Lingo element for more information:

cancelIdleLoad
finishIdleLoad
freeBlock
freeBytes
idleHandlerPeriod
idleLoadDone
idleLoadMode
idleLoadPeriod
idleLoadTag
idleReadChunkSize
loaded of member
memorySize
movieFileFreeSize
movieFileSize
on idle
preLoad
preLoad of member
preLoadBuffer member
preLoadEventAbort
preLoadNetThing
preLoadEventAbort
preLoadMember
preLoadMode of CastLib
preLoadMovie
preLoadRAM
purgePriority of member
ramNeeded
size of member
unload
unloadMember
unloadMovie

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-casts

Click a Lingo element for more information:

activeCastLib
castLib
fileName of castLib
findEmpty
name of CastLib
number of CastLib
number of castLibs
number of members of castLib
preLoadMode of CastLib
save castLib

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-cast members

Click a Lingo element for more information:

center of member
crop
depth of member
duplicate member
erase member
fileName of member
new
number of member
palette of member
pasteClipBoardInto
pattern
picture of member

filled of member
height of member
hilite of member
importFileInto
media of member
member
modified of member
name of member
pictureP
rect of member
regPoint
shapeType
transitionType
type of member
URL of member
width of member

Lingo elements-sprites

Click a Lingo element for more information:

backColor of sprite
castLibNum of sprite
constrainH
constraint of sprite
constrainV
cursor of sprite
foreColor of sprite
height of sprite
ink of sprite
left of sprite
loc of sprite
locH of sprite
locV of sprite
memberNum of sprite
moveableSprite of sprite
on beginSprite
on endSprite
paletteRef
puppetSprite
scriptNum of sprite
sprite
sprite...intersects
sprite...within
stretch of sprite
type of sprite
trails of sprite
updateStage
visible of sprite
width of sprite
zoomBox

Lingo elements-frames

Click a Lingo element for more information:

frame
frameLabel
framePalette
frameScript
frameSound1
frameSound2
frameTempo
frameTransition
label
marker
on enterFrame
on exitFrame
puppetPalette
puppetTempo
puppetTransition

Lingo elements-Score generation

Click a Lingo element for more information:

beginRecording
clearFrame
chunkSize of member
deleteFrame
duplicateFrame
endRecording
insertFrame
scoreColor
scoreSelection
scriptType
updateFrame
updateLock

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-external files

Click a Lingo element for more information:

@ pathname operator
applicationPath
closeDA
closeResFile
closeXlib
copyToClipBoard
downloadNetThing
fileName of castLib
fileName of member
fileName of window
getNthFileNameInFolder
importFileInto
movieFileFreeSize
moviePath
open
openDA
openResFile
openXlib
pathName
preloadNetThing
searchCurrentFolder
searchPath
setCallBack
showResFile
showXlib
sound playFile
URL of member
xFactoryList

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-movie in a window

Click a Lingo element for more information:

closeWindow
drawRect of window
fileName of window
forget window
frontWindow
modal of window
moveToBack
moveToFront
name of window
on activateWindow
on closeWindow
on moveWindow
on openWindow
on resizeWindow
on zoomWindow
open window
rect of window
sourceRect
tell
title of window
titleVisible of window
visible of window
window
windowList
windowPresent
windowType

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-parent scripts

Click a Lingo element for more information:

actorList

ancestor

new

property

stepFrame

Lingo elements-lists

Click a Lingo element for more information:

- [] (list operators)
- add
- addAt
- addProp
- append
- count
- deleteAll
- deleteAt
- deleteOne
- deleteProp
- duplicate
- findPos
- findPosNear
- getaProp
- getAt
- getLast
- getOne
- getPos
- getProp
- getPropAt
- ilk
- list
- listP
- param
- paramCount
- min
- setaProp
- setAt
- setProp
- set
- sort

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-code structures & syntax

Click a Lingo element for more information:

- # (symbol operator)
- ¬ (continuation symbol)
- -- (comment delimiter)
- case
- end
- end case
- end if
- end repeat
- exit
- exit repeat
- global
- if ... then ...
- if ... then ... else
- next
- next repeat
- or
- otherwise
- repeat while
- repeat with
- repeat with...down to
- repeat with...in list
- the
- VOID

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-strings

Click a Lingo element for more information:

- &
- &&
- char...of
- chars
- itemDelimiter
- last
- length
- line...of

contains
delete
EMPTY
item...of
number of items in
number of lines in
string
stringP
word...of

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-functions

Click a Lingo element for more information:

abs
and
atan
charToNum
cos
exp
FALSE
float
floatP
integer
integerP
mod
or
sqrt
TRUE

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-fields

Click a Lingo element for more information:

after
alignment of member
autoTab of member
backColor of member
before
border of member
boxDropShadow
boxType of member
char...of
charPosToLoc
chars
charToNum
contains
delete
dropShadow of member
editable of member
editable of sprite
font of member
fontSize of member
fontStyle of member
foreColor of member
height of member
lineCount of member
lineHeight
lineHeight of member
linePosToLocV
lineSize of member
lineSize of sprite
locToCharPos
locVToLinePos
margin
number of chars in
number of items in
number of words in
offset
pageHeight of member
put...after
put...before
put...into
scrollByPage
scrollLine
scrollTop of member
selEnd
selStart

hilite
in
into
item...of
string
text of member
wordWrap of member
word...of

Lingo elements-digital video

Click a Lingo element for more information:

controller of member
digitalVideoTimeScale
digitalVideoType
directToStage of member
duration of member
frameRate of member
loop of member
movieTime of sprite
movieRate of sprite
pauseAtStart
quickTimePresent
setTrackEnabled
timeScale of member
trackCount(member)
trackCount(sprite)
trackEnabled
trackNextKeyTime
trackNextSampleTime
trackPreviousKeyTime
trackPreviousSampleTime
trackStartTime(member)
trackStartTime(sprite)
trackStopTime(member)
trackStopTime(sprite)
trackText
trackType (member)
trackType (sprite)
video of member
videoForWindowsPresent

Lingo elements-sound

Click a Lingo element for more information:

channelCount
puppetSound
sampleRate
sampleSize of member
sound close
sound fadeIn
sound fadeOut
sound of member
sound playFile
sound stop
soundBusy
soundEnabled
volume of sprite

Lingo elements-Shockwave audio

Click a Lingo element for more information:

bitsPerSample of member
bitRate of member
copyrightInfo of member
duration of member
getError
getErrorString
play member
preLoadBuffer member
preLoadTime of member
sampleRate of member
soundChannel of member
state of member

Lingo elements-interface elements

Click a Lingo element for more information:

- alert
- buttonStyle
- buttonType
- checkBoxAccess
- checkBoxType
- checkMark of menuItem
- enabled of menuItem
- installMenu
- menu
- menuItem
- name of menu
- name of menuItem
- number of menuItems
- number of menus
- script of menuItem

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-operators

Click a Lingo element for more information:

- # (pound sign)
- - (minus sign)
- -- (comment delimiter)
- & (concatenator)
- && (concatenator)
- * (multiplication)
- + (addition)
- / (division)
- < (less than)
- < = (less than or equal to)
- <> (not equal)
- = (equal sign)
- > (greater than)
- > = (greater than or equal to)
- [] (list operators)
- " (string constant)
- ¬ (continuation symbol)
- ( ) [parentheses]
- @ (pathname operator)

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-puppets

Click a Lingo element for more information:

- puppet of sprite
- puppetPalette
- puppetSound
- puppetSprite
- puppetTempo
- puppetTransition
- updateStage

Lingo elements-time

Click a Lingo element for more information:

- abbr, abbrev, abbreviated
- date
- delay
- framesToHMS
- HMStoFrames
- long
- short
- startTimer
- ticks
- time
- timer
- timeoutKeyDown
- timeoutLapsed
- timeoutMouse
- timeoutPlay
- timeoutScript

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-variables

Click a Lingo element for more information:

- clearGlobals
- global
- property
- put
- set...to and set...=
- showGlobals
- showLocals

Lingo elements-events

Click a Lingo element for more information:

- exitFrame
- keyDown
- on activeWindow
- on closeWindow
- on deactivateWindow
- on enterFrame
- on exitFrame
- on idle
- on keyDown
- on keyUp
- on mouseDown
- on mouseUp
- on moveWindow
- on openWindow
- on prepareFrame
- on prepareMovie
- on resizeWindow
- on rightMouseDown
- on rightMouseUp
- on startMovie
- on stepFrame
- on stopMovie
- on timeOut
- on zoomWindow

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-text and keys

Click a Lingo element for more information:

- BACKSPACE
- charToNum
- EMPTY
- ENTER
- hilite
- key
- keyCode
- keyPressed
- lastKey
- mouseChar
- numToChar
- RETURN
- string
- symbolP
- value

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-xtras

Click a Lingo element for more information:

- name of xtra
- number of xtras
- xtra
- xtras

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-points and rects

Click a Lingo element for more information:

- bottom of sprite
- inflate rect
- inside
- intersect
- map
- offset
- offset rect
- point
- rect
- rect of sprite
- right of sprite
- source rect
- top of sprite
- union rect

Lingo elements-A

Click a Lingo element for more information:

- abbr
- abbrev
- abbreviated
- abort
- abs
- activateWindow
- activeCastLib
- activeWindow
- actorList
- add
- addAt
- addProp
- after
- alert
- alerthook
- alignment of member
- ancestor
- and
- append
- applicationPath
- atan
- autoTab of member

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-B

Click a Lingo element for more information:

- backColor of member
- backColor of sprite
- BACKSPACE
- beep
- beepOn
- before
- beginRecording
- beginSprite
- behavesLikeToggle of member
- behavesLikeToggle of sprite
- birth
- bitsPerSample of member
- bitRate of member
- blend of sprite
- border of member
- bottom of sprite
- boxDropShadow of member
- boxType of member
- browserName
- buttonStyle
- buttonType

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-C

Click a Lingo element for more information:

- cacheDocVerify
- cacheSize
- call
- callAncestor
- cancelIdleLoad
- case
- castLib
- castLibNum of member
- castLibNum of sprite
- close window
- closeDA
- closeResFile
- closeWindow
- closeXlib
- colorDepth
- colorQD
- commandDown
- constrainH

castNum of sprite
center of member
centerStage
changeArea of member
channelCount of member
char...of
charPosToLoc
chars
charToNum
checkBoxAccess
checkBoxType
checkMark of menuItem
chunkSize of member
clearCache
clearFrame
clearGlobals
clickLoc
clickOn
constraint of sprite
constrainV
contains
continue
controlDown
controller of member
copyrightInfo of member
copyToClipBoard
cos
count
cpuHogTicks
crop of member
cuePassed
cuePointNames of member
cuePointTimes of member
currentSpriteNum
currentTime
cursor
cursor of sprite

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-D

Click a Lingo element for more information:

date
deactivateWindow
delay
delete
deleteAll
deleteAt
deleteFrame
deleteOne
deleteProp
depth of member
desktopRectList
digitalVideo
digitalVideoTimeScale
digitalVideoType of member
directToStage of member
do
dontPassEvent
doubleClick
down
downloadNetThing
drawRect of window
dropShadow of member
duplicate(list)
duplicate member
duplicateFrame
duration of member

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-E

Click a Lingo element for more information:

editable of member
ENTER

editable of sprite
else
EMPTY
emulateMultiButtonMouse
enabled of member
enabled of sprite
enabled of menuItem
end
end case
end repeat
endRecording
endSprite
enterFrame
erase member
EvalScript
exit
exit repeat
exitFrame
exitLock
exp
externalEvent
externalParamCount
externalParamName
externalParamValue

Lingo elements-F

Click a Lingo element for more information:

factory
fadeIn
fadeOut
FALSE
field
fileName of castLib
fileName of member
fileName of window
filled of member
findEmpty
findPos
findPosNear
finishIdleLoad
fixStageSize
float
floatP
floatPrecision
font of member
fontSize of member
fontStyle of member
foreColor of member
foreColor of sprite
forget window
frame
frameLabel
framePalette
frameRate of member
frameReady
frameScript
frameSound1
frameSound2
framesToHMS
frameTempo
frameTransition
freeBlock
freeBytes
frontWindow
fullColorPermit

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-G

Click a Lingo element for more information:

getaProp
getAt
getBehaviorDescription
getError
getPref
getProp
getPropAt
getPropertyDescriptionList

getErrorString
getLast
getLatestNetID
getNetText
getNthFileNameInFolder
getOne
getPos
global
go
go loop
go next
go previous
gotoNetMovie
gotoNetPage

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-H

Click a Lingo element for more information:

halt
height of member
height of sprite
hilite
hilite of member
HMStoFrames

Lingo elements-I

Click a Lingo element for more information:

Lingo elements-J

There are no Lingo elements that begin with the letter J.

Lingo elements-K

Click a Lingo element for more information:

key

keyCode

keyDown

keyDownScript

keyPressed

keyUp

keyUpScript

Lingo elements-L

Click a Lingo element for more information:

label
labelList
labelString of member
last
lastClick
lastEvent
lastFrame
lastKey
lastRoll
left of sprite
length
line...of
lineCount of member
lineHeight
lineHeight of member
linePosToLocV
lines
lineSize of member
lineSize of sprite
list
list operator ([])
listP
loaded of member
loc of sprite
locH of sprite
locToCharPos
locV of sprite
locVToLinePos
log
long
loop
loop of member

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-M

Click a Lingo element for more information:

machineType
map
map point
map rect
margin of member
marker
mAtFrame
max
maxInteger
mci
mDescribe
mDispose
me
media of member
mediaReady of member
member
member backColor
member memberType
member depth
mGet
min
mInstanceRespondsTo
mMessageList
mName
mNew
mod
modal of window
modified of member
mostRecentCuePoint
mouseChar
mouseDown
mouseDownScript
mouseEnter
mouseH
mouseItem
mouseLeave
mouseLine
mouseMember

member fileName
member foreColor
member height
member hilite
member loaded
member name
member number
member of sprite
member palette
member picture
member purgePriority
member rect
member regPoint
member scriptText
member text
member width
memberNum of sprite
memorySize
menu
menuItem
menuItems
menus
method
mouseUp
mouseUpOutSide
mouseUpScript
mouseV
mouseWithin
mouseWord
move member
moveableSprite of sprite
moveToBack
moveToFront
moveWindow
movie
movieFileFreeSize
movieFileSize
movieName
moviePath
movieRate of sprite
movieTime of sprite
mPerform
mPut
mRespondsTo
multiSound

Lingo elements-N

Click a Lingo element for more information:

name of CastLib
name of member
name of menu
name of menuItem
name of window
name of xtra
netAbort
netDone
netError
netLastModDate
netMIME
netPresent
netStatus
netTextResult
new
not
nothing
number of CastLib
number of castLibs
number of chars in
number of items in
number of lines in
number of member
number of members
number of members of castLib
number of menuItems
number of menus
number of words in
number of xtras
numChannels of member

next
next repeat
numToChar

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-O

Click a Lingo element for more information:

objectP
of
offset
offset rect
on
on activateWindow
on alerthook
on beginSprite
on closeWindow
on cuePassed
on deactivateWindow
on endSprite
on enterFrame
on EvalScript
on exitFrame
on getBehaviorDescription
on getPropertyDescriptionList
on idle
on keyDown
on keyUp
on mouseDown
on mouseEnter
on mouseLeave
on mouseUp
on mouseUpOutSide
on mouseWithin
on moveWindow
on openWindow
on prepareFame
on prepareMovie
on resizeWindow
on rightMouseDown
on rightMouseUp
on runPropertyDialog
on startMovie
on stepFrame
on stepMovie
on stopMovie
on streamStatus
on timeOut
on zoomWindow
open
open window
openDA
openResFile
openWindow
openXlib
optionDown
or
otherwise

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-P

Click a Lingo element for more information:

pageHeight of member
palette of member
paletteMapping
paletteRef
param
paramCount
pass
pasteClipBoardInto
preLoad
preLoad of member
preLoadBuffer member
preLoadEventAbort
preLoadMember
preLoadMode of CastLib
preLoadMovie
preloadNetThing

Lingo elements-Q

Click a Lingo element for more information:

Lingo elements-R

Click a Lingo element for more information:

ramNeeded
random
randomSeed
rect
rect of member
rect of sprite
rect of window
rect point
regPoint of member
relative
repeat while
repeat with
repeat with...down to
repeat with...in list
resizeWindow
restart
result
RETURN
return
right of sprite
rightMouseDown
rightMouseDown
rightMouseUp
rightMouseUp
rollOver
romanLingo
runMode
runPropertyDialog

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Lingo elements-S

Click a Lingo element for more information:

sampleRate of member
sampleSize of member
save castLib
saveMovie
score
scoreColor of sprite
scoreSelection
script of menuItem
scriptInstanceList of sprite
scriptNum of sprite
scriptsEnabled of member
scriptText of member
scriptType of member
scrollByLine
scrollByPage
scrollTop of member
searchCurrentFolder
searchPath
searchPaths
selection
selection of castLib
sound close
sound fadeIn
sound fadeOut
sound of member
sound playFile
sound stop
soundBusy
soundChannel of member
soundEnabled
soundLevel
sourceRect
SPACE
sprite
sprite...intersects
sprite...within
spriteBox
spriteNum
sqrt
stage
stageBottom
stageColor

selEnd
selStart
sendAllSprites
sendSprite
set...to and set...=
setaProp
setAt
setButtonImageFrom- CastMember
setCallBack
setPref
setProp
setTrackEnabled
shapeType
shiftDown
short
showGlobals
showLocals
showResFile
showXlib
shutDown
sin
size of member
sort
stageLeft
stageRight
stageTop
startMovie
starts
startTime of sprite
startTimer
state of member
stepFrame
stepMovie
stillDown
stop
stop member
stopEvent
stopMovie
stopTime of sprite
streamName of member
streamStatus
stretch of sprite
string
stringP
switchColorDepth
symbol
symbolP

Lingo elements-T

Click a Lingo element for more information:

TAB
tan
tell
tellStreamStatus
text of member
the
then
ticks
time
timeoutKeyDown
timeoutLapsed
timeoutLength
timeoutMouse
timeoutPlay
trackCount(member)
trackCount(sprite)
trackEnabled
tracking
trackNextKeyTime
trackNextSampleTime
trackPreviousKeyTime
trackPreviousSampleTime
trackStartTime(member)
trackStartTime(sprite)
trackStopTime(member)
trackStopTime(sprite)
trackText

timeoutScript
timer
timeScale of member
title of window
titleVisible of window
to
top of sprite
trace
traceLoad
traceLogFile
trackType (member)
trackType (sprite)
trails of sprite
transitionType of member
TRUE
tweened of sprite
type of member
type of sprite

Lingo elements-U

Click a Lingo element for more information:

union rect
unLoad
unLoadMember
unloadMovie
updateFrame
updateLock
updateMovieEnabled
updateStage
URL of member

Lingo elements-V

Click a Lingo element for more information:

value
version
video of member
videoForWindowsPresent
visible of sprite
visible of window
VOID
voidP
volume of member
volume of sound
volume of sprite

Lingo elements-W

Click a Lingo element for more information:

when...then
while
width of member
width of sprite
window
windowList
windowPresent
windowType of window
with
within
word...of
wordWrap of member

Lingo elements-X

Click a Lingo element for more information:

xFactoryList
xtra

xtras

Lingo elements-Y

There are no Lingo elements that begin with the letter Y.

Lingo elements-Z

Click a Lingo element for more information:

zoomBox

zoomWindow

(symbol operator)

Syntax: `#`*symbolName*

This symbol operator defines a symbol. In addition to integers, floating-point numbers, strings, and objects, Lingo also has a symbol data type. The value *symbolName* begins with an alphabetical character and may be followed by any number of alphabetical or numerical characters.

The valid operations on symbols are:

- Assignment to a variable
- Comparison
- Being passed as a parameter to a handler or method
- Being returned as a value from a handler or method

A symbol is a self-contained unit that can be used to represent a condition or flag. It does not consist of individual characters in the same sense as a string. However, you can convert a symbol to a string for display purposes by using the `string` function.

Symbols take up much less space than strings and can be manipulated. Essentially, symbols have the speed and memory advantages of integers but give you the descriptive power of strings.

The following are some important points about symbol syntax:

- Symbols are case-insensitive.
- Symbols can't start with a number.
- Symbols use the 128 ASCII characters, and letters with diacritical or accent marks are treated as their base letter.

All symbols, global variables, and names of parameters passed to global variables are stored in a common lookup table. Because 16-bit projectors for Windows 3.1 use the segmented memory model, no single list can exceed 64K in size. As a result, you can typically have no more than approximately 5000 symbols in a Windows 3.1 projector.

For more information about using symbols, strings, and other values, see Chapter 2, "Script Basics," in *Learning Lingo*.

Example:

This statement sets the variable named state to the symbol #`Playing`:

```
set state = #Playing
```

{button See also,AL(`Lingo_pound_sign')}

- (minus sign)

Syntax (negation): *−expression*

This arithmetic operator reverses the sign of the value of *expression*.

This is an arithmetic operator with a precedence level of 5.

Syntax (subtraction): *expression1 - expression2*

This arithmetic 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 1: (negation):

This statement reverses the sign of the expression 2 + 3:

```
put -(2 + 3)
```

The result is -5.

Example 2 (subtraction):

This statement subtracts the integer 2 from 5, and then displays the result in the Message window:

```
put 5 - 2
```

The result is 3, which is an integer.

Example 3 (subtraction):

This statement subtracts the floating-point number 1.5 from 3.25, and then displays the result in the Message window:

```
put 3.25 - 1.5
```

The result is 1.75, which is a floating-point number.

-- (comment delimiter)

Syntax: `--` [*comment*]

The comment delimiter, which is a double hyphen, indicates the beginning of a script comment. On any line, what appears between the comment symbol (double hyphen) and the end-of-line return character is interpreted as a comment instead of a Lingo statement.

Example:

This handler uses a double hyphen to make the second line a comment:

```
on resetColors
   -- This handler resets the sprite's colors.
   set the foreColor of sprite 1 to 35
      -- bright red
      set the backColor of sprite 1 to 36
         -- light blue
end resetColors
```

& (concatenator)

Syntax: *expression1* `&` *expression2*

This 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.

Note: Lingo allows some commands and functions that take only one argument to be used without parentheses surrounding the argument. This rarely presents a problem. However, when the argument includes an operator, Lingo interprets only the first argument as part of the function.

The `open window` command is one case. It allows one argument that specifies which window to open. If you use the `&` operator to define a pathame and file name, Director only interprets the string before the `&` operator as a file name. For example, Lingo interprets the statement `open window the applicationPath & "theMovie"` as `(open window the applicationPath) & ("theMovie")`. Avoid this problem by surrounding the entire phrase that includes an operator in parentheses. Thus, a successful version of the above statement would be

```
open window (the applicationPath & "theMovie")
```

The parentheses clear up Lingo's confusion by changing the precedence by which Lingo deals with the operator and treats the two parts of the argument as one complete argument.

Example 1:

This statement concatenates the strings "abra" and "cadabra":

```
put "abra" & "cadabra"
```

The result is the string "abracadabra".

Example 2:

This statement concatenates the strings "$" and the content of the variable `price`. The concatenated string is then assigned to the field cast member Price:

```
put "$" & price into field "Price"
```

&& (concatenator)

Syntax: *expression1* `&&` *expression2*

This 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 1:

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 2:

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 were Tuesday, March 15, 1996, the result would be the string `Tuesday, March 15, 1996`.

() (parentheses)

Syntax: `(`*expression)*

This grouping operator performs a grouping operation on an expression. It is used to control the order of execution of the operators in an expression, and override the automatic precedence order. It causes the expression contained within the parentheses to be evaluated first. When parentheses are nested, the contents of inner ones are evaluated before the contents of outer ones.

This is a grouping operator with a precedence level of 5.

Note: Lingo allows some commands and functions that take only one argument to be used without parentheses surrounding the argument. This rarely presents a problem. However, when the argument includes an operator, Lingo interprets only the first argument as part of the function.

The `open window` command is one case. It allows one argument that specifies which window to open. If you use the `&` operator to define a pathame and file name, Director only interprets the string before the `&` operator as a file name. For example, Lingo interprets the statement `open window the applicationPath & "theMovie"` as `(open window the applicationPath) & ("theMovie")`. Avoid this problem by surrounding the entire phrase that includes an operator in parentheses. Thus, a successful version of the above statement would be

```
open window (the applicationPath & "theMovie")
```

The parentheses clear up Lingo's confusion by changing the precedence by which Lingo deals with the operator and treats the two parts of the argument as one complete argument.

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*

This arithmetic 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 1:

This statement multiplies the integers 2 and 3, and then displays the result in the Message window:

```
put 2 * 3
```

The result is 6, which is an integer.

Example 2:

This statement multiplies the floating-point numbers 2.0 and 3.1414, and then 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*

This arithmetic 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 1:

This statement adds the integers 2 and 3, and then displays the result in the Message window:

```
put 2 + 3
```

The result is 5, which is an integer.

Example 2:

This statement adds the floating-point numbers 2.5 and 3.25, and then displays the result in the Message window:

```
put 2.5 + 3.25
```

The result is 5.75, which is a floating-point number.

/ (division)

Syntax: *expression1* / *expression2*

This arithmetic operator performs an arithmetic division on two numerical expressions, dividing *expression1* by *expression2*. If both expressions evaluate to integers, the quotient is an integer. If either or both expressions evaluate to floating-point numbers, the quotient is a floating-point number.

This is an arithmetic operator with a precedence level of 4.

Example 1:

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 2:

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*

This comparison operator compares two expressions. When *expression1* is less than *expression2*, the condition is TRUE. When *expression1* is greater than or equal to *expression2*, the condition is FALSE.

This operator can compare strings, integers, floating-point numbers, rects, and points.

This is a comparison operator with a precedence level of 1.

<= (less than or equal to)

Syntax: *expression1* <= *expression2*

This comparison operator compares two expressions. When *expression1* is less than or equal to *expression2*, the condition is TRUE. When *expression1* is greater than *expression2*, the condition is FALSE.

This operator can compare strings, integers, floating-point numbers, rects, and points.

This is a comparison operator with a precedence level of 1.

<> (not equal)

Syntax: *expression1* <> *expression2*

This comparison operator compares two expressions. When *expression1* is not equal to *expression2*, the condition is TRUE. When *expression1* is equal to *expression2*, the condition is FALSE.

This operator can compare strings, integers, floating-point numbers, rects, and points.

This is a comparison operator with a precedence level of 1. This operator also works with symbols and objects.

= (equal sign)

Syntax: *expression1* = *expression2*

This comparison operator compares two expressions or strings. When *expression1* is equal to *expression2*, the condition is TRUE. When *expression1* is not equal to *expression2*, the condition is FALSE.

This operator can compare strings, integers, floating-point numbers, rects, and points.

This is a comparison operator with a precedence level of 1. This operator also works with symbols and objects.

> (greater than)

Syntax: *expression1* > *expression2*

This comparison operator compares two expressions. When *expression1* is greater than *expression2*, the condition is TRUE. When *expression1* is less than or equal to *expression2*, the condition is FALSE.

This operator can compare strings, integers, floating-point numbers, rects, and points.

This is a comparison operator with a precedence level of 1.

>= (greater than or equal to)

Syntax: *expression1* >= *expression2*

This comparison operator compares two expressions. When *expression1* is greater than or equal to *expression2*, the condition is TRUE. When *expression1* is less than *expression2*, the condition is FALSE.

This operator can compare strings, integers, floating-point numbers, rects, and points.

This is a comparison operator with a precedence level of 1.

[] (list operators)

Syntax: `[entry1, entry2, entry3, ...]`

These square brackets specify that the entries within the brackets are a list.

There are 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 value and a property. 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 piece of machinery, associated with it. This property list could be useful for tracking how many of each piece of machinery are currently on the Stage in the 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 in order 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 are case-sensitive but strings aren't case-sensitive.

A linear list or a property list can contain no values at all. An empty list consists of two square brackets ([]). To clear a linear list, set the list to `[]`. To clear a property list, set the list to [:].

You can modify, test, or read items in a list.

You do not have to worry about explicitly disposing lists. Lists are automatically disposed when they are no longer referred to by any variable. When the list is held within a global variable, it persists from movie to movie.

You can quickly initialize a list by doing so in the `on prepareMovie` handler. An alternative way is to write the list as a field cast member and assign the list to a variable. You can then handle the list by handling the variable.

Not all PC keyboards have square brackets. If this is the case, use the `list` function to create a list.

Note: Lingo passes instances of a list as a reference to the list.

For more information about using lists, see Chapter 10, "Lists," in *Learning Lingo*.

Example 1:

This statement defines a list by making the variable `machinery` equal to the list:

```
set machinery = [#gears:6, #balls:3, #ramps:8]
```

Example 2:

This handler sorts the list "aList," and then displays the result in the Message window:

```
on sortList aList
   sort aList
   put aList
end sortList
```

If the movie issues the statement `sortList machinery`, where machinery is the list in Example 1: above, the result is `[#balls:3, #gears:6, #ramps:8]`.

Example 3:

This statement creates an empty linear list:

```
set x = []
```

Example 4:

This statement creates an empty property list:

```
set x = [:]
```

{button See also,AL(`Lingo_list_operators')}

" (string constant)

Syntax: "

When used before and after a string, quotation marks indicate that the string within the quotation marks is a literal string - not a variable, numerical value, or Lingo element. Quotation marks must always surround literal names of cast members, casts, windows, and external files.

Example:

The following statement uses quotation marks to indicate that the string "San Francisco" is a literal string. In this case, it's the name of a cast member.

```
put the loaded of member "San Francisco"
```

{button See also,AL(`Lingo_string_constant')}

¬ (continuation symbol)

Syntax: *first part of a statement on this line*¬

second part of statement ¬

third part of statement here

This special character, 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.

You can do this on several successive lines. However, one complete statement can have no more than 256 characters.

Create this character by pressing Alt+Enter.

Example:

This statement uses the ¬ character to wrap the statement onto several lines:

```
set the memberNum of sprite mySprite ¬
   to the number of member ¬
   "This is a long cast name."
```

@ (pathname operator)

Syntax: `@`

This operator defines the pathname to the current movie's folder. It has the important advantage that a pathname defined this way can be understood 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 used with the @ operator, any of these pathname separators are valid on either platform.

You can build on this pathname to specify folders that are above or below the current movie's folder.

- Add a pathname separator immediately after the @ symbol to specify a folder up in the hierarchy.
- Add folder and file names (separated by /, \, or :) after the current folder name to specify subfolders and files within folders.

Relative pathnames in Lingo are the best way to indicate the location of a linked file in a folder different than the movie's folder.

Note: Lingo allows some commands and functions that take only one argument to be used without parentheses surrounding the argument. This rarely presents a problem. However, when the argument includes an operator such as @, Lingo interprets only the first argument as part of the function.

The parentheses clear up Lingo's confusion by changing the precedence by which Lingo deals with the operator and treats the two parts of the argument as one complete argument.

Example 1:

These are equivalent expressions for the subfolder bigFolder, which is in the current movie's folder:

```
@/bigFolder

@:bigFolder

@\bigFolder
```

Example 2:

These are equivalent expressions that specify the file linkedFile, which is in the subfolder bigFolder:

```
@:bigFolder:linkedFile

@\bigFolder\linkedFile

@/bigFolder/linkedFile
```

Example 3:

This expression specifies the file `linkedFile`, which is located one level up from the current movie's folder:

```
@//linkedFile
```

This specifies the file `linkedFile`, which is located two levels up from the current movie's folder:

```
@:::linkedFile
```

Example 4:

These are equivalent expressions that specify the file `linkedFile`, which is in the folder `otherfolder`. The folder otherFolder is in the folder one level up from the current movie's folder:

```
@::otherFolder:linkedFile
@\\otherFolder\linkedFile
@//otherFolder/linkedFile
```

{button See also,AL(`Lingo_at_symbol')}

abbr, abbrev, abbreviated

These elements are used by the **date** and **time** functions.

abort

Syntax: `abort`

This command has Lingo 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
```

{button See also,AL(`Lingo_abort')}

abs

Syntax: `abs (`*numericExpression)*

This 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 tracking mouse and sprite movement by converting coordinate differences (which can be either positive or negative) into distances (which are always positive). The `abs` function is also useful for handling mathematical functions, such as `sqrt` and `log`.

Example 1:

This statement calculates the absolute value of -2.2 and displays the result in the Message window:

```
put abs(-2.2)
```

Example 2:

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. If it is, the foreground color of sprite 6 is changed.

```
if abs (the mouseV - startV) > 30 then ¬
set the foreColor of sprite 6 to 95
```

activeCastLib

Syntax: `the activeCastLib`

This 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 of 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`.

```
set castLibOfInterest to the activeCastLib

set selectedMembers to the selection of castLib castLibOfIntrest
```

An equivalent way to write this would be:

```
set selectedMembers to the selection of the castLib the activeCastLib
```

activeWindow

Syntax: `the activeWindow`

This system property indicates which movie window is currently active. For the main movie, `the activeWindow` is the Stage. For a movie in a window, `the activeWindow` is the movie in a window itself.

Example:

This example places the word Active in the title bar of the window clicked on and places the word Inactive in the title bar of all other open windows:

```
on activateWindow

   set clickedWindow = getPos(the windowlist, ¬
     the activeWindow)

   set windowCount = count(the windowlist) into windowCount

   repeat with x = 1 to windowCount

      if x = clickedWindow then

         set the title of the activeWindow to "Active"

      else

         set the title of (getAt(the windowlist,x))
           to "Inactive"

      end if

   end repeat

end
```

{button See also,AL(`Lingo_activeWindow')}

actorList

Syntax: `the actorList`

The `actorList` property is a list of child objects that have been explicitly added to the list. Objects in `the actorList` receive a `stepFrame` message each time the playback head enters a frame.

To add an object to `the actorList`, use `add the actorList, theObject.` The object's `on stepFrame` handler in its parent or ancestor scripts will then be run automatically at each frame advance.

Clear objects from `the actorList` by setting `the actorList` to `[]`, which is an empty list.

Director doesn't clear the contents of `the actorList` when branching to another movie, which can cause unpredictable behavior in the new movie. To prevent child objects in the current movie from carrying over to the new movie, insert the statement `set the actorList = []` in the `on prepareMovie` handler of the new movie.

Don't remove an object from `the actorList` in an `on stepFrame` handler. Doing this might cause unexpected results.

For more information about parent scripts and child objects, see Chapter 12, "Parent Scripts and Child Objects," in *Learning Lingo*.

Example 1:

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 2:

This statement displays the contents of `the actorList` in the Message window:

```
put the actorList
```

Example 3:

This statement clears `the actorList`:

```
set the actorList = []
```

{button See also,AL(`Lingo_abortactorList')}

add

Syntax: `add linearList, value`

This command 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 applies to linear lists only. When used on a property list, the `add` command gives an error.

For more information about lists, see Chapter 10, "Lists," in *Learning Lingo*.

Example 1:

These statements add the value 2 to the list named `bids`. The resulting list is [3, 4, 1, 2]:

```
set bids = [3, 4, 1]

add bids, 2
```

Example 2:

This statement adds 2 to the sorted linear list [1, 4, 5]. The new item remains in alphanumeric order because the list is sorted:

```
add bids, 2
```

addAt

Syntax: `addAt` *list* , *position* , *value*

This command adds a value to the list at the position specified by *position*. Use this command when you need to add an item at a specific location in a list.

The `addAt` command works with linear lists only. Using `addAt` with a property list produces a script error.

For more information about lists, see Chapter 10, "Lists," in *Learning Lingo*.

Example:

This statement adds the value 8 to the fourth position in the list named `bids`, which is [3, 2, 4, 3, 6, 7]:

```
set bids = [3, 2, 4, 5, 6, 7]

addAt bids, 4, 8
```

The resulting value of `bids` is [3, 2, 4, 8, 5, 6, 7].

addProp

Syntax: `addProp` *list* , *property* , *value*

This command 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.

When 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.

The `addProp` command works with property lists only. Using `addProp` with a linear list produces a script error.

For more information about lists, see Chapter 10, "Lists," in *Learning Lingo*.

Example 1:

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:

```
addProp bids, #kayne, 3
```

The result is the list [#gee: 4, #kayne: 3, #ohasi: 1].

Example 2:

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:

```
addProp bids, #kayne, 7
```

The result is the list [#gee: 4, #kayne: 3, #kayne:7, #ohasi: 1].

after

See: **put...after** command.

alert

Syntax: `alert` *message*

This 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 in your movie. The message can contain up to 255 characters.

The message must be a string. If you want to include a number variable in an alert, use the `string` function to handle the variable.

Example 1:

The following statement produces an alert stating that there is no CD-ROM drive connected:

```
alert "There is no CD-ROM drive ¬
connected."
```

Example 2:

This statement produces an alert stating that a file was not found:

```
alert "The file" && QUOTE & filename & QUOTE ¬
   && "was not found."
```

{button See also,AL(`Lingo_alert')}

alertHook

Syntax: `the alertHook`

This system property specifies a parent script that contains the `on alertHook` handler. When the `on alertHook` handler is in effect, the handler determines whether a projector displays alerts about file errors or Lingo script errors.

Set `the alertHook` to zero to turn off the `on alertHook` handler's capability to handle alert statements.

Example:

The following statement specifies the parent script Alert as the script that determines whether to display alerts when an error occurs:

```
on prepareMovie
  set the alertHook = script "Alert"
end
```

{button See also,AL(`Lingo_alertHook')}

alignment of member

Syntax: `the alignment of member` *whichCastmember*

This field property determines the alignment used to display characters within the specified field cast member.

The value of the property is a string consisting of one of the following: "left," "center," or "right." The parameter *whichCastmember* can be either a cast name or a cast number.

The `alignment of member` property can be tested and set.

The field cast member must contain characters, if only a space, to use the `alignment of member` property. It has no effect on a cast member that contains no characters.

For more information about working with fields, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example:

This statement sets the variable named `characterAlign` to the current `alignment of member` setting for the field cast member Rokujo Speaks:

```
set characterAlign = the alignment of member "Rokujo Speaks"
```

This repeat loop consecutively sets the `alignment` of the field cast member Rove to left, center, and then right.

```
repeat with i = 1 to 3

   set the alignment of member "Rove" ¬

   to word i of "left center right"

end repeat
```

This property requires that the field cast member already contain characters, if only a space. It will not affect a cast member that contains no characters.

{button See also,AL(`Lingo_alignment_of_member')}

ancestor

Syntax: `property {`*optionalProperties*`} ancestor`

The `ancestor` 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. This is useful 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 the child objects, the `ancestor` property is usually assigned in the `on new` handler within the parent script. When you send a message to a child object that does not have a defined handler, that message is forwarded 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 can be changed "on the fly." This allows you to 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 accessed by child objects. To refer to property variables within the ancestor script, you must use this syntax:

```
set propertyVariable of me to value
```

For example, this statement changes the property variable `legCount` within an ancestor script to 4:

```
set the legCount of me to 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
```

For more information about using the ancestor property with parent scripts and child objects, see Chapter 12, "Parent Scripts and Child Objects," in *Learning Lingo*.

For more information about using the ancestor property with behaviors, see Chapter 15, "Authoring Behaviors," in *Learning Lingo*.

Example:

The following scripts present an example of using the `ancestor` property. Each of these scripts is a cast member. Using the ancestor script `Animal` and the parent scripts `Dog` and `Man`, they 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 new
```

The second script `Man` sets the property variable `race` to African, 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 "African"
   set the ancestor of me to new(script "Animal")
   set the legCount of me to 2
   return me
end new
```

{button See also,AL(`Lingo_ancestor')}

and

Syntax: *logicalExpression1* `and` *logicalExpression2*

This logical operator determines whether two logical expressions are both TRUE. When both *logicalExpression1* and *logicalExpression2* are TRUE, the result is TRUE (1). When either or both expressions are FALSE, the result is FALSE (0).

The precedence level of this logical operator is 4.

Example 1:

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 2:

The first logical expression in this statement is TRUE; the second logical expression is FALSE. Because both logical expressions are not TRUE, the logical operator gives the result 0, which is the numerical equivalent of FALSE:

```
put 1 < 2 and 2 < 1

-- 0
```

{button See also,AL(`Lingo_and')}

append

Syntax: `append` *list*, *value*

This command 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 in accordance with the list's order.

The `append` command works with linear lists only. Using `append` with a property list produces a script error.

For more information about lists, see Chapter 10, "Lists," in *Learning Lingo*.

Example:

This statement adds the value 2 at the end of the sorted list named bids, which contains [1, 3, 4] even though this is not according to the list's sorted order:

```
set bids = [1, 3, 4]

append bids, 2

The resulting value of bids is [1, 3, 4, 2].
```

{button See also,AL(`Lingo_append')}

applicationPath

Syntax: `the applicationPath`

This property determines the path or location of the folder that contains the running copy of the Director application. The value is a string.

If you use `the applicationPath` followed by & and a path to a subfolder, surround the entire expression in parentheses so that Lingo parses the expression as one phrase.

This property can be tested but not set.

Example 1:

This statement displays the pathname for the folder that contains the Director application. The result is Z:\Program Files\Director\Movies:

```
put the applicationPath

--"Z:\Program Files\Director\Movies"
```

Example 2:

This statement opens the movie "Sunset Boulevard" as a movie in a window:

```
open window (the applicationPath & "Film Noir\Sunset¬ Boulevard")
```

atan

Syntax: `atan (`*number)*

This function calculates the arctangent, which is the angle whose tangent is the specified number. The result is a value in radians between pi/2 and +pi/2.

Example:

This statement gives the arctangent of 1:

```
atan (1)
```

The result, to four decimal places, is 0.7854, which is approximately pi/4.

{button See also,AL(`Lingo_atan')}

autoTab of member

Syntax: `the autoTab of member` *whichCastmember*

This field cast member property determines whether the editable field that follows the field cast member specified by *whichCastmember* becomes the active field after the user presses Tab. Tabbing order depends on sprite number order, not position on the Stage.

- When `the autoTab of member` is TRUE, pressing Tab makes the next editable field sprite on Stage the active field.
- When `the autoTab of member` is FALSE, pressing Tab does not make the next editable field sprite on Stage the active field.

For more information about fields, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example:

This statement makes the field that follows the field cast member Comments active after the user presses Tab:

```
set the autoTab of member "Comments" to TRUE
```

backColor of cast

This Lingo element is obsolete. Use **backColor of member** instead.

backColor of member

Syntax: `set the backColor of member` *whichCastmember* `to` *colorNumber*

This cast member property sets the background color of the specified field cast member.

The `backColor of member` value depends on the color depth of the monitor. It ranges from 0 to 255 for 8-bit color, from 0 to 15 for 4-bit color, and so on. 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.)

Example:

This statement changes the color of the characters in cast member 1 to the color in palette entry 250.

```
set the backColor of member 1 to 250
```

backColor of sprite

Syntax: `the backColor of sprite` *whichSprite*

This sprite property determines the background color of the sprite specified by *whichSprite*. Setting `the backColor` using a Lingo script is the same as choosing the background color from the tool palette when the sprite is selected on the Stage. For the value set by Lingo 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. It does not affect how a field or button cast member is displayed. An 8-bit bitmap is affected, but generally not in a useful way.

The `backColor of sprite` 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.)

The `backColor of sprite` property can be tested and set.

Example 1:

The following statement sets the variable `oldColor` to the background color of sprite 5:

```
put the backColor of sprite 5 into oldColor
```

Example 2:

The following statement randomly changes the background color of a random sprite between sprite 11 and sprite 13 to color number 36:

```
set the backColor of sprite (10 + random(3)) to 36
```

{button See also,AL(`Lingo_backColor_of_sprite')}

BACKSPACE

Syntax: `BACKSPACE`

This character constant represents the Backspace key. This key is marked "Backspace" in Windows and "delete" on the Macintosh keyboard.

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*`}`

This 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 sound is the sound assigned in the Sounds Properties dialog box.
- For the Macintosh, the beep sound is the sound selected in Alert Sounds in the Sound control panel. If the Volumes in the Sound control panel is set to 0, the menu bar flashes instead.

Example:

This statement causes two beeps if the field Answer is empty:

```
if field "Answer" = EMPTY then beep 2
```

beepOn

Syntax: `the beepOn`

This property determines whether the computer beeps when the user clicks outside an active sprite-a sprite that has a script associated with it.

If the `beepOn` property is set to TRUE, clicking outside active sprites results in a beep. The default value is FALSE.

Scripts that set the `beepOn` property should be placed in frame or movie scripts.

The `beepOn` property can be tested and set.

Example 1:

This statement sets the `beepOn` property to TRUE:

```
set the beepOn to TRUE
```

Example 2:

This statement sets `the beepOn` to the opposite of its current setting:

```
set the beepOn to (not the beepOn)
```

before

See: **put...before** command

beginRecording

Syntax: `beginRecording`

This 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.

You can't start a Score recording session from within an `on enterFrame` handler.

For more information about Lingo that generates Score, see Chapter 13, "Authoring from Lingo," in *Learning Lingo*.

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
      set horizontal = 0
      set vertical = 300
      repeat with i = 1 to numberOfFrames
         go to frame i
         set the member of sprite 20 to ¬
         member "Ball"
         set the locH of sprite 20 to horizontal
         set the locV of sprite 20 to vertical
         set the type of sprite 20 to 1
         set the foreColor of sprite 20 to 255
         set horizontal = horizontal + 3
         set vertical = vertical + 2
         updateFrame
      end repeat
   endRecording
end
```

{button See also,AL(`Lingo_beginRecording')}

behavesLikeToggle of member

Syntax: `the behavesLikeToggle of member` *whichCastmember*

This button cast member property determines whether a button behaves as a push button or a toggle.

- When `the behavesLikeToggle of member` is TRUE, the button behaves like a toggle.
- When `the behavesLikeToggle of member` is FALSE, the button behaves like a push button.

This property can be tested and set.

Example:

This statement makes the button cast member Panic behave like a toggle:

```
set the behavesLikeToggle of member "Panic" to TRUE
```

{button See also,AL(`behavesLikeToggle_member')}

behavesLikeToggle of sprite

Syntax: `the behavesLikeToggle of sprite` *whichSprite*

This button sprite property determines whether a button behaves as a push button or a toggle.

- When `the behavesLikeToggle of sprite` is TRUE, the button behaves like a toggle.
- When `the behavesLikeToggle of sprite` is FALSE, the button behaves like a push button.

This property can be tested and set.

Example:

This statement makes the button sprite number 50 behave like a toggle:

```
set the behavesLikeToggle of sprite 50 to TRUE
```

{button See also,AL(`behavesLikeToggle_sprite')}

birth

This function is obsolete. Use the **new** function instead.

bitRate of member

Syntax: `the bitRate of member` *whichCastmember*

This sound cast member property returns the bit rate of the specified Shockwave Audio (SWA) cast member that has been preloaded from the server. The value is in Kbps.

The `bitRate of member` property returns (`0`) until streaming has started.

Example:

This handler displays the bit rate of SWA cast member Louie Prima in the field cast member BitRate Display:

```
on exitFrame
  if the state of member "Louie Prima" = 2 then
    put the bitRate of member "Louie Prima"¬
    into member "BitRate Display"
  end if
end
```

bitsPerSample of member

Syntax: `the bitsPerSample of member "`*whichCastmember*`"`

This Shockwave Audio streaming cast member property indicates the bit depth of the original file that has been SWA-encoded. This property is only available after the SWA sound is 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:

```
put the bitsPerSample of member "Paul Robeson" into member "How Deep"
```

blend of sprite

Syntax: `the blend of sprite` *whichSprite*

Using this sprite property, you can set or determine the sprite's blend value. (The blend ink effect must be applied to the sprite for this property to have any significance.) For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet.

Blend values can be from 0 to 100, which correspond 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.

Example 1:

This statement sets the blend value of sprite 3 to 40 percent:

```
set the blend of sprite 3 to 40
```

Example 2:

This statement displays the blend value of sprite 3 in the Message window:

```
put the blend of sprite 3
```

border of member

Syntax: `the border of member` *whichCastmember*

This 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 ten pixels wide:

```
set the border of member "Title" to 10
```

bottom of sprite

Syntax: `the bottom of sprite` *whichSprite*

This sprite property is the bottom vertical coordinate of the bounding rectangle of the sprite specified by *whichSprite*.

The `bottom of sprite` property can be tested but not set directly. Set the `rect of sprite` property to set the bottom vertical coordinate of a sprite.

Example:

This statement assigns the vertical coordinate of the bottom of sprite numbered (i + 1) to the variable named `lowest`:

```
set lowest = put the bottom of sprite (i + 1)
```

{button See also,AL(`Lingo_bottom_of_sprite')}

boxDropShadow of member

Syntax: `the boxDropShadow of member` *whichCastMember*

This field 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 ten pixels wide:

```
set the boxDropShadow of member "Title" to 10
```

boxType of member

Syntax: `the boxType of member` *whichCastmember*

This field 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:

```
set the boxType of member "Editorial" to #scroll
```

browserName

Syntax: `browserName` *pathName*

or

`browserName()`

or

`browserName(#enabled,` *trueOrFalse*`)`

The `browserName` property determines several things, depending on the syntax used.

- The syntax `browserName` *pathName* 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 suntax `browserName()` returns the name of the currently specified browser.
- The syntax `browserName(#enabled,` *trueOrFalse*`)` determines whether the specifed browser launches automatically when the `goToNetPage` command is issued. The statement `browserName(#enabled, TRUE)` has the specifed browser launch automatically. The statement `browserName(#enabled, FALSE)` has the specified browser not launch automatically.

This property can be tested and set.

Example 1:

This expression refers to the location of the Netscape browser:

```
browserName "My Disk:My Folder:Netscape"
```

Example 2:

This statement displays the browser name in a Message window:

```
put browserName()
```

buttonStyle

Syntax: `the buttonStyle`

This property determines the visual response of buttons when a user clicks a button, and then moves the pointer over other buttons without releasing the mouse button.

The `buttonStyle` property can have these values:

- **0-list style:** When the `buttonStyle` property is set to 0 (list style), subsequent buttons highlight when the pointer passes over them. If the user releases the mouse button while the pointer is over such a button, the script associated with that button is activated.
- **1-dialog style:** When the `buttonStyle` property is set to 1 (dialog style), only the first button clicked highlights. Subsequent buttons are not highlighted. If the user releases the mouse button while the pointer is over a button other than the original button clicked, the script associated with that button is not activated.

The `buttonStyle` property can be tested and set. The default value is `0` (list style). You can set or test this property in any type of script.

Example 1:

The following statement sets the `buttonStyle` property to 1:

```
set the buttonStyle to 1
```

Example 2:

This statement remembers the current setting of the `buttonStyle` property by putting the current `buttonStyle` in the variable `buttonStyleValue`:

```
set buttonStyleValue to the buttonStyle
```

{button See also,AL(`Lingo_buttonStyle')}

buttonType

Syntax: `the buttonType of member` *whichCastmember*

This button cast member property indicates the specified button cast member's type. Possible values are `#pushButton, #checkBox`, or `#radioButton`.

Example:

This statement makes the button cast member Editorial a check box:

```
set the buttonType of member "Editorial" to #checkBox
```

cacheDocVerify

Syntax: `cacheDocVerify #setting`

or

`cacheDocVerify()`

This function sets how often the contents of a page on the internet are refreshed with information from the cache. Possible values are `#once` and `#always`. The default value is `#once`.

The form `cacheDocVerify()` returns the current setting of the cache.

The `cacheDocVerify` function is only valid for movies run in Director or as projectors. This function is not valid for Shockwave movies because Shockwave movies use the network settings of the browser in which they run.

Example:

```
on resetCache
  set current=cacheDocVerify()
  if current = #once then
    alert "Turning cache verification on"
    cacheDocVerify #always
  end if
end
```

{button See also,AL(`Lingo_cacheDocVerify')}

cacheSize

Syntax: `cacheSize` *Size*

or

`cacheSize()`

This function sets Director's cache size. The value is in kilobytes.

The form `cacheSize()` returns the cache size setting.

The `cacheSize` function is valid only for movies run in Director or as projectors. This function is not valid for Shockwave movies because Shockwave movies 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
```

{button See also,AL(`Lingo_cacheSize')}

call

Syntax: `call` *#handlerName, script, [args...]*

or

`call` *(#handlerName, scriptInstance, [args...])*

This command sends a message that invokes a handler in specified scripts.

- Replace *handlerName* with the name of the handler to be activated.
- Replace *script* with references to the script or a list of scripts.

  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. In this case, if the handler is not defined in the `ancestor` script, no alert is generated.
- Replace *args* with any optional parameters to be passed to the handler.

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, scripts of cast members, frame scripts, or movie scripts.

Example 1:

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 2:

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 tool
property ancestor

on new me
  set ancestor = new(script "Animal", 2)
  set tool = ""
  return me
end

on run me
  put "Man running with "&the legCount of me&" legs "&tool
end

on hold me, newTool
  set tool = newTool
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
-- "Running with 2 legs and rock"
```

  This statement makes 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 "
```

{button See also,AL(`Lingo_call')}

callAncestor

Syntax: `callAncestor` *#handlerName, script, [args...]*

This command sends a message to a child object's `ancestor` script.

- Replace *handlerName* with the name of the handler to be activated.
- Replace *script* with references to the script or a list of scripts.

  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 scripts, the message is sent to each item in the list in turn. In this case, if the handler is not defined in the `ancestor` script, no alert is generated.
- Replace *args* with any optional parameters to be passed to the handler.

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 directly access 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 tool
property ancestor

on new me
  set ancestor = new(script "Animal", 2)
  set tool = ""
  return me
end

on run me
  put "Man running with "&the legCount of me&" legs "&tool
end

on hold me, newTool
  set tool = newTool
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 the ancestor script for both men:

```
callAncestor #run,[m,m2]
-- "Animal running with 2 legs "
-- "Animal running with 2 legs "
```

{button See also,AL(`Lingo_call_ancestor')}

cancelIdleLoad

Syntax: `cancelIdleLoad` *loadTag*

This command cancels the loading of all cast members that have the specified load tag.

Example:

This statement cancels loading cast members that have the idle load tag 20:

```
cancelIdleLoad 20
```

{button See also,AL(`Lingo_cancelIdleLoad')}

case

Syntax: `case` *case expression* `of`

expression1 : *Statement(s)*

expression2 :

multipleStatements

.
.
.

expression3, *expression4* :

Statement(s)

{`otherwise` *statement(s)*}

`end case`

This 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. The comparison starts at the beginning and continues through each line in order until Lingo encounters an expression that matches *case expression*.

When a matching expression is found, Lingo 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 appear on the same line. Multiple statements appear on indented lines immediately below the matching expression.

When there is more than one possible match that causes Lingo to execute the same statements, the expressions must be separated by commas. (The 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 statement following `otherwise` is executed if there are no matches.

Example 1:

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 2:

This case statement tests whether the cursor is over sprite 1, 2, or 3 and runs the corresponding Lingo if it is:

```
case rollover() of
  1: puppetSound "Horn"
  2: puppetSound "Drum"
```

```
  3: puppetSound "Bongos"
end case
```

cast

This is obsolete. Use **member** instead.

castLib

Syntax: `castLib` *whichCast*

This 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 `the castLib` to specify the alternate cast.

Example 1:

This statement displays the number of the cast Buttons in the Message window:

```
put the number of castLib "Buttons"
```

Example 2:

This statement assigns cast member 5 in cast number 4 to sprite 10:

```
set the member of sprite 10 to member 5 of castLib 4
```

castLibNum of member

Syntax: `the castLibNum of member` *whichCastMember*

This cast member property determines the number of the cast that contains the specified cast member. This property can be tested but not set.

Example:

This statement determines the number of the cast that cast member Jazz is assigned to:

```
put the castLibNum of member "Jazz"
```

castLibNum of sprite

Syntax: `the castLibNum of sprite` *whichSprite*

This sprite property determines the number of the cast that contains the cast member assigned to the specified sprite. This property can be tested and set.

If you change `the castLibNum of sprite` without changing `the memberNum of sprite`, 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 of sprite` property.

Example:

This statement changes the cast member assigned to sprite 5 by switching its cast to "Wednesday Schedule":

```
set the castLibNum of sprite 5 to the number ¬
of castLib "Wednesday Schedule"
```

{button See also,AL(`Lingo_castLibNum_of_sprite')}

castmembers

This is obsolete. Use **number of members** instead.

castNum of sprite

This sprite property is now obsolete. It was used in earlier versions of Director to identify which cast member was assigned to a sprite. Use **member of sprite** or **memberNum of sprite** instead.

castType of cast

This is obsolete. Use **type of member** instead.

center of cast

This is obsolete. Use **center of member** instead.

center of member

Syntax: `the center of member` *whichCastmember*

This movie and digital video cast member property interacts with the `crop of member` cast member property. It can be tested and set.

- When `the crop of member` is FALSE, `the center of member` has no effect.
- When `the crop of member` is TRUE and `the center of member` is TRUE, cropping occurs around the center of the digital video cast member.
- When `the crop of member` is TRUE and `the center of member` is FALSE, the digital video's right and bottom sides are cropped.

For more information about controlling digital video, see Chapter 7, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement causes the digital video cast member Interview to be displayed in the top left corner of the sprite:

```
set the center of member "Interview" to FALSE
```

{button See also,AL(`Lingo_center_of_member')}

centerStage

Syntax: `the centerStage`

This property determines whether the Stage is centered on the monitor when the movie is loaded. Place the statement that includes this property in the movie that precedes the movie you want it to affect.

- If `the centerStage` is TRUE, the Stage is centered.
- If `the centerStage` is FALSE, the Stage is not centered.

This property is useful for checking Stage location before a movie plays from a projector.

The `centerStage` property can be tested and set. The default value is TRUE.

Example 1:

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 2:

This statement changes the `centerStage` property to the opposite of its current value:

```
set the centerStage to (not the centerStage)
```

{button See also,AL(`Lingo_centerStage')}

changeArea of member

Syntax: `the changeArea of member` *whichCastMember*

This transition cast member property determines whether the transition applies to the changing area on the Stage. It can be tested and set. Its effect is similar to selecting the Changing Area Only option in the **Frame Properties Transition** dialog box.

- When `the changeArea of member` is TRUE, the transition applies to the changing area only.
- When `the changeArea of member` is FALSE, the transition applies to the entire Stage.

Example:

This statement makes the transition cast member Wave apply only to the changing area on the Stage:

```
set the changeArea of member "Wave" to TRUE
```

channelCount of member

Syntax: `the channelCount of member` *whichCastmember*

This sound cast member property determines the number of channels in the specified cast member. This is useful for determining whether a sound is in mono or stereo. This property can be tested but not set.

Example:

This statement determines how many channels are in the sound cast member Jazz:

```
put the channelCount of member "Jazz"
```

char...of

Syntax: `char` *whichCharacter* `of` *chunkExpression*

`char` *firstCharacter* `to` *lastCharacter* `of` *chunkExpression*

This chunk expression 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 like TAB and RETURN.

You can test but not set the `char...of` keyword.

Example 1:

This statement displays the first character of the string $9.00:

```
put char 1 of "$9.00"
-- "$"
```

Example 2:

This statement displays the entire string $9.00:

```
put char 1 to 5 of "$9.00"
-- "$9.00"
```

Example 3:

This statement changes the first five characters of the second word of the third line of a field cast member:

```
set char 1 to 5 of word 2 of line 3 ¬
   of member "quiz" to "?????"
```

{button See also,AL(`Lingo_char_of')}

charPosToLoc

Syntax: `charPosToLoc(member` *whichCastMember*`,` *nthCharacter*`)`

This function gives the point in the specified field cast member 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. The point is the point in the entire field cast member, not the part of the field cast member that appears on the Stage.

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:

```
set location to charPosToLoc(member "Headline", 50)
```

chars

Syntax: `chars(`*stringExpression* `,` *firstCharacter* `,` *lastCharacter)*

This 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 gives the value EMPTY.

Example 1:

This statement identifies the sixth character in the word Macromedia:

```
put chars("Macromedia", 6, 6)

-- "m"
```

Example 2:

This statement identifies the sixth through tenth characters of the word Macromedia:

```
put chars("Macromedia", 6, 10)

-- "media"
```

Example 3:

This statement attempts to identify the sixth through twentieth characters of the word Macromedia. Because the word has only ten characters, the result includes only the sixth to tenth characters.

```
put chars ("Macromedia", 6, 20)

-- "media"
```

{button See also,AL(`Lingo_chars')}

charToNum

Syntax: `charToNum(`*stringExpression)*

This function returns the ASCII code number 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 one other alphanumeric key.

A problem can arise when testing characters. Director treats upper- and lower-case letters the same if you compare them using the equal sign (=) operator.

For example, the statement `put ("M" = "m")` gives 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.

For a demonstration of the `charToNum` function, see the sample movie **Keyboard Lingo**

Example 1:

This statement displays the ASCII code number for the letter A:

```
put charToNum("A")

-- 65
```

Example 2:

This statement checks whether 0 is the ASCII code number of the character assigned to the variable `nextChar`:

```
if charToNum(nextChar) = 0 then foundNUL
```

{button See also,AL(`Lingo_charToNum')}

checkBoxAccess

Syntax: `the checkBoxAccess`

This property determines what happens when the user clicks a check box or radio button created with button tools in the Tools window. There are three possible results:

0-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 only be set by scripts.

The default value is 0.

The `checkBoxAccess` property can be tested and set.

Example 1:

This statement sets the `checkBoxAccess` property to 1, which lets the user click check boxes and radio buttons on but not off:

```
set the checkBoxAccess to 1
```

Example 2:

This statement records the current setting of the `checkBoxAccess` property by putting the value in the variable `oldAccess`:

```
set oldAccess to the checkBoxAccess
```

{button See also,AL(`Lingo_checkBoxAccess')}

checkBoxType

Syntax: `the checkBoxType`

This system property determines what is inserted in check boxes to indicate whether they are selected. There are three possible styles:

0-Creates a standard check box that contains an "x" when the check box is selected. This is the default.

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.

The default value is 0.

The `checkBoxType` 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.

```
set the checkBoxType to 1
```

{button See also,AL(`Lingo_checkBoxType')}

checkMark of menuItem

Syntax: `the checkMark of menuItem` *whichItem* `of menu` *whichMenu*

This menu item property determines whether the specified custom menu item is displayed with a checkmark.

- When it is TRUE, a checkmark appears next to the custom menu item.
- When it is FALSE, no checkmark appears.

The default value is FALSE.

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.

The `checkMark of menuItem` property can be tested and set.

Example:

This handler unchecks any items that are checked in the custom menu specified by the argument `theMenu`. For example, `unCheck` ("Format") unchecks all the items in the Format menu:

```
on unCheck theMenu
   put the number of menuItems of menu theMenu into n
   repeat with i = 1 to n
      set the checkMark of menuItem i of menu ¬
      theMenu to FALSE
   end repeat
end unCheck
```

{button See also,AL(`Lingo_checkMark_of_menuItem')}

chunkSize of member

Syntax: `the chunkSize of member` *whichCastMember*

This transition cast member property, which determines the transition's chunk size, does the same as setting the smoothness slider in the Frame Properties: Transition dialog box.

This property can be tested and set.

Values are the number of pixels in each chunk of the transition and can be any value from 1 to 128 pixels.

Example:

This statement sets the chunk size of the transition cast member Fog to 4 pixels:

```
set the chunkSize of member "Fog" to 4
```

clearCache

Syntax: `clearCache`

This command clears Director's network cache.

The `clearCache` command is valid only for movies that play in Director or as projectors. This command isn't valid for Shockwave movies because Shockwave movies use the network settings of the browser in which they run.

If a file is in use, it isn't cleared from the cache.

Example:

This handler clears the cache when the mouse button is released:

```
on mouseUp
  clearCache
end
```

{button See also,AL(`Lingo_clearCache')}

clearFrame

Syntax: `clearFrame`

This command erases everything already in the current frame's sprite and effects channels. It works during Score recording only.

For more information about generating Score from Lingo, see Chapter 12, "Authoring from Lingo," In *Learning Lingo*.

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
                set the frameScript to 25
                updateFrame
          end repeat
   endRecording
end
```

{button See also,AL(`Lingo_clearFrame')}

clearGlobals

Syntax: `clearGlobals`

This command sets all global variables to `VOID`.

This 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`

This function identifies the last place on the screen where the mouse was clicked. The location is given as a point.

For more information about detecting what the user clicks, see Chapter 7, "Working with Fields and User Input," In *Learning Lingo*.

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)
```

clickOn

Syntax: `the clickOn`

This 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, `the clickOn` equals `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` function can be checked within a repeat loop. However, neither `the clickOn` nor `the clickLoc` functions change value when the handler is running. The value that you obtain is the value from before the handler started.

For more information about detecting what the user clicks, see Chapter 7, "Working with Fields and User Input," In *Learning Lingo*.

Example 1:

This statement checks whether sprite 7 was the last active sprite clicked:

```
if the clickOn = 7 then alert "Sorry, try again."
```

Example 2:

This statement sets `the foreColor` of the last active sprite that was clicked to a random color:

```
set the foreColor of sprite (the clickOn) to ¬
   random(255)-1
```

{button See also,AL(`Lingo_clickOn')}

close window

Syntax: `close window` *windowIdentifier*

This 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 the `windowList`, use the syntax `close window` *number*, where you replace *number* with the window's number in `the windowList`.

Lingo allows you to attempt to close a window that is already closed.

For more information about using a movie in a window, see Chapter 10, "Movies in a Window," In *Learning Lingo*.

Example 1:

This statement closes the window named Panel, which is in the subfolder MIAW Sources within the current movie's folder:

```
close window "@/MIAW Sources/Panel"
```

Example 2:

This statement closes the window that is number 5 in `the windowList`:

```
close window 5
```

{button See also,AL(`Lingo_close_window')}

closeDA

This Lingo element is obsolete.

closeResFile

Syntax: `closeResFile [`*whichFile*`]`

This command, which closes the resource file on the Macintosh, is now obsolete.

{button See also,AL(`Lingo_closeResFile')}

closeXlib

Syntax: `closeXlib [`*whichFile*`]`

This command closes the Xlibrary file specified by the string *whichFile*. If the Xlibrary file is in a folder other than the current movie, *whichFile* must specify a pathname. If no file is specified, all open Xlibraries are closed.

Xtras and **XObjects** are stored in Xlibrary files. Xlibrary files are resource files that contain XCOD (**XObjects**) resources. 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 **XObjects** is optional.

It is good practice to close any file you have opened as soon as you are finished using it.

Example 1:

This statement closes all open Xlibrary files:

```
closeXlib
```

Example 2:

This statement closes the Xlibrary Video Disc Xlibrary when it is in the same folder as the movie:

```
closeXlib "Video Disc Xlibrary"
```

Example 3:

This statement closes the Xlibrary Transporter Xtras in the folder New Xtras on the same disk as the movie. The disk is identified by the variable `currentDrive`:

```
closeXlib currentDrive & ¬
   ":New Xtras:Transporter Xtras"
```

{button See also,AL(`Lingo_closeXlib')}

colorDepth

Syntax: `the colorDepth`

This property determines the color depth of the computer's monitor.

- In Windows, using this property lets you check the monitor's color depth. Setting `the colorDepth` takes effect only if the change can be made without restarting the computer. Always verify that the color depth actually changed after attempting to set it.
- On the Macintosh, using 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 colors
32	16,777,216 colors

If you try to set a monitor's color depth to a value that is impossible for the monitor, the monitor's color depth doesn't change.

On computers with more than one monitor, the `colorDepth` property refers to the monitor that the Stage is on. If the Stage spans more than one monitor, `the colorDepth` indicates the greatest depth of those monitors; setting `the colorDepth` attempts to put all those monitors to the specified depth.

The `colorDepth` property can be tested and set. On the Macintosh, the default value is the value set in the Monitors control panel.

Example 1:

This statement makes playing the segment "Full color" dependent on whether the monitor color depth is set to 256 colors:

```
if the colorDepth = 8 then play movie "Full color"
```

Example 2:

This handler attempts to change the color depths. If the change is not successful, an alert appears.

```
on tryToSetColorDepth desiredDepth

   set 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 tryToSetColorDepth
```

When changing the user's color depth, it is good practice to restore the original depth when the movie is finished.

On Windows, the command `set the colorDepth = 0` restores the user's preferred settings from the control panel.

{button See also,AL(`Lingo_colorDepth')}

colorQD

Syntax: `the colorQD`

This function is now obsolete. It was used in earlier versions of Director to indicate whether the Color QuickDraw software was available on a Macintosh. Computers that are currently supported always give TRUE for `the colorQD`.

To determine whether a computer is black and white, test `the colorDepth` instead.

{button See also,AL(`Lingo_colorQD')}

commandDown

Syntax: `the commandDown`

This function determines whether the Command key is being pressed on the Macintosh or the Control key is being pressed in Windows.

- The `commandDown` function is TRUE when the Command key is being pressed on the Macintosh or the Control key is being pressed in Windows.
- The `commandDown` function is FALSE when the Command key is not being pressed on the Macintosh or the Control key is not being pressed in Windows.

You can use `the commandDown` together with the element `the key` to determine when the Macintosh Command key or the Windows Control key is pressed in combination with another key. This lets you create handlers that are executed when the user presses specified Command-key or Control-key combinations.

Note that Command-key and Control-key equivalents for Director's authoring menus take precedence while playing the movie, unless you have installed custom Lingo menus or are playing a projector version of the movie.

For a demonstration of modifier keys in Lingo, see the sample movie **Keyboard Lingo**.

Example:

These statements pause a projector whenever the user presses Command-A on the Macintosh or Control-A in Windows. 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 Command or Control and P keys are pressed at the same time and pauses the movie if they are:

```
on prepareMovie

   set the keyDownScript to "doCommandKey"

end prepareMovie

on doCommandKey

   if (the commandDown) and (the key = "a") then go to the frame

end
```

{button See also,AL(`Lingo_commandDown')}

constrainH

Syntax: `constrainH (`*whichSprite* , *integerExpression)*

This function evaluates *integerExpression*, and then gives a value that depends on the horizontal coordinates of the left and right edges of *whichSprite*.

- 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 is changed to the value of the left coordinate.
- When the value is greater than the right horizontal coordinate, the value is changed to the value of the right coordinate.

The `constrainH` and `constrainV` functions constrain only one axis each; the `constraint of sprite` property limits both. Note that this function does not change the sprite's properties.

For more information about manipulating sprites on Stage, see Chapter 6, "Manipulating Sprites," in *Learning Lingo*.

Example 1:

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 2:

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)
```

{button See also,AL(`Lingo_constrainH')}

constraint of sprite

Syntax: `the constraint of sprite` *whichSprite*

This sprite property determines the constraints on the position of the sprite specified by *whichSprite*. When the `constraint of sprite` property is turned on, the sprite specified by *whichSprite* is constrained to the bounding rectangle of another sprite.

The `constraint of 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.

To remove a constraint of sprite property, set it to 0:

```
set the constraint of sprite whichSprite to 0
```

The `constraint of sprite` property can be tested and set. The default value is 0.

The `constraint of sprite` property is useful for constraining a moveable sprite to the bounding rectangle of another sprite. This is a way to simulate a "track" for a slider control or restrict where on the screen a user can drag an object in a game.

For more information about manipulating sprites on Stage, see Chapter 6, "Manipulating Sprites," in *Learning Lingo*.

Example 1:

This statement constrains sprite (i + 1) to the boundary of sprite 14.

```
set the constraint of sprite (i + 1) to 14
```

Example 2:

This statement checks whether sprite 3 is constrained and activates the handler `showConstraint` if it is. (The operator `<>` performs the same operation as "not equal to.")

```
if the constraint of sprite 3 <> 0 then ¬
   showConstraint
```

{button See also,AL(`Lingo_constraint_of_sprite')}

constrainV

Syntax: `constrainV (`*whichSprite* , *integerExpression*`)`

This function evaluates *integerExpression*, and then gives a value that depends on the vertical coordinates of the top and bottom edges of the sprite specified by *whichSprite*.

- 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 is changed to the value of the top coordinate.
- When the value is greater than the bottom coordinate, the value is changed to the value of the bottom coordinate.

Note that this function does not change the sprite properties.

For more information about manipulating sprites on Stage, see Chapter 6, "Manipulating Sprites," in *Learning Lingo*.

Example 1:

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 2:

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 locV of sprite 1 to ¬

   constrainV(2, the mouseV)
```

{button See also,AL(`Lingo_constrainV')}

contains

Syntax: *stringExpression1* contains *stringExpression2*

This operator compares two strings.

- When *stringExpression1* contains *stringExpression2*, the condition is TRUE (1).
- When *stringExpression1* does not contain *stringExpression2*, the condition is FALSE (0).

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.

For more information about manipulating fields, see Chapter 7, "Working with Fields and User Input" in *Learning Lingo*.

Example:

This statement determines whether a string contains the character contained in aLetter before converting the string using the `value()` function:

```
on isNumber aLetter

   put "1234567890." into digits

   if digits contains aLetter then

      return TRUE

   else
      return FALSE

   end if

end isNumber
```

Note: The string comparison is not sensitive to case or diacritical marks; "a" and "□" are treated the same.

{button See also,AL(`Lingo_contains')}

continue

Syntax: `continue`

The `continue` command was used in earlier versions of Director to resume playing the movie after a pause. It is no longer recommended. Use `go to the frame +1` instead.

For more information about how to pause and continue a movie, see Chapter 3, "Navigation," in *Learning Lingo*.

controlDown

Syntax: `the controlDown`

This function determines whether the Control key on a Macintosh or the Control key on a Windows computer is being pressed.

- The `controlDown` function is TRUE when the Control key is being pressed.
- The `controlDown` function is FALSE when the Control key is not being pressed.

You can use the `controlDown` function together with `the key` to check for combinations of the Control key and another key.

For a demonstration of modifier keys and Lingo, see the sample movie **Keyboard Lingo**.

Example:

This `on keyDown` handler checks whether the key that is pressed is the Control key and activates the `on doControlKey` handler if it is. 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
```

{button See also,AL(`Lingo_controlDown')}

controller of cast

This is obsolete. Use **controller of member** instead.

controller of member

Syntax: `the controller of member` *castName*

This 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 of member` property applies to QuickTime and QuickTime for Windows digital video only.

- Setting `the controller of member` for a Video for Windows digital video performs no operation and generates no error message.
- Checking the `controller of member` property for a Video for Windows digital video always returns FALSE.

The digital video must be in `directToStage` playback mode in order to display the controller.

For more information about controlling digital video, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement causes the QuickTime cast member to Demo show its controller:

```
set the controller of member "Demo" to 1
```

{button See also,AL(`Lingo_controller_of_member')}

copyrightInfo of member

Syntax: `copyrightInfo of member "`*whichCastmember*`"`

This property is the copyright text in a Shockwave Audio (SWA) file. This property is available only after the SWA sound is playing or after the file has been preloaded using the `preLoadBuffer` command.

This property can be tested. In SoundEdit16, version 2.0.4 or higher, this property can be set.

Example:

This statement has Director display the copyright information of the Shockwave Audio file SWAfile in a field cast member named Info Display:

```
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: `copyToClipBoard member` *whichCastmember*

This command copies the specified cast member to the Clipboard. You can use this command to copy cast members between movies or applications. The Cast window does not need to be the active window when you use the `copyToClipBoard` command.

This is best used during authoring and not in projectors. Use in projectors can cause memory problems.

Example 1:

This statement copies the cast member named chair to the Clipboard:

```
copyToClipBoard member "chair"
```

Example 2:

This statement copies cast member number 5 to the Clipboard:

```
copyToClipBoard member 5
```

{button See also,AL(`Lingo_copyToClipBoard')}

cos

Syntax: `cos (`*angle)*

This function calculates the cosine of the specified angle. The angle 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 cos (pi() /2)
```

{button See also,AL(`Lingo_cos')}

count

Syntax: `count (`*list*`)`

`count(`*theObject*`)`

This function returns the number of entries in a list or the number of properties in a parent script. When counting properties in a parent script, it doesn't count properties in an ancestor script.

The `count` command works with linear and property lists.

For more information about lists, see Chapter 10, "Lists," in *Learning Lingo*.

Example:

This statement displays the number 3, the number of entries:

```
put count ( [10, 20, 30])
-- 3
```

cpuHogTicks

Syntax: `the cpuHogTicks`

This global property controls how often Director releases control of the CPU to allow the computer to process background events such as events in other applications, network events, clock updates, and other keyboard events. Its default setting is 20 ticks.

To give more time to Director, set `cpuHogTicks` to a higher value, so that there is more time between releasing the CPU to background events. This can be useful for controlling how the computer responds to network operations.

Another possible use is in a movie that has the user hold 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. You can create faster auto-repeating key performance by setting `cpuHogTicks` to a lower value. However, this can slow down animation.

The `cpuHogTicks` property works only on the Macintosh.

Example:

This statement has Director release control of the CPU every 6 ticks, which is one-tenth of a second:

```
set the cpuHogTicks = 6
```

crop of cast

This is obsolete. Use **crop of member** instead.

crop of member

Syntax: `the crop of member` *whichCastmember*

This cast member property affects how the digital video cast member is displayed inside a sprite when the digital movie is larger than the sprite that it appears in. It can be tested and set.

- When the `crop of member` property is FALSE, the cast member is scaled-either stretched or shrunk-to fit inside the sprite rectangle.
- When the `crop of member` property is TRUE, the cast member is not scaled. It is cropped to fit inside the sprite rectangle.

For more information about controlling digital video, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement instructs Lingo to crop any sprite that refers to the digital video cast member Interview:

```
set the crop of member "Interview" to TRUE
```

{button See also,AL(`Lingo_crop_of_member')}

cuePointNames of member

Syntax: `the cuePointNames of member` *whichCastmember*

This property provides a list of cue point names. If a cue point is not named, an empty string ("") is inserted as a placeholder in the list.

This property is supported by sound cast members created in SoundEdit, QuickTime digital video cast members, and Xtras cast members that contain cue points. The list of cue point names may not be available for Xtras that generate cue points at runtime.

Cue point names are useful for synchronizing sound, QuickTime, and animation. For more information about synchronizing media, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement obtains the name of the third cue point of a cast member:

```
put (getAt(the cuePointNames of member "symphony"), 3)
```

{button See also,AL(`Lingo_cuePointNames')}

cuePointTimes of member

Syntax: `the cuePointTimes of member` *whichCastmember*

This property provides a list of the times of the cue points, in milliseconds, of a given cast member.

This property is supported by sound cast members created in SoundEdit, QuickTime digital video cast members, and Xtras cast members that support cue points. This list of cue points may not be available for Xtras that generate cue points at runtime.

Cue point times are useful for synchronizing sound, QuickTime, and animation. For more information about synchronizing media, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement obtains the time of the third cue point of a sound cast member:

```
put getAt(the cuePointTimes of member "symphony"), 3)
```

{button See also,AL(`Lingo_cuePointTimes')}

currentSpriteNum

Syntax: `the currentSpriteNum`

This property indicates the number of the sprite where the current event occurred.

The `currentSpriteNum` property is similar to `spriteNum of me`, but it doesn't require the `me` reference. It is useful in cast member, frame, and movie scripts for identifying which sprite was involved in an event. (Cast member, frame, and movie scripts have no way to determine which sprite is involved in an event because they have no access to the sprite's `me` variable.)

For example, if a script of a cast member includes an `on mouseDown` handler, the script can use `the currentSpriteNum` to determine whether the `on mouseDown` event was passed to the sprite level either because there was no `on mouseDown` event handler in the sprite scripts, or because the event was passed.

This property can be tested and set.

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
  set the member of sprite the currentSpriteNum = ¬
  member   "DownPict"
end
```

{button See also,AL(`Lingo_currentSpriteNum')}

currentTime

Syntax: `the currentTime`

This property returns the current playing time of a sound sprite, QuickTime digital video sprite, or any Xtra that supports cue points. The time is in milliseconds.

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 shows the current time, in seconds, of the sound sprite in sprite channel 10:

```
set the text of member "time" to (the currentTime of sprite 10) / 1000
```

{button See also,AL(`Lingo_currentTime')}

cursor

Syntax: `cursor [`*castNumber* `,` *maskCastNumber* `]`

or

`cursor` *whichCursor*

This command changes the cast member that is used for a cursor. The `cursor` command stays in effect until you turn it off by setting the cursor to zero.

Use the syntax `cursor` [*castNumber* , *maskCastNumber*] to specify the number of a cast member to use as a cursor and its optional mask. The hot spot of the cursor 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 x 16 pixels, Director crops it to a 16 x 16 square, starting in the upper left corner of the image. The bitmap's registration point is the cursor's hot spot.

Use the syntax `cursor` *whichCursor* to use the default cursors that are supplied by the system. The term *whichCursor* must be an integer that specifies the appearance of the cursor. The following values specify cursors:

- **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

To hide the cursor, set the cursor to 200 (a blank cursor).

During system events such as loading a file, the operating system may display the watch cursor, and then change to the pointer cursor when returning control to the application. This overrides the `cursor` command settings from the previous movie. Therefore, in a presentation using a custom cursor for multiple movies, store any special cursor resource number as a global variable. Global Lingo variables remain in memory between movies. This allows you to use the `cursor` command at the beginning of any new movie that is loaded.

Cursor commands can be interrupted by an Xtra or other external agent. If the cursor is set to a value in Director and something else takes control of the cursor, resetting the cursor to the original value has no effect because Director doesn't perceive that the cursor changed. You can work around this by explicitly setting the cursor to a third value and then resetting it to the original value.

For more information about custom cursors, see Chapter 9, "Creating User Interfaces," in *Learning Lingo*.

Notes:

- In Windows, a cursor can't be a resource; it must be a cast member. If a cursor isn't available in Windows because it hasn't been converted from a resource to a cast member, Lingo uses the standard arrow cursor instead. It is recommended that you don't make custom cursors resources when you create movies that you intend to play on both Macintosh and Windows computers.
- Be sure not to confuse `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.

Example:

On the Macintosh, this statement changes the cursor to a watch cursor 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
```

{button See also,AL(`Lingo_cursor')}

cursor of sprite

Syntax: `the cursor of sprite` *whichSprite* `to [`*castNumber*`,` *maskCastNumber*`]`

`the cursor of sprite` *whichSprite* `to` *whichCursor*

On the Macintosh, this sprite property determines the cursor resource that is used when the pointer is over the sprite specified by the integer expression *whichSprite*. The `cursor of sprite` property stays in effect until you turn it off by setting the cursor to 0.

When you set `the cursor of sprite` 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 of sprite` property for that channel to 0.

Note: In Windows, a cursor can't be a resource; it must be a cast member. If a cursor isn't available in Windows because it hasn't been converted from a resource to a cast member, Lingo uses the standard arrow cursor instead. It is recommended that you don't make custom cursors resources when you create movies that you intend to play on both Macintosh and Windows computers.

The `cursor of sprite` property is an integer that specifies the resource ID number of the cursor. The following cursors are always available:

0	no cursor set; uses system default
-1	arrow (pointer) cursor
1	I-beam cursor
2	crosshair cursor
3	crossbar cursor
4	watch cursor
200	blank cursor

To hide the cursor, set `the cursor` to 200 (a blank cursor resource).

To use custom cursors, set the `cursor of sprite` property to a list that contains the cast member to use as the cursor or the number that specifies a system cursor.

The `cursor of sprite` property is useful for changing the cursor when the mouse pointer is over specific regions of the screen. You can use this to indicate regions where certain actions are possible when the user clicks.

If the sprite is a bitmap that has Matte ink applied, the cursor changes only when the cursor is over the matted portion of the sprite.

When the cursor is over the location of a sprite that has been removed, the rollover still occurs. Avoid this problem by not performing rollovers over 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.

The `cursor of sprite` property can be tested and set.

For more information about custom cursors, see Chapter 9, "Creating User Interfaces," in *Learning Lingo*.

Example:

This statement changes the cursor that appears over sprite 20 to a watch cursor:

```
set the cursor of sprite 20 to 4
```

{button See also,AL(`Lingo_cursor_of_sprite')}

date

Syntax:

```
the abbr date
the abbrev date
the abbreviated date
the date
the long date
the short date
```

This function gives the current date in the system clock in one of three formats: `abbreviated`, `long`, or `short`. If no format is specified, the default is short. The abbreviated format can also be referred to as `abbrev` and `abbr`.

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 1:

This statement gives the abbreviated date:

```
put the abbreviated date
-- "Sat, Sep 7, 1991"
```

Example 2:

This statement gives the long date:

```
put the long date
-- "Saturday, September 7, 1991"
```

Example 3:

This statement gives the short date:

```
put the short date
-- "9/7/91"
```

Example 4:

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.

{button See also,AL(`Lingo_date')}

delay

Syntax: `delay` *numberOfTicks*

This command makes the playback head stand still for a given amount of time. The integer expression *numberOfTicks* specifies the number of ticks to wait. The only mouse and keyboard activity possible during this time is to stop the movie by pressing Control+Alt+Period.

The `delay` command can be applied only when the playback head is moving. However, when delay is in effect, handlers can still run. Place scripts using 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 `the timer` to a variable and wait for an amount of time to pass before exiting the frame. (An example is shown in the following set.)

Because it increases the time of individual frames, the `delay` command is useful for controlling the playback rate of a sequence of frames.

Example 1:

This handler delays the movie for 2 seconds when the playback head exits the current frame:

```
on exitFrame
  delay 2 * 60
end exitFrame
```

Example 2:

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 keyDown
```

Example 3:

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 specifed 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
```

Note: The `delay` command does not function when the playback head is not moving.

{button See also,AL(`Lingo_delay')}

delete

Syntax: `delete` *chunkExpression*

This 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.

Example 1:

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 2:

This statement deletes the first character of the string in the variable `bidAmount`, if that character is the dollar sign ($).

```
if char 1 of bidAmount = "$" then delete char 1 ¬
   of bidAmount
```

{button See also,AL(`Lingo_delete')}

deleteAll

Syntax: `deleteAll` *list*

This command deletes all items in the specified list.

The empty list keeps the same type. A linear list remains a linear list; a property list remains a property list.

For more information about lists, see Chapter 10, "Lists," in *Learning Lingo*.

Example:

This statement deletes every item in the list named propList:

```
deleteAll propList
```

deleteAt

Syntax: `deleteAt` *list* , *number*

This command deletes the item in the position specified by number from the list specified by *list*. The value *number* is the item's position in the order of the list.

If you try to delete an object that isn't in the list, Director displays an alert. You can avoid this by first checking whether the item is in the list.

The `deleteAt` command works with linear and property lists.

For more information about lists, see Chapter 10, "Lists," in *Learning Lingo*.

Example:

This statement deletes the second item from the list named designers, which contains ["gee", "kayne", "ohashi"]:

```
set designers = ["gee", "kayne", "ohashi"]
deleteAt designers, 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 count( theList ) < theIndex then
       beep
   else
      deleteAt theList, theIndex
   end if
end
```

{button See also,AL(`Lingo_deleteAt')}

deleteFrame

Syntax: `deleteFrame`

This command deletes the current frame. After the current frame is deleted, the next frame becomes the new current frame.

The `deleteFrame` command works during a Score generation session only.

For more information about generating Score from Lingo, see Chapter 13, "Authoring from Lingo" in *Learning Lingo*.

Example:

The following handler checks whether the sprite in channel 10 of the current frame has gone past the right edge of a 640 x 480 Stage and deletes the frame if it has:

```
on testSprite
   beginRecording
    if the locH of sprite 10 > 640 ¬
    then deleteFrame
   endRecording
end
```

{button See also,AL(`Lingo_deleteFrame')}

deleteOne

Syntax: `deleteOne` *`list`*`,` *`value`*

This command deletes a value from a linear or property list. If the value appears in the list more than once, `deleteOne` deletes only the first occurrence.

When the list is a property list, the property associated with the deleted value is also removed from the list.

Attempting to delete a property has no effect.

For more information about lists, see Chapter 10, "Lists," in *Learning Lingo*.

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.

```
set days = ["Tuesday", "Wednesday", "Friday"]

deleteOne days, "Wednesday"

put days

-- ["Tuesday", "Friday"]
```

The `put days` statement causes the Message window to display the result, which is: `["Tuesday", "Friday"]`.

deleteProp

Syntax: `deleteProp` *list, item*

This command deletes the specified item from the specified list.

- For linear lists, replace *item* with a number for the list position of the item to delete. 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 delete. Deleting a property deletes its associated value also. If the list has more than one of the same property, only the first property in the list is deleted.

For more information about lists, see Chapter 10, "Lists," in *Learning Lingo*.

Example:

This statement deletes the property color from the list [#height:100, #width: 200, #color: 34, #ink: 15], which is called `spriteAttributes`:

```
deleteProp spriteAttributes, #color
```

The result is the list [#height:100, #width: 200, #ink: 15].

{button See also,AL(`Lingo_deleteProp')}

depth of cast

This is obsolete. Use **depth of member** instead.

depth of member

Syntax: `the depth of member` *whichCastmember*

This cast member property gives the color depth of the bitmap cast member specified by *whichCastmember*. Black and white is 1-bit color depth; 256 colors is 8-bit color depth; thousands of colors is 16-bit color depth; and millions of colors is 32-bit color depth.

This property can be tested but not set.

Example:

This statement determines the color depth of the cast member Shrine:

```
put the depth of member "Shrine"
```

deskTopRectList

Syntax: `the deskTopRectList`

This system property indicates the size of the computer's monitors and their position in the desktop. It is useful for checking whether objects such as windows, sprites, and pop-ups appear entirely on one screen.

The result is a list of standard rect coordinates, where each rect 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 rect coordinates is the size of the first monitor. If a second monitor is present, a second set of coordinates show where the corners of the second monitor are relative to the first monitor.

This property can be tested but not set.

Example 1:

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 x 768 pixels and the second monitor is 640 x 480 pixels.

Example 2:

This handler tells how many monitors are in the current system:

```
on countMonitors

   return count( deskTopRectList )

end
```

digitalVideo

See : center of member, controller of member, crop of member, directToStage of member, duration of member, frameRate of member, loop of member, pausedAtStart of member, preload of member, sound of member, trackStartTime(member), trackStopTime(member), video of member, volume of member cast member properties; movieRate of sprite and movieTime of sprite properties

digitalVideoTimeScale

Syntax: `the digitalVideoTimeScale`

This system property determines the time scale the system uses to track digital video cast members.

The time scale is a measurement in units per second. For example, if `the digitalVideoTimeScale` is set to:

- **100** - The time scale is 1/100th of a second (and the movie is tracked in 100 units per second);
- **500** - The time scale is 1/500th 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.

Setting `the digitalVideoTimeScale` lets you precisely access tracks by making sure 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 the system uses to measure digital video to 600 units per second:

```
set the digitalVideoTimeScale to 600
```

digitalVideoType of member

Syntax: `the digitalVideoType of member whichCastmember`

This digital video cast member property indicates the format of the specified digital video. This property can be tested but not set. Possible values are `#quickTime` or `#videoForWindows`.

For more information about controlling digital video, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

The following statement tests whether the cast member Today's Events is a QuickTime or AVI digital video and displays the result in the Message window:

```
put the digitalVideoType of member "Today's Events"
```

directToStage of member

Syntax: `the directToStage of member` *castName*

This property determines whether a digital video cast member plays in front of all other layers on the Stage.

- When this property is set to 1, a digital video plays in front of all other layers.
- When this property is set to 0, a digital video can appear in any layer of the Stage's animation planes. (For QuickTime digital video in Windows, the `directToStage of member` property is always TRUE. Setting the `directToStage of member` to FALSE has no effect in Windows.)

No other cast member can appear in front of a `directToStage` digital video. Also, ink effects do not affect the appearance of a `directToStage` digital video.

Using this property may improve the playback performance of a digital video movie cast member.

When a digital video's `directToStage` property is TRUE, Director writes the digital video directly to the screen without first being composited 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 jump to another similar screen and the video doesn't completely disappear.)

Example:

This statement makes the QuickTime movie "The Residents" always play in the top layer of the Stage:

```
set the directToStage of member "The Residents" to 1
```

do

Syntax: `do` *stringExpression*

This 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.

Example:

This command performs the statement contained within quotation marks:

```
do "beep 2"

do getAt(commandList, 3)
```

dontPassEvent

This command is obsolete. Use **stopEvent** instead.

doubleClick

Syntax: `the doubleClick`

This function lets Director treat two mouse clicks within the time set for a double-click as a double-click rather than two single clicks.

If Lingo doesn't call this function, Director treats two such clicks as two single clicks. However, when Lingo calls this function, it tests whether two clicks have occurred within the time set for a double click:

- If the last two mouse clicks occurred within the time span for a double-click, the `doubleClick` function returns TRUE.
- If the last two mouse clicks didn't occur within the time span for a double-click, the `doubleClick` function returns FALSE.

For more information about determining what the user does with the mouse, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example 1:

This statement sends 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 2:

The following handler tests whether a double-click occurs. 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 the 20 ticks, the `doubleClickAction` handler runs. If a second click doesn't occur, 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
```

{button See also,AL(`Lingo_doubleClick')}

downloadNetThing

Syntax: `downloadNetThing` *URL, localFile*

This command downloads a file from the internet to a local disk so it can be used later without a download delay.

- *URL* is 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, a graphic.
- *localFile* is the pathname and file name for the file on the local disk.

The current movie continues playing while `downloadNetThing` loads the file to a local disk. Use `netDone` to find out whether downloading is finished.

Director movies in authoring mode and projectors support the `downLoadNetThing` command, but this command isn't available in the Shockwave player. This protects users from having files copied from the internet to their disk unintentionally.

Although many network operations can be active at a 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.

Example:

These statements download an external cast from a URL to the folder that Director is in, and then makes that file the external cast named Cast of Thousands:

```
downLoadNetThing("http://www.cbDeMille.com/Thousands.cst",¬
the applicationPath&"Thousands.Cst")
set the fileName of castLib "Cast of Thousands" = ¬
the applicationPath&"Thousands.Cst"
```

{button See also,AL(`Lingo_downLoadNetThing')}

drawRect of window

Syntax: `the drawRect of window` *windowName*

This window property identifies the rectangular coordinates of the section of the movie that appears in the movie's window. The coordinates are given as a rect, with entries in the order left, top, right, and bottom.

This can be useful for scaling or panning movies. However, text in strings doesn't rescale when this property is changed. Rescaling bitmaps can also impact performance.

The `drawRect of window` property can be tested and set.

Example 1:

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 2:

This statement sets the rect of the movie to the values of the rect `movieRectangle.` The portion of the movie within the rect is the part of the movie that appears in the window:

```
set the drawRect of window "Control Panel" ¬

   to movieRectangle
```

{button See also,AL(`Lingo_drawRect_of_window')}

dropShadow of member

Syntax: `the dropShadow of member` *whichCastMember*

This field cast member property determines the size of the drop shadow for text in a field cast member. Possible values are a range of pixels.

For more information about determining what the user does with the mouse, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example:

This statement sets the drop shadow of the field cast member Comment to five pixels:

```
set the dropShadow of member "Comment" to 5
```

duplicate(list)

Syntax: `duplicate(oldList)`

or

`set <x> = duplicate(oldList)`

This function returns a copy of a list. It is useful for saving the current content of a list for later use. Nested lists (list items that are themselves lists) are copied as lists, with all their content duplicated.

When you assign a list to a variable, the variable contains a reference to the list, not the list itself. You can make an independent copy of the list by using the following structure:

`set` *newList* = `value(string(`*oldList*`))`

For more information about lists, see Chapter 10, "Lists," in *Learning Lingo*.

Example:

This statement makes a copy of the list `CustomersToday` and assigns it to the variable `CustomerRecord`:

```
put duplicate(CustomersToday) into CustomerRecord
```

duplicate cast

This is obsolete. Use **duplicate member** instead.

duplicate member

Syntax: `duplicate member` *original* `[,` *new* `]`

This 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 because it essentially creates another cast member in memory. This could result in memory problems.

Example 1:

This statement makes a copy of cast member Desk and places it in the first empty Cast window position:

```
duplicate member "Desk"
```

Example 2:

This statement makes a copy of cast member Desk and places it in Cast window position 125:

```
duplicate member "Desk", member 125
```

duplicateFrame

Syntax: `duplicateFrame`

This 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 during Score generation only.

The `duplicateFrame` command performs the same function as the `insertFrame` command.

For more information about generating Score from Lingo, see Chapter 13, "Authoring from Lingo," in *Learning Lingo*.

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
 set the memberNum of sprite 20 to ¬
      the number of member "Ball" of castLib "Toys"
   repeat with i = 0 to numberOfFrames
      duplicateFrame
 end repeat
 endRecording
end
```

duration of cast

This is obsolete. Use **duration of member** instead.

duration of member

Syntax: `the duration of member` *whichCastmember*

This property determines the duration of the specified Shockwave Audio (SWA) cast member.

- When *whichCastmember* is a streaming sound file, this is the duration of the sound. The duration property returns 0 until streaming has started. Setting `the 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 but not set.

Example 1:

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 the state of member "Louie Prima" = 2 then
    put the duration of member "Louie Prima"¬
    into member "Duration Displayer"
  end if
end
```

Example 2:

This statement sets the duration of the transition cast member Fog to 1 second:

```
set the duration of member "Fog" = 60
```

{button See also,AL(`Lingo_duration_of_member')}

editable of member

Syntax: `the editable of member` *whichCastmember*

This field cast member property determines whether the specified field cast member is editable on Stage.

- When `the editable of member` is TRUE, the specified field cast member is editable.
- When `the editable of member` is FALSE, the specified field cast member isn't editable.

Example:

This statement makes the field cast member Answer editable:

```
set the editable of member "Answer" = TRUE
```

editable of sprite

Syntax: `the editable of sprite` *whichSprite*

This sprite property indicates whether a field sprite is editable.

- When the field can be edited by the user, the `editable of sprite` property is TRUE.
- When the field cannot be edited by the user, the `editable of sprite` property is FALSE.

For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet.

Use `editable of sprite` property to change whether a field can be edited as the movie plays. This lets you turn editable on and off, depending on current conditions in the movie.

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.

The `editable of sprite` property can be tested and set.

For more information about handling several editable fields in a movie, see *Learning Lingo*.

Example 1:

This handler first makes the sprite channel a puppet and then makes the field sprite editable:

```
on myNotes
  puppetSprite 5, TRUE
  set the editable of sprite 5 to TRUE
end
```

Example 2:

This statement checks whether a field sprite is editable and displays a message if it is:

```
if the editable of sprite 13 = TRUE ¬
then set the text of member "Notice"¬
to "Please enter your answer below."
```

EMPTY

Syntax: `EMPTY`

This 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`:

```
set the text of member "Notice" to EMPTY
```

emulateMultiButtonMouse

Syntax: `the emulateMultiButtonMouse`

This system property determines whether a movie interprets clicking the mouse button with the Control key pressed on the Macintosh the same as clicking the right mouse button in Windows. Because the Macintosh has no right mouse button, Lingo that responds to right mouse button clicks has no direct Macintosh equivalent.

- When `emulateMultiButtonMouse` is TRUE, the movie treats clicking the mouse button while the Control key is pressed on the Macintosh the same as clicking the right mouse button on a Windows computer.
- When `the emulateMultiButtonMouse` is FALSE, the movie doesn't treat clicking the mouse button while the Control key is pressed on the Macintosh the same as clicking the right mouse button on a Windows computer.

Setting this property to TRUE lets you provide consistent mouse button responses for cross-platform movies.

For more information about responding to the keyboard and mouse, see Chapter 7, "Working with User Input," in *Learning Lingo*.

Example:

The following statement checks whether the movie is playing on a Windows computer and sets the `emulateMultiButtonMouse` property to TRUE if it is:

```
if the machineType = 256 then set ¬
the emulateMultiButtonMouse to TRUE
```

{button See also,AL(`Lingo_emulateMultiButtonMouse')}

enabled of member

Syntax: `the enabled of member` *whichMember*

This property determines the default state for buttons created with the button editor.

- Disable the button by setting `enabled of member` to zero or FALSE.
- Enable the button by setting `enabled of member` to non-zero or TRUE.

When getting this property, a disabled button returns FALSE and an enabled button returns TRUE.

Example:

This statement causes all subsequent sprites created from this cast member to be initially disabled.

```
set the enabled of member 2 to FALSE
```

enabled of menuItem

Syntax: `the enabled of menuItem` *whichItem* `of menu` *whichMenu*

This menu item property determines whether the menu item specified by *whichItem* is displayed in plain text or dimmed, and whether it is selectable. The term *whichMenu* specifies the menu that contains the menu item.

- If `the enabled of menuItem` is TRUE, the menu item appears in plain text and is selectable.
- If `the enabled of menuItem` is FALSE, the menu item appears dimmed and is not selectable.

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.

Although some Lingo for formatting custom menus works on the Macintosh only, `the enabled of menuItem` works on both Windows and Macintosh computers.

The enabled property can be tested and set. The default value is TRUE.

For more information about custom menus, see Chapter 9, "Creating User Interfaces," in *Learning Lingo*.

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
```

{button See also,AL(`Lingo_enabled_of_menuItem')}

enabled of sprite

Syntax: `the enabled of sprite` *whichSprite*

This button sprite property enables or disables a button sprite.

- Disable the button by setting `enabled of sprite` to zero or FALSE.
- Enable the button by setting `enabled of sprite` to non-zero or TRUE.

Getting this property returns TRUE for enabled sprites and FALSE for disabled sprites.

Note: This property does not affect existing sprites.

Example:

This statement disables the button in sprite 4:

```
set the enabled of sprite 4 to FALSE
```

end

This keyword marks the end of handlers, methods, and multiline control structures.

{button See also,AL(`Lingo_end')}

end case

Syntax: `end case`

This 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
```

{button See also,AL(`Lingo_end_case')}

end repeat

See repeat while
repeat with
repeat with...in list
repeat with...down to

endRecording

Syntax: `endRecording`

This 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
    set the memberNum of sprite 20 to ¬
    the number of member "Ball"
    set horizontal = 0
    set vertical = 300
    repeat with i = 0 to numberOfFrames
    set the locH of sprite 20 to horizontal
    set the locV of sprite 20 to vertical
    set horizontal = horizontal + 3
    set vertical = vertical + 2
    updateFrame
    end repeat
  endRecording
end
```

{button See also,AL(`Lingo_endRecording')}

ENTER

Syntax: `ENTER`

This character constant represents the Enter key.

- This key is marked as "enter" on the Macintosh keyboard.
- Although PC keyboards also label the key that enters a carriage return as Enter, the element `ENTER` only refers to the Enter key on the number pad. In Windows, use `RETURN` to represent the key that enters a carriage return.

Example:

This statement checks whether the Enter key is pressed and sends the playback head to the frame `addSum` if it is:

```
on keyDown

   if the key = ENTER then go to frame "addSum"

end
```

enterFrame

See: on enterFrame event handler.

erase cast

This is obsolete. Use **erase member** instead.

erase member

Syntax: `erase member` *whichCastmember*

This command deletes the specified cast member and leaves its slot in the Cast window empty.

Example 1:

This statement deletes the cast member named `Gear` in the cast Hardware:

```
erase member "Gear" of castLib "Hardware"
```

Example 2:

This handler deletes cast members `start` through `finish`:

```
on deleteMember start, finish
   repeat with i = start to finish
      erase member i
   end repeat
end on deleteMember
```

{button See also,AL(`Lingo_erase_member')}

EvalScript

See: on EvalScript event handler

exit

Syntax: `exit`

This keyword instructs Lingo to leave a handler and return to the place from where the handler was called. When 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 exits if it is:

```
on setColors

   if the colorDepth = 1 then exit

   set the foreColor of sprite 1 to 35

end setColors
```

{button See also,AL(`Lingo_exit')}

exit repeat

Syntax: `exit repeat`

This keyword instructs Lingo to leave a repeat loop and go to the statement following the `end repeat` statement, but 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 the number of chars ¬
      in testString
      if "aeiou" contains ¬
      char I of testString then exit repeat
   end repeat
   return i
end findVowel
```

{button See also,AL(`Lingo_exit_repeat')}

exitLock

Syntax: `the exitLock`

This property determines whether the user can quit to the Desktop from projectors.

- When `the exitLock` is FALSE, the user can quit to the desktop by pressing Control+period, Control+Q or Control+W.
- When `the exitLock` is TRUE, the user cannot quit to the desktop by pressing Control+period, Control+Q or Control+w.

The `exitLock` property can be tested and set. The default value is FALSE.

Example 1:

This statement sets the `exitLock` property to TRUE:

```
set the exitLock to TRUE
```

Example 2:

This handler checks whether Control+period , Control+Q, or Control+W was pressed and whether the `exitLock` property is set so that the user cannot exit to the Desktop. When this is the case, the playback head goes to the frame "quit sequence," which can provide an alternative way to exit the movie:

```
on checkExit

   if the commandDown and ¬

   (the key = "." or the key = "q") and ¬

   the exitLock = TRUE then go to frame "quit sequence"

end checkExit
```

exp

Syntax: `exp(`*integerOrFloat)*

This 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 exp(5)
-- 148.4132
```

externalEvent

Syntax: `externalEvent` *"string"*

This command sends the browser a string that the browser can interpret as a scripting language instruction. This allows a movie playing on the internet to communicate with the browser in which it is embedded.

Supported scripting languages vary for different browsers. To be used by a specific browser, the string sent by `externalEvent` must be in a language supported by the browser.

The `externalEvent` command does not produce a return value. There is no immediate way to determine if the browser handled the event or ignored it. Use `on EvalScript` within the browser to return a message to the movie.

Example 1:

These statements use `externalEvent` in the LiveConnect scripting environment, which is supported by Netscape 3.x.

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 quotes.

Statements within HTML:

```
function MyFunction(parm1, parm2) {

  //script here

}
```

Statements within a script in the movie :

```
externalEvent ("MyFunction('parm1','parm2')")
```

Example 2:

These statements use `externalEvent` in the ActiveX scripting environment.

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, rather than defining a function, you can define a script for the event:

```
<SCRIPT FOR="NameOfShockwaveInstance"

EVENT="externalEvent(aParam)" LANGUAGE="VBScript">

  'script here

</SCRIPT>
```

Statements within a script in the movie:

Within the movie, include the function and any parameters as part of the string for `externalEvent`:

```
externalEvent ("MyFunction ('parm1','parm2')")
```

externalParamCount

Syntax: `externalParamCount()`

This function returns the number of parameters an HTML EMBED or OBJECT tag is passing to a Shockwave movie.

This function is only valid for Shockwave movies that are running in a browser. It doesn't work for movies during authoring or 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 they are:

```
if externalParamCount() > 0 then

  -- perform some action

end if
```

externalParamName

Syntax: `externalParamName(`*`n`*`)`

This function returns the name of a specific parameter in the list of external parameters from an HTML EMBED or OBJECT tag.

- If `n` is an integer, `externalParamName` returns the nth 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`.

This function is only valid for Shockwave movies that are running in a browser. It cannot be used with Director movies or projectors.

Example:

This statement puts the value of a given external parameter into the variable `myVariable`:

```
if externalParamName(swURL) then
   set myVariable to externalParamValue(swURL)
end if
```

externalParamValue

Syntax: `externalParamValue(n)`

This function returns a specific value from the external parameter list in an HTML EMBED or OBJECT tag.

- If `n` is an integer, `externalParamValue` returns the nth 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 is only valid for Shockwave movies that are running in a browser. It can't be used with movies running inside Director or projectors.

Example:

This statement puts the value of an external parameter into the variable `myVariable`:

```
if externalParamName(swURL) then
   set myVariable to externalParamValue(swURL)
end if
```

factory

This Lingo is now outdated.

FALSE

Syntax: `FALSE`

This logical 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.

Example:

This statement turns off the `soundEnabled` property by setting it to FALSE:

```
set the soundEnabled to FALSE
```

{button See also,AL(`Lingo_FALSE')}

field

Syntax: `field` *whichField*

This 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 can be read from or put into the field. You can also use chunk expressions with fields.

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 1:

This statement puts the characters 5 through 10 of the field name entry into the variable `myKeyword`:

```
put char 5 to 10 of field "entry" into myKeyword
```

Example 2:

This statement checks whether the user entered the word *desk* and goes to the frame *deskBid* if he or she did:

```
if field "bid" contains "desk" then go to "deskBid"
```

{button See also,AL(`Lingo_field')}

fileName of cast

This is obsolete. Use **fileName of member** instead.

fileName of castLib

Syntax: `the fileName of castLib` *whichCast*

This property gives the file name of the specified cast.

- For an external cast, `the fileName of castLib` gives the cast's full pathname and file name.
- For an internal cast, `the fileName of castLib` depends on which internal cast is specified. For the first internal cast library, `the fileName of castLib` is the name of the movie. For remaining internal casts, `the fileName of castLib` is an empty string.

The `fileName of castLib` property accepts URLs as references. However, to use a cast from the internet, use the `downloadNetThing` command to download the cast's file to a local disk first, and then set `the fileName of castLib` to the file on the disk. This minimizes problems with waiting for the cast to download.

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 only be tested for internal casts.

Example 1:

This statement displays the pathname and file name of the external cast Buttons in the Message window:

```
put the fileName of castLib "Buttons"
```

Example 2:

This statement sets the file name of the external cast Buttons to Content.cst:

```
set the fileName of castLib "Buttons" to ¬
the pathName&"Content.cst"
```

The movie then uses the external cast file Content.cst as the cast Buttons.

Example 3:

These statements download an external cast from a URL to the same folder that Director is in, and then makes that file the external cast named Cast of Thousands:

```
downLoadNetThing("http://www.cbDeMille.com/Thousands.cst",¬

the applicationPath&"Thousands.Cst")

set the fileName of castLib "Cast of Thousands" = ¬

the applicationPath&"Thousands.Cst"
```

{button See also,AL(`Lingo_fileName_of_castLib')}

fileName of member

Syntax: `the fileName of member` *whichCastmember*

This cast member property refers to the name of the file assigned to the linked cast member specified by *whichCast member*. This is useful for switching which external linked file is assigned to a cast member while the 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.

The `fileName of member` property accepts URLs as a reference. However, to use a file from a URL, use the `downloadNetThing` command to download the file to a local disk first, and then set `the fileName of member` to the file on the local disk. This minimizes problems with waiting for the file to download.

The `fileName of member` property can be tested and set. After the file name is set, Director uses that file the next time the cast member is used.

For more information about using `the fileName of member` to assign content to cast members, see Chapter 13, "Authoring from Lingo," in *Learning Lingo*.

Example:

This statement makes the QuickTime movie "ChairAnimation" the linked file assigned to cast member 40:

```
set the fileName of member 40 = "ChairAnimation"
```

These statements download an external file from a URL to the same folder that Director is in, and then makes that file the media for the sound cast member Norma Desmond Speaks:

```
downLoadNetThing("http://www.cbDeMille.com/Talkies.AIF",¬
the applicationPath&"Talkies.AIF")
set the fileName of member "Norma Desmond Speaks" = ¬
the applicationPath&"Talkies.AIF"
```

{button See also,AL(`fileName_of_member')}

fileName of window

Syntax: `the fileName of window` *whichWindow*

This 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.

You assign a movie to a window by setting `the fileName of window` for the window to the movie's file name. This is required before you can play the movie in the window.

The `fileName of window` property accepts URLs as a reference. However, to use a movie file from a URL, use the `downloadNetThing` command to download the movie file to a local disk first, and then set `the fileName of window` to the file on the local disk. This minimizes problems with waiting for the file to download.

The `fileName of window` property can be tested and set.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

Example 1:

This statement assigns the file named Control Panel to the window named Tool Box:

```
set the fileName of window "Tool Box" = ¬
   "Control Panel"
```

Example 2:

This statement displays the file name of the file assigned to the window named Navigator:

```
put the fileName of window "Navigator"
```

Example 3:

These statements download a movie file from a URL to the same folder that Director is in, and then assign that file to the window My Close Up:

```
downLoadNetThing("http://www.cbDeMille.com/Finale.DIR",¬
the applicationPath&"Finale.DIR")
set the fileName of window "My Close Up" = ¬
the applicationPath&"Finale.DIR"
```

filled of member

Syntax: `the filled of member` *whichCastmember*

This shape cast member property indicates whether the specified cast member is filled with a pattern.

- When `the filled of member` is TRUE, the shape is filled with a pattern.
- When `the filled of member` is FALSE, the shape isn't filled with a pattern.

Example:

The following statements make the shape cast member Target Area a filled shape and assigns it the pattern numbered 0, which is a solid color:

```
set the filled of member "Target Area" = TRUE

set the pattern of member "Target Area" = 0
```

{button See also,AL(`Lingo_filled_of_member')}

findEmpty

Syntax: `findEmpty(member` *whichCastmember)*

This function gives the next empty cast member position or the position after the cast member specified by *whichCastmember*. This function works on the current cast only.

Example:

This statement finds the first empty cast member on or after cast member 100.

```
put findEmpty(member 100)
```

findPos

Syntax: `findPos(`*list*`,` *property)*

This command identifies which position the property specified by *property* holds in the property list specified by *list.*

The `findPos` command works with property lists only. If you try to use `findPos` with linear lists, `findPos` returns a bogus number if the value of *prop* is a number and a script error if the value of *prop* is a string.

The `findPos` command performs the same function as the `findPosNear` command, except that the result of the `findPos` command 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]:

```
findPos(Answers, #c)
```

The result is 3, because c is the third property in the list.

{button See also,AL(`Lingo_findPos')}

findPosNear

Syntax: `findPosNear(`*sortedList , valueOrProperty)*

This command identifies which position the item specified by *valueOrProperty* holds in the specified sorted list.

The `findPosNear` command works with sorted lists only.

- For sorted linear lists, replace *valueOrProperty* with a value.
- For sorted property lists, replace *valueOrProperty* with a property.

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 that has the most similar alphanumeric name. This would be useful in finding the closest name 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]:

```
findPosNear(Answers, #Ni)
```

The result is 1, because Ni is closest to Nile, the first property in the list.

{button See also,AL(`Lingo_findPosNear')}

finishIdleLoad

Syntax: `finishIdleLoad` *loadTag*

This command completes 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
```

fixStageSize

Syntax: `the fixStageSize`

This property determines whether the Stage size remains the same when you load a new movie, regardless of the Stage size saved with that movie. When the `fixStageSize` property is TRUE, the Stage size remains the same when you load a new movie.

The `fixStageSize` property cannot change the Stage size for a movie that is currently playing. This property is primarily used for movies played back with the player.

The `fixStageSize` property can be tested and set. The default value is TRUE.

Example:

This statement determines if the `fixStageSize` property is turned on and sends the playback head to a specified frame if it is.

```
if the fixStageSize = FALSE then ¬
   go to frame "proper size"
```

This statement sets the `fixStageSize` property to the opposite of its current setting:

```
set the fixStageSize to (not the fixStageSize)
```

{button See also,AL(`Lingo_fixStageSize')}

float

Syntax: `float (`*expression)*

This function converts an expression to a floating-point number. The number of digits that follow the decimal point is set using the `floatPrecision` property.

Example:

This statement converts the integer 1 to floating-point 1.

```
put float(1)

-- 1.0
```

{button See also,AL(`Lingo_float')}

floatP

Syntax: `floatP(`*expression)*

This function indicates whether the value specified by *expression* is a floating-point number.

- The `floatP` function is TRUE (1) if *expression* is a floating-point number.
- The `floatP` function is FALSE (0) if *expression* is not a floating-point number.

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 it is TRUE:

```
put floatP(3.0)

-- 1
```

This statement tests whether 3 is a floating-point number. The Message window displays the number 0, indicating that it is FALSE:

```
put floatP(3)

-- 0
```

{button See also,AL(`Lingo_floatP')}

floatPrecision

Syntax: `the floatPrecision`

This system 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 is 15 significant digits.

The `floatPrecision` property determines only the number of digits used to display floating-point numbers. The number of digits used to perform calculations doesn't change.

- If `the floatPrecision` is a number from 1 to 15, floating-point numbers display that number of digits after the decimal point. Trailing zeros remain.
- If `the floatPrecision` is zero, floating-point numbers are rounded to the nearest integer. No decimal points appear.
- If `the floatPrecision` is a negative number, floating-point numbers are rounded to the absolute value for the number of decimal places. Trailing zeros are dropped.

The `floatPrecision` property can be tested and set. The default value is 4.

Example 1:

This statement rounds off the square root of 3.0 to three decimal places:

```
set the floatPrecision to 3

set x = sqrt(3.0)

put x

-- 1.732
```

Example 2:

This statement rounds off the square root of 3.0 to eight decimal places:

```
set the floatPrecision to 8

put x

-- 1.73205081
```

font of member

Syntax: `the font of member` *whichCastmember*

This field property determines the typeface of the font used to display the specified field cast member. The parameter *whichCastmember* can be either a cast member name or number.

The `font of member` field property can be set, affecting every line in the field. When tested, it returns the height of the first line of the field.

The field cast member must contain characters, if only a space, to use the `font of member` property. It has no effect on a cast member that contains no characters.

Example:

This statement sets the variable named `oldFont` to the current `font of member` setting for the field cast member Rokujo Speaks:

```
set oldFont = the font of member "Rokujo Speaks"
```

{button See also,AL(`Lingo_font_of_member')}

fontSize of member

Syntax: `the fontSize of member` *whichCastmember*

This field property determines the size of the font used to display the specified field cast member. The parameter *whichCastmember* can be either a cast member name or number.

The `fontSize` field property can be tested and set.

Example:

This statement sets the variable named `oldSize` to the current `fontSize of member` setting for the field cast member Rokujo Speaks:

```
set oldSize =  the fontSize of member "Rokujo Speaks"
```

This property requires that the field cast member already contain characters, if only a space. It will not affect a cast member that contains no characters.

{button See also,AL(`Lingo_fontSize_of_member')}

fontStyle of member

Syntax: `the fontStyle of member` *whichCastmember*

This field property determines the styles applied to the font used to display the specified field cast member.

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 `extend`. On the Macintosh, `condense` is also an available style.

Use the word `plain` to remove all of the styles that are currently applied. The parameter *whichCastmember* can be either a cast member name or number.

The field cast member must contain characters, if only a space, to use the `fontStyle of member` property. It has no effect on a cast member that contains no characters.

The `fontStyle of member` field property can be tested and set.

Example 1:

This statement sets the variable named `oldStyle` to the current `fontStyle of member` setting for the field cast member Rokujo Speaks:

```
set oldStyle = the fontStyle of member "Rokujo"
```

Example 2:

This statement sets the `fontStyle of member` setting for the field cast member Rokujo Speaks to bold italic:

```
set the fontStyle of member "Poem" to "bold, italic"
```

{button See also,AL(`Lingo_fontStyle_of_member')}

foreColor of cast

This is obsolete. Use **foreColor of member** instead.

foreColor of member

Syntax: `set the foreColor of member` *castName* `to` *colorNumber*

This cast member property sets the foreground color of a field cast member.

Example:

This statement changes the color of the field in cast member 1 to the color in palette entry 250:

```
set the foreColor of member 1 to 250
```

foreColor of sprite

Syntax: `the foreColor of sprite` *whichSprite*

This 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.

The foreground color applies only to 1-bit bitmap and shape cast members. It does not affect the display of a field or button cast member. An 8-bit, 16-bit, or 24-bit bitmap is affected, but generally not in a useful way.

The value of a sprite's background color ranges from 0 to 255 for 8-bit color, or 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.)

Changing a sprite's foreground color in an `on mouseDown` handler is a useful way to indicate when a sprite is clicked.

The `foreColor of sprite` property can be tested and set.

Example 1:

The following statement sets the variable `oldColor` to the foreground color of sprite 5:

```
set oldColor to the foreColor of sprite 5
```

Example 2:

The following statement makes 36 the number for the foreground color of a random sprite from sprite 11 to sprite 13.

```
set the backColor of sprite (10 + random(3)) to 36
```

forget window

Syntax: `forget window` *whichWindow*

This command instructs Lingo to close and delete the window specified by *whichWindow* when the window is no longer in use and no other variables refer to it.

When a `forget window` command is given, the window and the MIAW disappear without calling the `on stopMovie`, `on closeWindow`, or `on deactivateWindow` handlers.

If there are multiple 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:

```
forget window "Control Panel"
```

{button See also,AL(`Lingo_forget_window')}

frame

Syntax: `the frame`

This function returns the number of the current frame of the current movie.

Example:

This statement sends the playback head to the frame before the current frame:

```
go to (the frame - 1)
```

{button See also,AL(`Lingo_frame')}

frameLabel

Syntax: `the frameLabel`

This 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.

The `frameLabel` property can be tested at any time. It can also be set during a Score generation session.

Example:

This statement checks the label of the current frame. In this case, the current `frameLabel` is Start:

```
put the frameLabel

-- "Start"
```

{button See also,AL(`Lingo_frameLabel')}

framePalette

Syntax: `the framePalette`

This frame property identifies the cast member number of the palette used in the current frame.

The palette doesn't have to be set in the current frame. The frame's palette is whichever palette is currently in effect.

The `framePalette` property can be tested. It can also be set during a Score generation session.

Example 1:

This statement checks the palette used in the current frame. In this case, the palette is cast member 45:

```
put the framePalette
-- 45
```

Example 2:

This statement makes cast member 45, which is a palette cast member, the palette for the current frame:

```
set the framePalette to 45
```

{button See also,AL(`Lingo_framePalette')}

frameRate of cast

This is obsolete. Use **frameRate of member** instead.

frameRate of member

Syntax: `the frameRate of member` *QTcastmember*

This digital video cast member property specifies the playback frame rate for the specified digital video cast member. The possible values for `the frameRate of member` correspond to the radio buttons for selecting digital video playback options.

- When `the frameRate of member` is between 0 and 255, the digital video movie plays every frame at that frame rate. The `frameRate of member` property cannot be greater than 255.
- When `the frameRate of member` is set to -1, the digital video movie plays every frame at its normal rate.
- When `the frameRate of member` is set to -2, the digital video movie plays every frame as fast as possible.

Example 1:

This statement sets the frame rate of the QuickTime digital video cast member Rotating Chair to 30 frames per second:

```
set the frameRate of member "Rotating Chair" to 30
```

Example 2:

This statement instructs the QuickTime digital video cast member Rotating Chair to play every frame as fast as possible:

```
set the frameRate of member "Rotating Chair" to -2
```

{button See also,AL(`Lingo_frameRate_of_member')}

frameReady

Syntax: `frameReady(`*frameN*`)`

or

`frameReady(`*frameN, frameZ*`)`

or

`frameReady()`

This function determines if all the cast members for *frameN* are downloaded from the internet and available locally. *frameN* is the number of the frame.

- `frameReady (`*frameN*`)` determines whether the cast members for *frameN* are downloaded.
- `frameReady (`*frameN, frameZ*`)` determines whether the cast members for frames *frameN* through *frameZ* are downloaded.
- `frameReady()` determines if the entire movie is downloaded.

The function can be used with Director movies, projectors, and Shockwave movies.

Note: This property is only useful when streaming a movie or cast. Streaming a movie is activated by setting the **Movie:Playback** properties in the Modify menu to Use Media As Available or Show Placeholders.

This function can be tested but not set.

For a demonstration of the `frameReady` function, see the sample movie **Streaming Shockwave**

Example:

This statement determines if 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 cast members are
  locally available

    go to frame 20

  else

    -- resume animating loop while background streaming

    got to frame 1

  end if

end
```

mediaReady of member

frameScript

Syntax: `the frameScript`

This frame property identifies the 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. (This property could not be set in earlier versions of Director.)

Example 1:

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 2:

This statement makes the script cast member "Button responses" the frame script for the current frame:

```
set the frameScript to member "Button responses"
```

frameSound1

Syntax: `the frameSound1`

This frame property determines the number of the cast member assigned to the first sound channel in the current frame. 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:

```
set the frameSound1 to member "Jazz"
```

frameSound2

Syntax: `the frameSound2`

This frame property determines the number of the cast member assigned to the second sound channel for the current frame. 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:

```
set the frameSound2 to member "Jazz"
```

framesToHMS

Syntax: `framesToHMS(`*frames , tempo , dropFrame , fractionalSeconds)*

This function converts the specified number of frames to their equivalent length in hours, minutes, and seconds. This is useful for predicting the actual playtime of a movie or controlling a video playback device.

- The integer expression *frames* specifies the number of frames.
- The integer expression *tempo* specifies the tempo in frames per second.
- The *dropFrame* argument is a logical expression. Normally, this is FALSE. This argument is meaningful only if FPS is set to 30 frames per second. (Drop frame is a method of compensating for the color NTSC frame rate, which is not exactly 30 frames per second.)
- The *fractionalSeconds* argument determines what happens to residual frames. When TRUE replaces *fractionalSeconds*, the residual frames are converted to the nearest hundredth of a second. When FALSE replaces *fractionalSeconds*, the residual frames are returned as an integer number of frames.

The resulting string uses the form: "sHH:MM:SS.FFD", where:

s-"-" if the time is less than zero, or space if the time is greater than or equal to zero

HH-hours

MM-minutes

SS-seconds

FF-fraction of a second if *fractionalSeconds* is TRUE or frames if *fractionalSeconds* is FALSE

D-"d" if *dropFrame* is TRUE or 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 "
```

{button See also,AL(`Lingo_framesToHMS')}

frameTempo

Syntax: `the frameTempo`

This frame property indicates the tempo assigned to the current frame.

The `frameTempo` property can be tested. It can also be set during a Score recording session. (This property could not be set in earlier versions of Director).

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
```

{button See also,AL(`Lingo_frameTempo')}

frameTransition

Syntax: `the frameTransition`

This frame property gives the number of the transition cast member assigned to the current frame. During a Score recording session, you can also set this property as a way to specify transitions.

Example:

When used in a Score recording session, this statement makes the transition cast member Fog the transition for the frame that Lingo is currently recording:

```
set the frameTransition to member "Fog"
```

freeBlock

Syntax: `the freeBlock`

This 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. In order to load a cast member, you need 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!"
```

{button See also,AL(`Lingo_freeBlock')}

freeBytes

Syntax: `the freeBytes`

This 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`, because it reports all free memory, not just contiguous memory.

On the Macintosh, if you turn on Use System Temporary Memory in Director's General Preferences or a projector's Options dialog box, the `freeBytes` function returns all the free memory that is available to the application. This is the same as 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"
```

{button See also,AL(`Lingo_freeBytes')}

frontWindow

Syntax: `the frontWindow`

This system property indicates which movie in a window is currently frontmost on the Stage. When the Stage is frontmost, `the frontWindow` is `the Stage`. When a media editor or floating palette is frontmost, `the frontWindow` is `VOID`.

This property can be tested but not set.

Example:

This statement determines whether the Stage is currently the frontmost window and, if it is, brings the window Try This to the front:

```
if the frontWindow = the stage then ¬
moveToFront window "Try This"
```

fullColorPermit

Syntax: `the fullColorPermit`

This property determines whether the computer's offscreen buffer is set to the full color possible for the monitor.

- When `the fullColorPermit` is TRUE, the offscreen buffer is a 32-bit buffer on a 32-bit monitor, and a 16-bit buffer on a 16 bit-monitor. This is the default.
- When `the fullColorPermit` is FALSE, the maximum depth of the offscreen animation buffer is 8 bits, regardless of the monitor's color depth.

Setting this property can be useful for projectors, which require a smaller offscreen buffer when running 8-bit color on a 16-bit or 32-bit monitor.

Example:

This statement sets `the fullColorPermit` to FALSE, which allows a projector to use an 8-bit buffer on a computer with a 16-bit or 32-bit monitor.

```
set the fullColorPermit = FALSE
```

getaProp

Syntax: `getaProp(`*list*`,` *item*`)`

This command identifies the value associated with the item specified by *item* 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. The result is the value at that position.
- When the list is a property list, replace *item* with a property in the list. 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 does the same as the `getAt` command.

Example 1:

This statement identifies the value in the third position of the linear list Answers, which consists of [10, 12, 15, 22]:

```
getaProp(Answers, 3)
```

The result is 15, because 15 is the third value in the list.

Example 2:

This statement identifies the property associated with the value 15 in the property list Answers, which consists of [#a: 10, #b:12, #c:15, #d:22]:

```
getaProp(Answers, #c)
```

The result is 15, which is the value associated with the property.

{button See also,AL(`Lingo_getaProp')}

getAt

Syntax: `getAt(`*list, position)*

This command identifies the item in the position specified by *position* in the specifed list. If the list contains fewer elements than the specified position, an alert appears.

The `getAt` command works with linear and property lists. This command does the same as the `getaProp` command for linear lists.

Example:

This statement causes the Message window to display the third item in the list Answers, which consists of [10, 12, 15, 22]:

```
getAt(Answers, 3)
```

The result is 15.

{button See also,AL(`Lingo_getAt')}

getBehaviorDescription

See: on getBehaviorDescription handler.

getError

Syntax: `getError` *whichCastmember*

This function returns an error number indicating the type of error that occurred with a Shockwave Audio cast member. (The `getErrorString` function returns a message describing the error.) The following are possible values for `getError` and `get ErrorString`:

`getError()` result	**Corresponding value returned by `getErrorSting()`**
0	OK
1	memory
2	network
3	playback device
99	other

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 getError (member "Norma Desmond Speaks") <> 0 then
    put getErrorString (member "Norma Desmond Speaks")
    into member "Display Error Name"
  end if
end
```

{button See also,AL(`Lingo_getError')}

getErrorString

Syntax: `getErrorString` *"whichCastmember"*

This function returns a string that is an error message indicating the type of error that occurred with a Shockwave Audio cast member. (The `getError` function returns the number of the error message.) The following are possible values for `getError` and `get ErrorString`:

`getError()` result	**Corresponding value returned by `getErrorSting()`**
0	OK
1	memory
2	network
3	playback device
99	other

Example:

This handler uses `getError` to determine whether an error involving the Shockwave Audio cast member Norma Desmond Speaks occurred. If an error occurred, the handler uses `getErrorString` to obtain the error message and assigns the message to a field cast member:

```
on exitFrame
  if getError (member "Norma Desmond Speaks") <> 0 then
    put getErrorString (member "Norma Desmond Speaks")
    into member "Display Error Name"
  end if
end
```

{button See also,AL(`Lingo_getErrorString')}

getLast

Syntax: `getLast(`*list)*

This command identifies the last value in the list specified by *list*. The `getLast` command works with linear and property lists.

Example 1:

This statement identifies the last item in the list Answers, which consists of [10, 12, 15, 22]:

```
put getLast(Answers)
```

The result is 22.

Example 2:

This statement identifies the last item in the list Bids, which consists of [#Gee:750, #Kayne:600, #Ohashi:850]:

```
put getLast(Bids)
```

The result is 850.

getLatestNetID

Syntax: `getLastestNetID`

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 net lingo command.

For more information about Shockwave movies and the internet, see Chapter 14, "Shockwave, the Internet, and Lingo," in *Learning Lingo*.

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
```

{button See also,AL(`Lingo_getLatestNetID')}

getNetText

Syntax: `getNetText` *URL*

This function starts the retrieval of text from a file on an ftp or HTTP server.

Use `netDone` to find out when the `getNetText` operation is complete. Use `netTextResult` to return the text retrieved by `getNetText`.

The function works with relative URLs.

For more information about Shockwave movies and the internet, see Chapter 14, "Shockwave, the Internet, and Lingo," in *Learning Lingo*.

Example:

This script retrieves text from the URL http://BigServer.com/sample.txt and assigns it to the field cast member Display Text:

```
on mouseUp
  set netID = getNetText "http://BigServer.com/sample.txt"
end
on idle
  if netDone(netID) then
    put the netTextResult into member "Display Text"
  end if
end
```

{button See also,AL(`Lingo_getNetText')}

getNthFileNameInFolder

Syntax: `getNthFileNameInFolder(`*folderPath, fileNumber)*

This function returns a file name from the directory folder at the specified path and number within the folder. To be found by the `getNthFileNameInFolder` function, Director movies must be set to be visible in the folder structure. However, the `getNthFileNameInFolder` function finds other types of files whether they are visible or invisible. If the function returns an EMPTY string, you have specified a number greater than the number of files in the folder.

To return the name of the current folder, use `-1` as the *fileNumber* parameter.

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 the movie is running on.

- For example, on the Macintosh, use a pathname such as `HardDisk:Director:Movies`. To look for files on the Macintosh desktop, you would use the path `HardDisk:Desktop Folder`.
- To specify a pathname in Windows, use a directory path such as `C:\Director\Movies`.

Example:

The following handler returns a list of file names in the folder at the current path. To call the function, use parentheses, as in `put currentFolder()`:

```
on currentFolder
put [] into fileList
  repeat with i = 1 to the maxInteger
    put getNthFileNameInFolder(the pathName, i) ¬
      into n
    if n = EMPTY then exit repeat
      append(fileList, n)
  end repeat
  return fileList
end currentFolder
```

If Director is running in the current folder, this statement displays the folder's name in the Message window:

```
put getNthFileNameInFolder(the applicationPath,  -1)
```

{button See also,AL(`Lingo_getNthFileNameInFolder')}

getOne

Syntax: `getOne(`*list , value)*

This command identifies the position or property associated with the value specified by *value* in the list specified by *list*.

- When the list is a linear list, the result is the value's position in the list.
- When the list is a property list, the result is the property associated with the value in the list.

For values contained in the list more than once, only the first occurrence is displayed. The `getOne` command gives 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 1:

This statement identifies the position of the value 12 in the linear list Answers, which consists of [10, 12, 15, 22]:

```
getOne(Answers, 12)
```

The result is 2, because 12 is the second value in the list.

Example 2:

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]:

```
getOne(Answers, 12)
```

The result is #b, which is the property associated with the value 12.

{button See also,AL(`Lingo_getOne')}

getPos

Syntax: `getPos(`*list , value)*

This command 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 gives the value 0.

The `getPos` command works with linear and property lists.

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]:

```
getPos(Answers, 12)
```

The result is 2, because 12 is the second value in the list.

{button See also,AL(`Lingo_getPos')}

getPref

Syntax: `getPref(`*`prefName`*`)`

This function retrieves the content of the specified file. It only works in browsers.

When you use this function, replace *prefName* with the name of a file that was previously created by the `setPref` function. If no such file exists, `getPref` returns VOID.

The file name used for *prefName* must be a valid file name only. Do not include a pathname when specifying *prefName*. The only valid file extensions for *prefName* are txt or htm; any other extension is rejected.

For more information about Shockwave movies and the internet, see Chapter 14, "Shockwave, the Internet, and Lingo," in *Learning Lingo*.

Example:

This handler retrieves the content of the file Test and then assigns the file's text to the field Total Score:

```
on mouseUp

   set theText = getPref("Test")

   put theText into member "Total Score"

end
```

{button See also,AL(`Lingo_getPref')}

getProp

Syntax: `getProp(`*list , property)*

This command identifies the value associated with the property specified by *property* in the property list specified by *list*.

The `getProp` command works with property lists only. Using `getProp` with a linear list produces a script error.

The `getProp` command is identical to the `getaProp` command, except that the `getProp` command displays an error message when the specified property is not in the list.

Example:

This statement identifies the value associated with the property `#c` of 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.

{button See also,AL(`Lingo_getProp')}

getPropAt

Syntax: `getPropAt(`*list* , *index)*

This command 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, Director displays an error message.

The `getPropAt` command works with property lists only. Using `getPropAt` with linear lists produces a script error.

Example:

This statement displays the second property in the given list.

```
put getPropAt([#a:10, #b:20],2)

-- #b
```

getPropertyDescriptionList

See: on getPropertyDescriptionList

global

Syntax: `global` *variable1* `[,` *variable2* `]` `[,` *variable3* `]...`

This keyword identifies a variable as a global variable so that it can be shared by other handlers or movies.

Every handler that examines or changes the contents of a global variable must use the `global` keyword to identify the variables as global. Otherwise, the handler treats the variable as a local variable even if it is declared to be global in another handler.

A global variable can be declared in a handler or the Message window. Its value can be used by any other handlers or scripts that declare the variable as global.

Example:

```
global startingPoint

set startingPoint = whichMenu
```

{button See also,AL(`Lingo_global')}

go

Syntax: `go {to} {frame}` *whichFrame*

`go {to} movie` *whichMovie*

`go {to} {frame}` *whichFrame* `of movie` *whichMovie*

This command causes the playback head to jump to the frame specified by *whichFrame* of 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 pathname.)

The phrase "go to the frame" has the playback head loop in the current frame. This is a convenient way to keep the playback head in the same frame, but keep Lingo active. Avoid using "go to the frame" in a frame that has a transition. This slows down the movie and overwhelms the processor as it constantly tries to perform the transition.

It's better to refer to marker labels instead of frame numbers, because editing a movie can cause frame numbers to change. Thus, a command like `go to frame 35` can become incorrect. It's also easier to read your script if you use marker labels.

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. If you want to suspend the handler while playing the movie, use the `play` command.

When you specify a movie to play, you must also specify its path if the movie is in a different folder. There's no need to include the movie's DIR or DXR file extension in the `go to movie` command. In fact, it's often better to omit the suffix during development. If you use DIR files during the main development cycle and then later protect those movies, Lingo will fail when it encounters hard-coded file names with the incorrect suffix. If you omit the file suffix, Lingo looks for a file with either suffix in the specified path.

To go to a movie at a URL, it can be more efficient to 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 loading a movie: `the beepOn`, `the constraint` properties, `the keyDownScript`, `the mouseDownScript`, `the mouseUpScript`; `the cursor of sprite` and `immediate of sprite` properties; `the cursor` and `puppetSprite` commands; and custom menus. However, `the timeoutScript` is not reset when loading a movie.

Example 1:

This statement sends the playback head to the marker named "start":

```
go to "start"
```

Example 2:

This statement sends the playback head to the marker named Memory in the movie named "Noh Tale to Tell":

```
go to frame "Memory" of movie "Noh Tale to Tell"
```

Example 3:

This handler has the movie loop in the current frame. This is useful for making the movie wait in a frame while the movie plays so that it can respond to events:

```
on exitFrame
  go to the frame
end
```

{button See also,AL(`Lingo_go')}

go loop

Syntax: `go loop`

This command causes the playback head to continuously return to the first marker to the left and then loop back. If no markers are to the left of the playback head, the playback head continues to the right.

The `go loop` command is equivalent to the statement `go to the marker(0)` that was used in earlier versions of Lingo.

Example:

This statement causes the movie to loop between the current frame and the previous marker:

```
go loop
```

{button See also,AL(`Lingo_go_loop')}

go next

Syntax: `go next`

This 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 first marker to the left. If there are no markers to the left, the playback head goes to frame 1.

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
```

{button See also,AL(`Lingo_go_next')}

go previous

Syntax: `go previous`

This command sends the playback head to the previous marker in the movie.

If there is only one marker to the left of the frame, the playback head will go back one marker. If there are two (or more) markers to the left of the current frame, the playback head will go back two (or more) markers.

Example:

This statement sends the playback head to the previous marker in the movie:

```
go previous
```

{button See also,AL(`Lingo_go_previous')}

gotoNetMovie

Syntax: `gotoNetMovie` *URL*

or

`gotoNetMovie (`*URL*`)`

This command retrieves and plays a new Shockwave movie from and ftp or HTTP 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.

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.

For more information about Shockwave movies and the internet, see Chapter 14, "Shockwave, the Internet, and Lingo," in *Learning Lingo*.

Example 1:

In this statement, the URL indicates a Director file name:

```
gotoNetMovie "http://www.yourserver.com/movies/movie1.dcr"
```

Example 2:

In this statement, the URL indicates a marker within a file name:

```
gotoNetMovie ¬ "http://www.yourserver.com/movies/buttons.dcr#Contents"
```

Example 3:

In this statement, `gotoNetMovie` is used as a function. The function returns the network ID for the operation.

```
set myNetID = gotoNetMovie ¬
("http://www.yourserver.com/movies/buttons.dcr#Contents")
```

gotoNetPage

Syntax: `gotoNetPage` *"URL", "targetName"*

This command opens a Shockwave movie or another MIME type file.

Only URLs are supported as valid parameters. Relative URLs work if the movie is on an ftp or HTTP server.

targetName is an optional HTML parameter that identifies or names the frame or window name in which you want the page to be 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 movie, wherever it is located.

Use `netDone` to find out when the `gotoNetPage` operation completes.

In the authoring environment, this command launches the preferred browser if it is enabled. In projectors, this command tries to launch the preferred browser. The user sets the preferred browser in the Network Preferences dialog box.

For more information about Shockwave movies and the internet, see Chapter 14, "Shockwave, the Internet, and Lingo," in *Learning Lingo*.

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
```

{button See also,AL(`Lingo_gotoNetPage')}

halt

Syntax: `halt`

This command exits the current handler and any handler that called it. After exiting all handlers, the `halt` command stops the movie.

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
```

{button See also,AL(`Lingo_halt')}

height of cast

This is obsolete. Use **height of member** instead.

height of member

Syntax: `the height of member` *whichCastmember*

This cast member property determines the height in pixels of the cast member specified by *whichCastmember*. The `height of member` property applies only to bitmap and shape cast members. It does not affect field or button cast members.

The `height of member` property can be tested but not set.

Example:

This statement assigns the height of cast member Headline to the variable `vHeight`:

```
set vHeight to the height of member "Headline"
```

{button See also,AL(`Lingo_height_of_member')}

height of sprite

Syntax: `the height of sprite` *whichSprite*

This sprite property determines the vertical size in pixels of the sprite specified by *whichSprite*. The height applies only to bitmap and shape cast members. It does not affect field or button cast members.

Setting this property does not have any effect on bitmap sprites unless the sprite's `stretch` property is set to TRUE. For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet.

The `height of sprite` property can be tested and set.

Example 1:

This statement sets the height of sprite 10 to 26 pixels:

```
set the height of sprite 10 to 26
```

Example 2:

This statement assigns the height of sprite (i + 1) to the variable `vHeight`:

```
set vHeight =  the height of sprite (i + 1)
```

{button See also,AL(`Lingo_height_of_sprite')}

hilite

Syntax: `hilite` *chunkExpression*

This command highlights (selects) the specified chunk in a field sprite. You can select any chunk that Lingo lets you define, such as a character, word, or line.

Example:

This statement highlights the fourth word in the field cast member Comments, which contains the string "Thought for the Day":

```
hilite word 4 of member "Comments"
```

{button See also,AL(`Lingo_hilite')}

hilite of cast

This is obsolete. Use **hilite of member** instead.

hilite of member

Syntax: `the hilite of member` *whichCastmember*

This button property determines whether a check box or radio button is selected.

- When the `hilite of member` button property is TRUE, the check box or radio button is selected.
- When the `hilite of member` button property is FALSE, the check box or radio button is not selected.

When *whichCastmember* is a string, it is used as the cast member name. When *whichCastmember* is an integer, it is used as the cast member number.

The `hilite of member` button property can be tested and set. The default value is FALSE.

Example 1:

This statement checks whether the button named "2400 baud" is selected and sets the baud rate to 2400 if it is:

```
if the hilite of member "2400 baud" = TRUE then ¬
   setBaudRate(2400)
```

Example 2:

This statement uses Lingo to select the button cast member `powerSwitch` by setting `the hilite of member` for the cast member to TRUE:

```
set the hilite of member powerSwitch to TRUE
```

{button See also,AL(`Lingo_hilite_of_member')}

HMStoFrames

Syntax: `HMStoFrames(`*hms* , *tempo* , *dropFrame* , *fractionalSeconds*)

This function converts movies measured in hours-minutes-seconds to the equivalent number of frames.

- The string expression *hms* specifies the time in the form "sHH:MM:SS.FFD", where:

  s is a dash (-) if the time is less than zero, or a space if the time is greater than or equal to zero.

  HH represents number of hours.

  MM represents number of minutes.

  SS represents number of seconds.

  FF represents fraction of a second if *fractionalSeconds* is TRUE. FF represents frames if *fractionalSeconds* is FALSE.

  D is the letter "d" if *dropFrame* is TRUE. D is a space if *dropFrame* is FALSE.

- The expression *tempo* specifies the tempo in frames per second.
- The *dropFrame* argument is a logical expression. When TRUE replaces *dropFrame*, it is a drop frame. When FALSE replaces *dropFrame*, it is not. When the string *hms* ends in a "d", the time is treated as a drop frame, regardless of the value of *dropFrame*.
- The *fractionalSeconds* argument determines the meaning of the fractional seconds. When it is set to TRUE, the numbers after the seconds specify a fraction of a second, to the nearest hundredth of a second. When it is set to FALSE, the numbers after the seconds specify the number of residual frames.

This property can also be used to convert a number of hours, minutes, and seconds into time if you set the *tempo* argument to 1 (one frame = one second). See the example that follows.

Example 1:

This statement determines the number of frames in a 1-minute, 30.1-second movie when the tempo is 30 frames per second. The *dropFrame* and *fractionalSeconds* arguments are both turned off:

```
put HMStoFrames(" 00:01:30.10 ", 30, FALSE, FALSE)

-- 2710
```

Example 2:

This statement converts 600 seconds into minutes:

```
>>  put framesToHMS(600, 1,0,0)

>>  -- " 00:10:00.00 "
```

Example 3:

This statement converts an hour and a half into seconds:

```
>>  put HMStoFrames("1:30:00", 1,0,0)

>>  -- 5400
```

{button See also,AL(`Lingo_HMStoFrames')}

idle

See: on idle event handler

idleHandlerPeriod

Syntax: `the idleHandlerPeriod`

This movie property determines the maximum number of ticks that passes until the movie sends an `idle` message. The default value is 0, which has the movie send idle handler messages as frequently as possible.

When the playback head enters a frame, Director starts a timer, repaints the appropriate sprites on the Stage, and then issues an `enterFrame` event. At this point, if the amount of time set for the tempo setting has elapsed, Director generates an `exitFrame` event and goes to the next specified frame.

However, if the amount of time set for this frame hasn't elapsed, Director waits until the time runs out. During this time, Director periodically generates an `idle` message. The amount of time between `idle` events is determined by `the idleHandlerPeriod`.

Example:

The following statement causes the movie to send an `idle` message at most once per second:

```
set the idleHandlerPeriod = 60
```

{button See also,AL(`Lingo_idleHandlerPeriod')}

idleLoadDone

Syntax: `idleLoadDone(`*loadTag*`)`

This function reports whether Director has loaded all cast members that have the specified load tag.

- The result is TRUE when all cast members with the given tag have been loaded.
- The result is FALSE when some cast members with the given load tag are still waiting to be loaded.

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) = TRUE then play "Kiosk"
```

{button See also,AL(`Lingo_idleLoadDone')}

idleLoadMode

Syntax: `the idleLoadMode`

This system property determines when the `preLoad` and `preLoadMember` commands attempt to load cast members during idle periods. The following values are possible values for `the idleLoadMode` and their effect:

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 itself performs no function; this system property only works in conjunction with the `preLoad` and `preLoadMember` commands.

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:

```
set the idleLoadMode = 2
```

{button See also,AL(`Lingo_idleLoadMode')}

idleLoadPeriod

Syntax: `the idleLoadPeriod`

This property determines the number of ticks that Director waits before returning to attempt to load cast members that are waiting to be loaded. The default value for `the idleLoadPeriod` is 0, which instructs Director to service the load queue as frequently as possible.

Example:

This statement instructs Director to service the set of cast members waiting to be loaded every 1/2 second (30 ticks):

```
set the idleLoadPeriod = 30
```

{button See also,AL(`Lingo_idleLoadPeriod')}

idleLoadTag

Syntax: `the idleLoadTag`

This system property is a number that identifies, or tags, the cast members that have been queued for loading when the computer is idle. The `idleLoadTag` is merely a convenience that identifies the cast members in a group that you want to preload.

The property can be tested and set. When you set the property, it can be any number that you choose.

Example:

This statement makes the number 10 the idle load tag.

```
set the idleLoadTag = 10
```

{button See also,AL(`Lingo_idleLoadTag')}

idleReadChunkSize

Syntax: `the idleReadChunkSize of member` *whichCastmember*

This movie property determines the maximum number of bytes from a cast member that Director can load when it loads cast members from the load queue. The property can be tested and set.

Example:

This statement specifies that 500K is the maximum number of bytes from cast member number 50 that Director can load in one attempt at loading cast members in the load queue:

```
set the idleReadChunkSize to 500000
```

if

Syntax:

```
if logicalExpression then then-statement
    or
if logicalExpression then then-statement
else else-statement
end if
    or
if logicalExpression then
    statement(s)
end if
    or
if logicalExpression then
    statement(s)
else
    statement(s)
end if
    or
if logicalExpression1 then
    statement(s)
else if logicalExpression2 then
    statement(s)
else if logicalExpression3 then
    statement(s)
end if
```

The `if...then` structure evaluates the logical expression specified by *logicalExpression*.

- When the condition is TRUE, Lingo executes the statement(s) that follow `then`.
- When it is FALSE, Lingo executes the statement(s) following `else`. If no statements follow `else`, Lingo exits the `if...then` structure.

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, but it is common practice to do so.

The `else` portion of the statement is optional. If you need to use more than one *then-statement* or *else-statement*, you must end with the form `end if`.

When you use `else`, it always corresponds to the previous `if` statement. This means that sometimes you need to include an `else nothing` statement to associate an `else` keyword with the proper `if` keyword.

Example 1:

This statement checks whether the Return key was pressed and then continues if it was:

```
if the key = RETURN then continue
```

Example 2:

This handler checks whether the Command and q keys were pressed simultaneously, and then executes the subsequent statements if it was:

```
on keyDown

if (the commandDown) and (the key = "q") then

   cleanUp

   quit

end if

end keyDown
```

{button See also,AL(`Lingo_if')}

ilk

Syntax: `ilk(list)`

`ilk(item, type)`

This function indicates the type of a list, rect, or point.

- The syntax `ilk(list)` returns whether *list* is a linear list or property list. For linear lists, `ilk(list)` returns `#list`; for property lists, `ilk(list)` returns `#propList`.
- The syntax `ilk(item, type)` compares the object represented by *item* and indicates whether the object is of the specified *type*. When the object is of the specified type, the `ilk` function returns TRUE. When the object isn't of the specified type, the `ilk` function returns FALSE. The following are the values that are returned for each combination of linear list, property list, point, or rect items and types:

Item	Possible type=returned value
linear list	#list=1, #linearlist=1, #proplist=0, #point=0, #rect=0
property list	#list=1, #linearlist=0, #proplist=1, #point=0, #rect=0
point	#list=1, #linearlist=0, #proplist=0, #point=1, #rect=0
rect	#list=1, #linearlist=0, #proplist=0, #point=0, #rect=1

Example 1:

This statement identifies whether the list named Bids is a property list and displays the result in the Message window:

```
put ilk(Bids, #proplist)
```

Because the list is a property list, the Message window displays 1, which is the numeric equivalent of TRUE.

Example 2:

This statement identifies whether the variable `Total` is a list and displays the result in the Message window:

```
put ilk(Total, #list)
```

Because the variable is not a list, the Message window displays 0, which is the numeric equivalent of FALSE.

importFileInto

Syntax:

```
importFileInto member whichCastmember , fileName

   or

importFileInto member whichCastmember of ¬
   castLib whichCast, fileName

   or

importFileInto(member whichCastMember, fileName)

   or

importFileInto member whichCastmember , URL
```

This command replaces the content of the cast member specified by *whichCastmember* with the file specified by *fileName*.

The `importFileInto` command is useful in three situations:

- When finishing developing a movie, use it to embed media that you have kept linked and external so that it could be edited during the project.
- When generating Score from Lingo, use it to assign content to new cast members that you created. For more information about generating Score and creating new cast members, see "Authoring from Lingo," in *Learning Lingo*.
- When downloading files from the internet, you can use it to download the file at a specific URL. However, to import a file from a URL, it's usually more efficient to use the `preloadNetThing` command to download the file to a local disk first, and then import the file from the local disk. This minimizes problems with waiting for the file to download.

Using this command in projectors can be a problem, because imported files can quickly consume memory.

Example 1:

This handler assigns a URL that contains a GIF file to the variable `tempURL`. The handler then uses the `importFileInto` command to import the file at the URL into a new bitmap cast member:

```
on exitFrame
   set tempURL = "http://www.dukeOfUrl.com/crown.gif"
   importFileInto(new(#bitmap), tempUrl)
end exitFrame
```

Example 2:

This statement replaces the content of the sound cast member Memory with the sound file `Wind`:

```
importFileInto member "Memory", "Wind"
```

Example 3:

These statements download an external file from a URL to the same folder that Director is in, and then imports that file into the sound cast member Norma Desmond Speaks:

```
downLoadNetThing http://www.cbDeMille.com/Talkies.AIF,¬
the applicationPath&"Talkies.AIF"
set the fileName of member "Norma Desmond Speaks" = ¬
```

```
the applicationPath&"Talkies.AIF"
```

in

number of chars in, number of items in, number of lines in, and number of words in functions

inflate rect

Syntax: `inflate (`*rectangle , widthChange , heightChange)*

This command changes the dimensions of the rectangle specified by *rectangle*. The change is relative to the center of the rectangle.

- The *widthChange* parameter specifies how much the rectangle changes horizontally.
- The *heightChange* parameter specifies how much the rectangle changes vertically.

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.

Values less than 0 for horizontal or vertical reduce the rectangle's size.

Example 1:

This statement increases the width of the rectangle by 4 pixels and the height by 2 pixels:

```
inflate (rect(10, 10, 20, 20), 2, 1)

-- Rect (8, 9, 22, 21)
```

Example 2:

This statement increases both the height and width of the rectangle by 20 pixels:

```
inflate (rect(0, 0, 100, 100), -10, -10)

-- Rect (10, 10, 90, 90)
```

initialToggleState of member

Syntax: `the initialToggleState of member` *whichCastmember*

This button cast member property determines the initial state for toggle buttons created with the Button editor.

- When `initialToggleState of member` is TRUE, the button is not toggled initially.
- When `initialToggleState of member` is FALSE, the button is toggled initially.

This property doesn't apply to push buttons. It can be tested and set.

Example:

This statement sets the toggle button cast member Panic to be toggled when it first appears:

```
set the initialToggleState of member "Panic" to FALSE
```

ink of sprite

Syntax: `the ink of sprite` *whichSprite*

This sprite property determines the ink effect applied to the sprite specified by *whichSprite*.

The following ink effects are available:

0-Copy	**9**-Mask
1-Transparent	**32**-Blend
2-Reverse	**33**-Add pin
3-Ghost	**34**-Add
4-Not copy	**35**-Subtract pin
5-Not transparent	**36**-Background transparent
6-Not reverse	**37**-Lightest
7-Not ghost	**38**-Subtract
8-Matte	**39**-Darkest

In the case of background transparent (ink effect 36), you set the color that becomes transparent by selecting the color from the background color chip in the Tools window while the sprite is selected in the Score. You can do the same thing by using Lingo to set the `backColor` property, but this is unpredictable when the sprite has more than 1-bit color.

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.

For further information about ink effects, see *Using Director*.

The ink sprite property can be tested and set. To change any sprite property using Lingo, the sprite must be a puppet.

Example 1:

This statement changes the variable `currentInk` to the value for the ink effect of sprite (i + 1):

```
put the ink of sprite 3 into currentInk
```

Example 2:

This statement gives sprite (i + 1) a matte ink effect by setting the ink effect of sprite property to 8, which specifies matte ink:

```
set the ink of sprite (i + 1) to 8
```

{button See also,AL(`Lingo_ink_of_sprite')}

insertFrame

Syntax: `insertFrame`

This command duplicates the current frame and its content. The duplicate frame is inserted after the current frame and then becomes the current frame. It can be used only during a Score recording session.

This command performs the same function as the `duplicateFrame` command.

Example:

The following handler generates a frame that has the transition cast member Fog assigned in the transition channel followed by a set of empty frames. The number of frames is determined by the argument `numberOfFrames`.

```
on animBall numberOfFrames
   beginRecording
    set the frameTransition to ¬
   the number of member "Fog"
   repeat with i = 0 to numberOfFrames
      insertFrame
   end repeat
   endRecording
end
```

inside

Syntax: `inside(`*point*`,` *rectangle)*

This function indicates whether the point specified by *point* is within the rectangle specified by *rectangle*.

- When the point is within the rectangle, the `inside` function is TRUE.
- When the point is outside the rectangle, the `inside` function is FALSE.

Example:

This statement indicates whether the point Center is within the rectangle Zone and displays the result in the Message window:

```
put inside(Center, Zone)
```

{button See also,AL(`Lingo_inside')}

installMenu

Syntax: `installMenu` *whichCastmember*

This command installs the menu defined in the field cast member specified by *whichCast member*. 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.

For an explanation of how menu items are defined in a field cast member, refer to the `menu` keyword.

Changing menus many times should be avoided. Numerous menu changes have an impact on system resources.

Hierarchical menus aren't available through this command.

If the menu is longer than the screen on the Macintosh, the menu can scroll. In Windows, if the menu is longer than the screen, only part of the menu appears.

Example 1:

This statement installs the menu defined in field cast member 37:

```
installMenu 37
```

Example 2:

This statement installs the menu defined in the field cast member named Menubar by using the `number of member` property to refer to the field cast member:

```
installMenu member "Menubar"
```

Example 3:

This statement disables menus that were installed by the `installMenu` command:

```
installMenu 0
```

{button See also,AL(`Lingo_installMenu')}

instance

This Lingo element is obsolete.

integer

Syntax: `integer(`*numericExpression)*

This 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 1:

This statement rounds off the number 3.75 to the nearest whole integer:

```
put integer(3.75)

-- 4
```

Example 2:

This statement rounds off the value in parentheses. This provides a usable value for the `locH of sprite` property, which requires an integer:

```
set the locH of sprite 1 ¬

   to integer(0.333 * stageWidth)
```

{button See also,AL(`Lingo_integer')}

integerP

Syntax: `integerP(`*expression)*

This function indicates whether the expression specified by *expression* is an integer:

- When *expression* can be evaluated to an integer, `integerP` is TRUE (1).
- When *expression* cannot be evaluated to an integer, `integerP` is FALSE (0).

The "P" in `integerP` stands for "predicate."

Example 1:

This statement checks whether the number 3 can be evaluated to an integer. Because it is an integer, the Message window displays the number 1, which is the numeric equivalent of TRUE:

```
put integerP(3)

-- 1
```

Example 2:

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 the Message window displays the number 0, which is the numeric equivalent of FALSE:

```
put integerP("3")

-- 0
```

Example 3:

This statement checks whether the numerical value of the string in field cast member Entry is an integer, and displays an alert if it isn't.

```
if integerP(value(field "Entry")) = FALSE then ¬

   alert "Please enter an integer."
```

{button See also,AL(`Lingo_integerP')}

intersect

Syntax: `intersect(`*rectangle1 , rectangle2)*

This 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:

```
set newRectangle = intersect(toolKit, Ramp)
```

{button See also,AL(`Lingo_intersect')}

into

This code fragment occurs in a number of Lingo constructs, such as `put...into`.

isPastCuePoint

Syntax: `isPastCuePoint(`*whichSpriteOrSound*`, `*cuePointID*`)`

This function returns the number of times a sprite passes a specified cue point in its media. This function can be used with SoundEdit, QuickTime, or Xtra files that support cue points.

Replace *whichSpriteOrSound* with a sprite channel or a sound channel. 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 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, the function returns TRUE if any cue points identified by *cuePointID* have passed, and FALSE if no cue points are passed.

Example:

This statement allows a sound to play until the third time the cue point Chorus End is passed:

```
if (isPastCuePoint(sound 1, "Chorus End")=3) then
   puppetSound 0
end if
```

isToggled of sprite

Syntax: `the isToggled of sprite` *whichSprite*

This button sprite property determines the state of toggle button sprites whose cast member was created with the Button Editor.

- When `isToggled of sprite` is TRUE, the button sprite is not toggled.
- When `isToggled of sprite` is FALSE, the button sprite is toggled.

This property doesn't apply to push buttons.

This property can be tested and set.

Example:

This statement sets the button type of sprite 1 to the "normal" button type.

```
set the isToggled of sprite 1 to FALSE
```

item...of

Syntax: `item` *whichItem* `of` *chunkExpression*

or

`item` *firstItem* `to` *lastItem* `of` *chunkExpression*

This chunk expression keyword specifies an item or a range of items in a chunk expression. An item in this case is any sequence of characters delimited by commas.

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 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 1:

This statement determines the third item in the chunk expression that consists of names of colors and displays the result in the Message window:

```
put item 3 of "red, yellow, blue green, orange"

-- "blue green"
```

Example 2:

The result is the entire chunk "blue green" because this is the entire chunk within the commas.

This statement determines the third through fifth item in the chunk expression and displays the result in the Message window:

```
put item 3 to 5 of "red, yellow, blue green, orange"

-- "blue green, orange"
```

Example 3:

This statement attempts to determine the third through fifth items in the chunk expression. Because there are only four items in the chunk expression, the fourth item is used instead of the fifth item. The result appears in the Message window:

```
put item 3 to 5 of "red, yellow, blue green, orange"

-- " blue green, orange"

put item 5 of "red, yellow, blue green, orange"

-- ""
```

Example 4:

This statement inserts the item Desk as the fourth item in the second line of the field cast member All Bids:

```
put "Desk" into item 4 of line 2 ¬

   of member "All Bids"
```

{button See also,AL(`Lingo_item_of')}

itemDelimiter

Syntax: `the itemDelimiter`

This property indicates the special character used to separate items.

You can use the `itemDelimiter` function to parse file names by setting `itemDelimiter` to a colon (:) on the Macintosh or a backslash (\) in Windows. Restore `the itemDelimiter` to a comma (,) for normal operation. Be sure to restore it to "," for normal operation.

The `itemDelimiter` function can be tested and set.

Example:

This handler determines the last component in a Macintosh pathname. The handler first records what the current delimiter is 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
   set save = the itemDelimiter
   set the itemDelimiter = ":"
   set f = the last item of pathName
   set the itemDelimiter = save
   return f
end
```

key

Syntax: `the key`

This function indicates the last key that was pressed. (This value is the 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. This is a way to provide keyboard shortcuts and other forms of interactivity for the user. When used in a primary event handler, the actions you specify are the first to be executed.

Note: The value of `the key` doesn't update 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 1:

These statements cause the movie to pause when the user presses the Return key. By setting the `keyDownScript` property 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 pauses the movie if it is:

```
on prepareMovie

   set the keyDownScript to "checkKey"

end prepareMovie

on checkKey

   if the key = RETURN then pause

end
```

Example 2:

This `on keyDown` handler checks whether the last key pressed is the Enter key, and then calls the handler `on addNumbers` if it is:

```
on keyDown

   if the key = ENTER then addNumbers

end keyDown
```

{button See also,AL(`Lingo_key')}

keyCode

Syntax: `the keyCode`

This function gives the numerical code for the last key pressed. (This keyboard code is the key's numerical value, not the ANSI value.)

You can use the `keyCode` function to detect when the user has pressed the arrow or function keys, which cannot be specified by `the key` function. The value of `keyCode` can vary on different keyboards.

Use the sample movie "Keyboard Lingo" to test which characters correspond to different keys on different keyboards.

The `keyCode` function can be tested but not set.

Example 1:

This handler uses the Message window to display the appropriate key code each time a key is pressed:

```
on enterFrame

   set the keydownScript = "put the keyCode"

end
```

Example 2:

This statement checks whether the up arrow (whose key code is 126) is pressed, and goes to the previous marker if it is:

```
if the keyCode = 126 then go to marker(-1)
```

Example 3:

This handler checks whether one of the arrow keys was pressed, and responds accordingly if one was:

```
on keyDown
  case (the keyCode) of
    123: TurnLeft
    126: GoForward
    125: BackUp
    124: TurnRight
  end case
end keyDown
```

{button See also,AL(`Lingo_keyCode')}

keyDown

See: on keyDown handler

keyDownScript

Syntax: `the keyDownScript`

This property specifies the Lingo that is executed when a key is pressed. The Lingo is written as a string, surrounded by quotation marks. It 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 pass 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 1:

This statement sets `the keyDownScript` to
`if the key = RETURN then go to the frame + 1`. When this is in effect, the movie always goes to the next frame whenever the user presses the Return key.

```
set the keyDownScript to "if the key = RETURN then go to ¬
the frame + 1"
```

Example 2:

This statement sets `the keyDownScript` to the custom handler *myCustomHandler*. A Lingo custom handler must be enclosed in quotation marks when used with the `keyDownScript` property.

```
set the keyDownScript to "myCustomHandler"
```

{button See also,AL(`Lingo_keyDownScript')}

keyPressed

Syntax: `the keyPressed`

This system property gives the character assigned to the key that was last pressed. The result is in the form of a string. When no key has been pressed, `the keyPressed` is an empty string.

The `keyPressed` property updates when the user presses keys while Lingo is in a repeat loop. This is an advantage over `the key`, 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 he or she did:

```
if the keyPressed = RETURN then updateData
```

keyUpScript

Syntax: `the keyUpScript`

This property specifies the Lingo that is executed when a key is released. The Lingo is written as a string, surrounded by quotation marks. It 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 pass 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 1:

This statement sets the `keyUpScript` to `if the key = RETURN then go the frame + 1`. When this is in effect, the movie always goes to the next frame whenever the user presses the Return key.

```
set the keyUpScript ¬
to "if the key = RETURN then go to the frame + 1"
```

Example 2:

This statement sets the `keyUpScript` to the custom handler *myCustomHandler*. A Lingo custom handler must be enclosed in quotation marks when used with the `keyUpScript` property.

```
set the keyUpScript to "myCustomHandler"
```

{button See also,AL(`Lingo_keyUpScript')}

label

Syntax: `label(`*expression)*

This 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 1:

This statement sends the playback head to the tenth frame after the frame labeled Start:

```
go to label("Start") + 10
```

Example 2:

This statement assigns the frame number of the fourth item in the label list to the variable `whichFrame`:

```
set whichFrame =  label(line 4 of the labelList)
```

{button See also,AL(`Lingo_label')}

labelList

Syntax: `the labelList`

This function gives a listing of the frame labels in the current movie, one label per line. The result is a Return-delimited string containing one label per line. The 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 1:

This statement makes a listing of frame labels the content of the field cast member Key Frames:

```
put the labelList into member "Key Frames"
```

Example 2:

This handler determines the label that starts the current scene:

```
on FindLastLabel
  repeat with i = 1 to the number of lines in the labelList
    if the frame < label(line i of the labelList) then
      return line i - 1 of the labelList
    end if
  end repeat
  return line (the number of lines in the labelList - 1)¬
  of the labelList
end FindLastLabel
```

{button See also,AL(`Lingo_labelList')}

labelString of member

Syntax: `the labelString of member` *whichCastMember*

This button cast member property determines the button's label. The `labelString of member` property's value is a string.

This property can be tested but not set.

Example:

This statement assigns the current label of button cast member Panic to the variable `buttonName`:

```
set buttonName to the labelString of member "Panic"
```

last

Syntax: `the last` *chunk* `in (`*chunkExpression)*

This function identifies the last chunk specified by *chunk* of the chunk expression specified by *chunkExpression*.

Chunk expressions refer to any character, word, item, or line in any container of characters. Containers include the contents of field cast members; variables that hold strings; and specified characters, words, items, lines, and ranges within containers.

Example 1:

This statement identifies the last word of the string "Macromedia, the multimedia company" and displays the result in the Message window:

```
put the last word of "Macromedia,¬
   the multimedia company"
```

The result is the word `company`.

Example 2:

This statement identifies the last character 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 letter `y`.

{button See also,AL(`Lingo_last')}

lastClick

Syntax: `the lastClick`

This function gives the time in ticks (60ths of a second) since the mouse button was last pressed.

The `lastClick` function can be tested but not set.

Example:

This statement checks whether it has been 10 seconds since the last mouse click, and sends the playback head to the marker No Click if it has:

```
if the lastClick > 10 * 60 then go to "No Click"
```

{button See also,AL(`Lingo_lastClick')}

lastEvent

Syntax: `the lastEvent`

This function gives the time in ticks (60ths of a second) since the last mouse click, rollover, or key press occurred.

Example:

This statement checks whether it has been 10 seconds since the last mouse click, rollover, or key press, and sends the playback head to the marker Help if it has:

```
if the lastEvent > 10 * 60 then go to "Help"
```

{button See also,AL(`Lingo_lastEvent')}

lastFrame

Syntax: `the lastFrame`

This property is the number of the last frame in the movie.

The `lastFrame` 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`

This function gives the time in ticks (60ths of a second) since the last key was pressed.

Example:

This statement checks whether it has been 10 seconds since the last key was pressed, and sends the playback head to the marker No Key if it has:

```
if the lastKey > 10 * 60 then go to "No Key"
```

{button See also,AL(`Lingo_lastKey')}

lastRoll

Syntax: `the lastRoll`

This function gives the time in ticks (60ths of a second) since the mouse was last moved.

Example:

This statement checks whether it has been 45 seconds since the mouse was last moved, and sends the playback head to the marker No Roll if it has:

```
if the lastRoll > 45 * 60 then go to "Attract Loop"
```

{button See also,AL(`Lingo_lastRoll')}

left of sprite

Syntax: `the left of sprite` *whichSprite*

This sprite property is 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.

The `left of sprite` property can be tested, but not set. Use the `rect of sprite` property to set the left horizontal coordinate of a sprite.

Example 1:

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 the left of sprite 3 < 0 then offLeftEdge
```

Example 2:

This statement measures the left horizontal coordinate of the sprite numbered (i + 1) and assigns the value to the variable named `vLowest`:

```
set vLowest = the left of sprite (i + 1)
```

{button See also,AL(`Lingo_left_of_sprite')}

length

Syntax: `length(`*string)*

This function gives the number of characters in the string specified by *string*. Spaces and control characters like Tab and Return count as characters.

Example 1:

This statement displays the number of characters in the string "Macro"&"media":

```
put length("Macro" & "media")

-- 10
```

Example 2:

This statement checks whether the content of the field cast member File Name has more than 31 characters and displays an alert if it does:

```
if length(field "File Name") > 31 then ¬

   alert "That file name is too long."
```

{button See also,AL(`Lingo_length')}

line...of

Syntax: `line` *`whichLine`* `of` *`chunkExpression`*

or

`line` *`firstLine`* `to` *`lastLine`* `of` *`chunkExpression`*

This chunk expression keyword specifies a line or a range of lines in a chunk expression. A line chunk is any sequence of characters delimited by Return characters, 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 1:

This statement assigns the first four lines of the variable `Action` to the field cast member To Do:

```
set the text of member "To Do" = line 1 to 4 ¬
   of Action
```

Example 2:

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 word 2 of line 3 of Notes
```

{button See also,AL(`Lingo_line_of')}

lineCount of member

Syntax: `the lineCount of member` *whichCastMember*

This field cast member property indicates the number of lines that appear in the field cast member on the Stage. The number of lines depends on how 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`:

```
set numberOfLines = the lineCount of member "Today's News"
```

lineHeight

Syntax: `lineHeight(member` *whichCastMember*`,` *lineNumber*`)`

This function gives 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`:

```
set headline = lineHeight(member "Today's News",1)
```

lineHeight of member

Syntax: `the lineHeight of member` *whichCastmember*

This field 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 of member` temporarily overrides the system's setting. After closing a movie, the field's line spacing returns to the system's setting. To use the desired line spacing throughout a movie, set `the lineHeight of member` in an `on prepareMovie` handler.

The `lineHeight of member` property can be tested and set.

Example:

This statement sets the variable `oldHeight` to the current `lineHeight of member` setting for the field cast member Rokujo Speaks:

```
set oldHeight = the lineHeight of member "Rokujo Speaks"
```

{button See also,AL(`Lingo_lineHeight_of_member')}

linePosToLocV

Syntax: `linePosToLocV(member` *whichCastMember*`,` *lineNumber*`)`

This function gives a specific line's distance, in pixels, from the top edge of the field cast member.

Example:

This statement measures the distance, in pixels, that the second line of the field cast member Today's News is from the top of the field cast member and assigns the result to the variable `startOfString`:

```
set startOfString = linePosToLocV(member "Today's News",2)
```

lineSize of member

Syntax: `the lineSize of member` *whichCastmember*

This shape cast member property determines the thickness, in pixels, of the border of the specified shape cast member. It can be tested and set.

Example:

This statement sets the thickness of the shape cast member Answer Box to 5 pixels:

```
set the lineSize of member "Answer Box" = 5
```

lineSize of sprite

Syntax: `the lineSize of sprite` *whichSprite*

This sprite property determines the thickness, in pixels, of the border of the sprite specified by *whichSprite*. The `lineSize of sprite` property applies only to shape sprites. For non-rectangular shapes the border is the edge of the shape, not its bounding rectangle.

The setting of `the lineSize of sprite` takes precedence over `the lineSize of member`. If Lingo changes `the lineSize of member` while a sprite is on Stage, the sprite's `lineSize` settting 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.

The `lineSize of sprite` property can be tested and set.

Example 1:

This statement displays the thickness of the border of sprite 4:

```
set thickness =  the lineSize of sprite 4
```

Example 2:

This statement sets the thickness of the border of sprite 4 to 3 pixels:

```
set the lineSize of sprite 4 to 3
```

list

Syntax: `list(`*value1* `,` *value2* `,` *value3...)*

This function 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, surround 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:

```
set designers = list("Gee", "Kayne", "Ohashi")
```

The result is the list ["Gee", "Kayne", "Ohashi"].

listP

Syntax: `listP(`*item)*

This function indicates whether the item specified by *item* is a list, rect, or point.

- When `listP` is TRUE (1), the item specified by *item* is a list, rect, or point.
- When `listP` is FALSE (0), the item specified by *item* is not a list, rect, or point.

Example 1:

This statement checks whether the list in the variable `designers` is a list, rect, or point and displays the result in the Message window:

```
put listP(designers)
```

The result is 1, which is the numerical equivalent of TRUE.

Example 2:

This statement checks whether the point in the variable `Spot` is a list, rect, or point and displays the result in the Message window:

```
put listP(Spot)
```

The result is 1, which is the numerical equivalent of TRUE.

{button See also,AL(`Lingo_listP')}

loaded of cast

This is obsolete. Use **loaded of member** instead.

loaded of member

Syntax: `the loaded of member` *whichCastMember*

This cast member property specifies whether the cast member specified by *whichCastMember* is loaded into memory.

- When `the loaded of member` is TRUE, the cast member is loaded into memory.
- When `the loaded of member` is FALSE, the cast member is not loaded into memory.
- Different cast member types have slightly different behaviors for loading.
- Shape and script cast members are always loaded in memory.
- Movie cast members are never unloaded.
- Digital video cast members can be preloaded and unloaded independently of whether they are being used. (A digital video cast member plays faster from memory than from disk.)

The `loaded of member` property can be tested but not set.

Example:

This statement checks whether cast member Demo Movie is loaded in memory and goes to an alternate movie if it isn't:

```
if the loaded of member "Demo Movie" = FALSE then ¬
go to "Waiting"
```

{button See also,AL(`Lingo_loaded_of_member')}

loc of sprite

Syntax: `the loc of sprite` *whichSprite*

This property determines the Stage coordinates of the specified sprite's registration point. The value is given as a point. The `loc of sprite property` can be tested and set.

Example:

This statement checks the Stage coordinates of sprite 6. The result is the point (50, 100):

```
put the loc of sprite 6

-- point(50, 100)
```

{button See also,AL(`Lingo_loc_of_sprite')}

locH of sprite

Syntax: `the locH of sprite` *whichSprite*

This sprite property is the horizontal position of the specified sprite's registration point. Sprite coordinates are relative to the upper left corner of the Stage. See *Using Director* for information about registration points.

The `locH of sprite` property can be tested and set. For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet.

Example 1:

This statement checks whether the horizontal position of sprite 9's registration point is to the right of the right edge of the monitor and moves the sprite's right edge to the edge of the Stage if it is:

```
if the locH of sprite 9 > the stageRight then ¬
   set the locH of sprite 9 to the stageRight
```

Example 2:

This statement puts sprite 15 at the same horizontal location as the mouse click:

```
set the locH of sprite (15) to the mouseH
```

{button See also,AL(`Lingo_locH_of_sprite')}

locToCharPos

Syntax: `locToCharPos(member` *whichCastMember*`,` *location*`)`

This 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 1:

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`:

```
set pageDesign to locToCharPos(member "Today's News", point(100,100))
```

Example 2:

This handler tells which character is under the cursor when the user clicks the mouse over the field sprite Information:

```
on mouseDown
  put locToCharPos(member "Information", the clickLoc - ¬
  the loc of sprite (the clickOn))
end
```

locV of sprite

Syntax: `the locV of sprite` *whichSprite*

This sprite property is the vertical position of the specified sprite's registration point. Sprite coordinates are relative to the upper left corner of the Stage. See *Using Director* for information about registration points.

The `locV of sprite` property can be tested and set. For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet.

Example 1:

This statement checks whether the vertical position of sprite 9's registration point is below the bottom of the Stage, and moves the sprite's bottom edge to the bottom of the Stage if it is:

```
if the locV of sprite 9 > the stageBottom then ¬
   set the locV of sprite 9 to the stageBottom
```

Example 2:

This statement puts sprite 15 at the same vertical location as the mouse click:

```
set the locV of sprite (15) to the mouseV
```

{button See also,AL(`Lingo_locV_of_sprite')}

locVToLinePos

Syntax: `locVToLinePos(member` *whichCastMember*`,` *locV*`)`

This 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`:

```
put locVToLinePos(member "Today's News", 150) into pageBreak
```

log

Syntax: `log(`*number)*

This function calculates the natural logarithm of the number specified by *number*, which must be a decimal number greater than zero.

Example 1:

This statement assigns the natural logarithm of 10.5 to the variable `Answer`. The result is calculated to two decimal places:

```
set Answer = log(10.5)
```

Example 2:

This statement calculates the natural logarithm of the square root of the value `Number`, and then assigns the result to the variable `Answer`:

```
set Answer = log(the sqrt of Number)
```

long

See: the **date** and **time** functions

loop

Syntax: `loop`

This 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 of cast

This is obsolete. Use **loop of member** instead.

loop of member

Syntax: `the loop of member` *whichCastmember*

This digital video cast member property determines whether the specified digital video movie cast member is set to loop.

- When `the loop of member` is TRUE, the digital video movie cast member loops.
- When `the loop of member` is set to FALSE, the digital video movie cast member doesn't loop.

Example:

This statement sets the QuickTime movie cast member Demo to loop:

```
set the loop of member "Demo" to 1
```

machineType

Syntax: `the machineType`

This function indicates the kind of computer that is currently being used. These codes indicate the type of computer:

1	Macintosh 512Ke	**25**	Macintosh LCIII
2	Macintosh Plus	**27**	PowerBook Duo 210
3	Macintosh SE	**28**	Macintosh Centris 650
4	Macintosh II	**30**	PowerBook Duo 230
5	Macintosh IIx	**31**	PowerBook 180
6	Macintosh IIcx	**32**	PowerBook 160
7	Macintosh SE/30	**33**	Macintosh Quadra 800
8	Macintosh Portable	**35**	Macintosh LC II
9	Macintosh IIci	**42**	Macintosh IIvi
11	Macintosh IIfx	**45**	PowerMac 7100/70
15	Macintosh Classic	**46**	Macintosh IIvx
16	Macintosh IIsi	**47**	Macintosh Color Classic
17	Macintosh LC	**48**	PowerBook 165c
18	Macintosh Quadra 900	**50**	Macintosh Centris 610
19	PowerBook 170	**52**	PowerBook 145
20	Macintosh Quadra 700	**53**	PowerComputing 8100/100
21	Classic II	**73**	PowerMac 6100/60
22	PowerBook 100	**76**	Macintosh Quadra 840av
23	PowerBook 140	**256**	IBM PC-type machine
24	Macintosh Quadra 950		

Note: These codes are for general classification purposes only. It is unwise to use them to make assumptions about the performance or screen size of the computer your movie is running on.

Example 1:

This statement checks whether the computer is a Macintosh Classic and plays the movie "Classic Movie" if it is:

```
if the machineType = 15 then play "Classic Movie"
```

Example 2:

These statements check whether the current operating system is Windows or Macintosh, and runs a handler intended for that platform:

```
if the machineType = 256 then
```

```
    WindowsActions
else
    MacintoshActions
end if
```

{button See also,AL(`Lingo_machineType')}

map

Syntax: `map(`*targetRect*`, `*sourceRect*`, `*destinationRect*`)`

or

`map(`*targetPoint*`, `*sourceRect*`, `*destinationRect*`)`

This function is used to position and size a rectangle or point, based on the relationship of a source rectangle to a target rectangle.

Example:

This handler modifies the rectangle of sprite n so that the sprite's new rect has the same relationship to its old rect as the dimensions of the Stage have to the rect of sprite 2:

```
on scaleMySprite n
   set the stretch of sprite to TRUE
   set the rect of sprite n = ¬
      map(the rect of sprite n, ¬
      the rect of sprite 2, ¬
      the rect of the stage)
   updateStage
end scaleMySprite
```

margin of member

Syntax: `the margin of member` *whichCastmember*

This field cast member property determines the size, in pixels, of the margin inside the field box.

Example:

The following sets the margin inside the box for the field cast member Today's News to 15 pixels:

```
set the margin of member "Today's News" to 15
```

marker

Syntax: `marker(`*integerExpression*`)`

or

`marker("`*string*`")`

This function returns the frame number of markers before or after the current frame. This can be 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 zero. 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 current frame.
- `marker(-2)` returns the frame number of the second marker before the current frame.

If the argument for marker is a string, marker returns the frame number of the first frame whose marker label matches the string.

Example 1:

This statement sends the playback head to the beginning of the current frame:

```
go to marker(0)
```

Example 2:

This statement sets the variable `nextMarker` equal to the next marker in the Score:

```
set nextMarker =  marker(1)
```

{button See also,AL(`Lingo_marker')}

mAtFrame

This method was used in earlier versions of Director. It is now obsolete and has been replaced by `stepFrame` events.

{button See also,AL(`Lingo_mAtFrame')}

max

Syntax: `max(list)`

or

`max (value1, value2, value3, ...)`

This 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 that < 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 in the content of the field cast member Congratulations:

```
on findWinner Bids
   set Winner = max(Bids)
   set the text of member "Congratulations" = ¬
   "You have won, with a bid of $" & Winner &"!"
end
```

maxInteger

Syntax: `the maxInteger`

This 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 31st power, minus 1).

This 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 fast as integers, but 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
  set b = 31
  set v = the maxInteger
  repeat while v > 0
    put b && "-" && v
    set b =  b-1
    set v =  v/2
  end repeat
end showMaxValues
```

mci

Syntax: `mci "`*string*`"`

The multimedia extensions for Windows respond to commands sent to the Media Control Interface (MCI). You can use the `mci` command to pass the strings specified by *strings* to the Windows media control interface.

Strings passed by the `mci` command play only in Windows; they are not executed on the Macintosh. Because the Macintosh does not support the MCI interface, the `mci` command gives you a way to include commands intended for the Windows environment within a movie that you create and can play on the Macintosh.

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"
```

mDescribe

This method was used in earlier versions of Director to return a list of all commands, arguments, and results that an XObject had available. Use `mMessageList` instead.

{button See also,AL(`Lingo_mDescribe')}

mDispose

This method was used in earlier versions of Lingo to support XObjects. Use lists and parent lists instead.

me

Syntax: `me`

This keyword can be used within parent scripts and behaviors as a shorthand means of referring to the current object that is an instance of the parent script or the behavior. It's basically a variable that contains the memory address of the object.

The term itself has no predefined meaning in Lingo. The term `me` is used by convention.

Example 1:

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:

```
set myBird1 to new(script "Bird")
```

This is the `on new` handler of the Bird script:

```
on new me

   return me

end
```

Example 2:

These are two sets of handlers that 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
  set myData to theData
  return me
end

on stepFrame me
  ProcessData me
end
```

This is the second set:

```
property myData

on new myAddress, theData
  set myData to theData
  return myAddress
end

on stepFrame myAddress
  ProcessData myAddress
end
```

{button See also,AL(`Lingo_me')}

media of member

Syntax: `the media of member` *whichCastmember*

This function returns data that describes the specified cast member. The result is a set of numbers that identifies the cast member.

You can use `the media of member` to copy the content of one cast member into another cast member by setting the second cast member's `media of member` value to `the media of member` value for the first.

For a film loop cast member, `the media of member` is a selection of frames and channels in the Score.

Setting `the media of member` can use large amounts of memory. It is best used during authoring only. To swap media in a projector, it's more efficient to set the `media of sprite` property.

Example

This statement copies the content of the cast member Sunrise into the cast member Dawn by setting the `media of member` value for Dawn to the `media of member` value for Sunrise:

```
set the media of member "Dawn" to the ¬
media of member "Sunrise"
```

{button See also,AL(`Lingo_media_of_member')}

mediaReady of member

Syntax: `the mediaReady of member` *whichCastmember*

This property determines if the specifed cast member is downloaded from the internet and available on the local disk

- When `the mediaReady of member` is TRUE, the media is available on a local disk.
- When `the mediaReady of member` is FALSE, the media isn't available on a local disk.

This property is only useful when streaming a movie or cast file. Streaming a movie is activated by setting the **Movie > Playback** properties in the Modify menu to Use Media As Available or Show Placeholders.

For a demonstration of the `mediaReady of member` function, see the sample movie **Streaming Shockwave**.

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 the mediaReady of member "background" = TRUE then
  set the memberNum of sprite 2 to 10
  -- 10 is the number of cast member "background"
end if
```

member

Syntax: `member` *whichCastmember*

`member` *whichCastmember* `of castLib` *whichCast*

This 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.

Example 1:

The following statement sets the `hilite` property of the button cast member named Enter Bid to TRUE:

```
set the hilite of member "Enter Bid" to TRUE
```

Example 2:

This statement puts the name of sound cast member 132 into the variable `soundName`:

```
set soundName = the name of member 123 ¬
of castLib "Las Vegas"
```

Example 3:

This statement determines whether cast member 9 has a name assigned:

```
if the name of member 9 = EMPTY then exit
```

member of sprite

Syntax: `the member of sprite` *whichSprite*

This property specifies a sprite's cast member and cast.

The `member of sprite` property differs from `memberNum of sprite`, which specifies only the sprite's number for its location in the cast but doesn't specify the cast itself. The `member of 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, you can use one of the following formats:

- The full member and cast description (`set the member of sprite x to ¬ member A of castLib B`)
- The cast member name `(set the member of sprite x to ¬ member "MELODY.WAV")`
- The unique integer that includes all cast libraries and corresponds to the mouseMember function (`set the member of sprite x to 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, then only the first name is used.

To specify a cast member by number only 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 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. The sprite has other properties that vary with the type of media element in that channel in the Score. For instance, 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. You'll likely not 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 1:

This statement assigns cast member 3 of cast number 4 to sprite 15:

```
set the member of sprite 15 = member 3 of castLib 4
```

Example 2:

The following handler shows a way to use the global numeric identifier from the `mouseMember` function with the cast member and cast identifier of the `member` sprite property:

```
--  This handler determines whether the mouse is within
--  the irregular outlines of a bitmap sprite with matte ink
on exitFrame
  set MM to the mouseMember
  set target to the number of member "hotspot" of castLib "extra buttons"
  if target = MM then put "above the hotspot"
  go the frame
```

```
end exitFrame
```

Example 3:

This handler uses the `member` sprite property in a different way, by converting the `mouseMember` numeric identifier into a name including cast member and cast name:

```
--  A handler to highlight a sprite when clicked on, and
--  to revert to "copy" ink if rolled-off:
on mouseDown
  set myMember to the member of sprite (the clickOn)
  set copyInk to 0
  set notCopy to 4
  repeat while the stillDown
    set MM to the mouseMember
    if (MM < 1) then set newInk to copyInk
      else if (member MM = myMember) then set newInk to notCopy
      else set newInk to copyInk
    set the ink of sprite (the clickOn) to newInk
    updateStage
  end repeat
end
```

{button See also,AL(`Lingo_member')}

memberNum of sprite

Syntax: `the memberNum of sprite` *whichSprite*

This sprite property identifies the number of the cast member associated with the specified sprite. Its value is the number for the cast member's location only; it doesn't refer to the cast member's cast.

The `memberNum` property is useful for switching cast members assigned to a sprite as long as the cast members are within the same cast. To switch among cast members in different casts, use `the member of sprite`. For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet.

A typical use of the `memberNum` property is to exchange 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.

The `memberNum of sprite` property can be tested and set.

Example 1:

The following statement switches the cast member assigned to sprite 3 to cast member number 35:

```
set the memberNum of sprite 3 to 35
```

Example 2:

This statement assigns the cast member Narrator to sprite 10 by setting `the memberNum of sprite` to Narrator's cast number:

```
set the memberNum of sprite 10 = the number of member "Narrator"
```

Example 3:

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
  set upButton to the memberNum of sprite (the clickOn)
  set downButton to upButton + 1
  repeat while the stillDown
    if rollover(the clickOn) then set the memberNum of sprite (the
clickOn) to ¬              downButton
    else set the memberNum of sprite (the clickOn) to upButton
    updateStage
  end repeat
  if rollover (the clickOn) then put "the button was activated"
end
```

{button See also,AL(`Lingo_memberNum_of_sprite')}

members

See: number of members property

memberType of member

Use `type of member` instead.

{button See also,AL(`Lingo_memberType_of_member')}

memorySize

Syntax: `the memorySize`

This function returns the total amount of memory allocated to the program, whether in use or free. It is useful for checking minimum memory requirements. The value is given in bytes.

Example:

This statement checks whether the computer allocates less than 500K and displays an alert if it does:

```
if the memorySize < 500 * 1024 then alert ¬
   "There is not enough memory to run this movie."
```

{button See also,AL(`Lingo_memorySize')}

menu

Syntax:

```
menu: menuName
    itemName | script
    itemName | script
    ...
```

or

```
menu: menuName
itemName | script
itemName | script
...
[more menus ]
```

This keyword specifies the actual content of custom menus, in conjunction with the `installMenu` command. Menu definitions are typed in field cast members. You refer to a particular menu definition by its cast member name or number.

The `menu` keyword specifies the name of the menu. In the subsequent lines you can specify the menu items for that menu. You can have a script execute when the user chooses that item by putting the script after the vertical bar (`|`) symbol. A new menu is defined by the subsequent occurrence of the `menu` keyword.

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. Because this formatting isn't available for many Windows computers, avoid this formatting in projectors that might play in Windows.)

Symbol	Example-Description (key combination)
`|`	`Open/O | go to frame "Open"`-Associates a script with the menu item
`@`	`menu: @`-On the Macintosh, creates the Apple symbol and enables Macintosh menu bar items when you define an Apple menu
`(`	`Save(`-Disables the menu item
`(-`	`(- `-Creates a disabled line in the menu
`!Ã`	`!ÃEasy Select`-On the Macintosh, checks the menu with a checkmark (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
`/`	`Quit/Q`-Defines a command-key equivalent

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.

Note: The following formatting tags work on the Macintosh only:

@ (for the Apple symbol)

`!Ã (Checkmarks for menu items)`

<B (Boldface)

<I (Italic text)

<U (Underlined text)

<0 (Outlined text)

<S (Shadow text)

Example:

This set of statements specifies the content of a custom File menu. 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"
```

{button See also,AL(`Lingo_menu')}

menuItem

See: name of menuItem, number of menuItems, checkMark of menuItem, enabled of menuItem, and script of menuItem menu item properties.

method

Syntax: `method methodName [`*argument1*`] [, `*argument2*`] ...`

This keyword is now obsolete. It was used to define a method in earlier versions of Director. Use lists and parent scripts instead.

{button See also,AL(`Lingo_method')}

mGet

This method was used for managing arrays in earlier versions of Director. Use lists instead.

{button See also,AL(`Lingo_mGet')}

min

Syntax: `min(`*list)*

or

`min (`*a1, a2, a3...*`)`

This 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
set vLowest = min(bids)
set the text of member "Sorry" = ¬
   "We're sorry, your bid of $" & vLowest && "is not a
   winner!"
end
```

{button See also,AL(`Lingo_min')}

mInstanceRespondsTo

This method is now obsolete. It was used in earlier versions of Director to check whether an instance of an XObject responds to a specific message.

mMessageList

Syntax: *XObject*`(mMessageList)`

This method is used with Xtras and XObjects to return a string that describes the Xtra or XObject and its methods.

mName

This method is now obsolete. It was used to determine which XObject created a specific instance.

mNew

This method is now obsolete. It was used to create XObjects and factories in earlier versions of Director. You should now use parent scripts and lists instead.

{button See also,AL(`Lingo_mNew')}

mod

Syntax: *integerExpression1* `mod` *integerExpression2*

This arithmetic 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 1:

This statement divides 7 by 4 and then displays the remainder in the Message window:

```
put 7 mod 4
```

The result is 3.

Example 2:

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 that has the number in the variable `mySprite` is an odd-numbered sprite by dividing the sprite number by 2 and then checking whether the remainder is 1. When the remainder is 1, which is the result for an odd-numbered number, the ink effect is set to copy:

```
on setInk
  if (mySprite mod 2) = 1 then
    set the ink of sprite mySprite to 0
  else
    set the ink of sprite mySprite to 8
  end if
end setInk
```

Example 3:

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
  set theBitmaps to [2,3,4,5,6,7]
  --  Specify which sprite channel is affected
  set theChannel to 1
  --  This cycles through the list
  set gCounter to 1 + (gCounter mod count(theBitmaps))
  set the memberNum of sprite theChannel to getAt(theBitmaps, gCounter)
  go the frame
end exitFrame
```

modal of window

Syntax: `the modal of window "`*window*`"`

This window property specifies whether movies can respond to events that occur outside the window specified by *window*.

- When the `modal of window` property is TRUE, movies cannot respond to events outside the window.
- When the `modal of window` property is FALSE, movies can respond to events outside the window.

Setting `the modal of window` to TRUE lets you make a specific movie in a window the only movie that the user can interact with.

Example:

This statement lets movies respond to events outside of the window Tool Panel:

```
set the modal of window "Tool Panel" to FALSE
```

modified of cast

This is obsolete. Use **modified of member** instead.

modified of member

Syntax: `the modified of member` *whichCastmember*

This function indicates whether the cast member specified by *whichCastmember* has been modified since it was read from the movie file.

- When `the modified of member` is TRUE (1), the cast member has been modified since it was read from the movie file.
- When `the modified of member` is FALSE (0), the cast member has not been modified since it was read from the movie file.

Example:

This statement tests whether the cast member Introduction has been modified since it was read from the movie file:

```
put the modified of member "Introduction"
```

The result is 0, which is the numerical equivalent of FALSE.

mostRecentCuePoint

Syntax: `the mostRecentCuePoint`

This property is the number that identifies the most recent cue point passed in the sprite. The value is the cue point's ordinal number of the cue point. If no cue points have been passed, the value is 0.

This property can be used with SoundEdit, QuickTime digital video, and Xtras that support cue points.

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 examples has the message window display the number for the most recent cue point passed in the sprite in sprite channel 1:

```
put the mostRecentCuePoint of sprite 1
```

{button See also,AL(`Lingo_mostRecentCuePoint')}

mouseChar

Syntax: `the mouseChar`

This integer function, used for field sprites, gives the number of the character that is under the cursor when the function 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` function can change in a handler or repeat loop. If a handler or repeat loop is using this function multiple times, it's usually a good idea to call the function once and assign its value to a local variable.

Example 1:

This statement determines whether the cursor is not 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 ¬

  set member "Instructions" = ¬
     "Please point to a character."
```

Example 2:

This statement assigns the character under the cursor in the specified field to the variable `currentChar`:

```
set currentChar = char (the mouseChar) of ¬
 member (the mouseMember)
```

Example 3:

This handler highlights the character under the cursor when the mouse button is pressed:

```
on mouseDown
  set thisField to the member of sprite (the clickOn)
  if the mouseChar < 1 then exit
  set lastChar to 0
  repeat while the stillDown
    set MC to the mouseChar
    if MC < 1 then next repeat
    if MC <> lastChar then
      hilite word MC of field thisField
      set lastChar to MC
    end if
  end repeat
end
```

{button See also,AL(`Lingo_mouseChar')}

mouseDown

Syntax: `the mouseDown`

This function indicates whether the mouse button is currently being pressed.

- When `the mouseDown` is TRUE, the button is being pressed.
- When `the mouseDown` is FALSE, the button is not being pressed.

Example 1:

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 2:

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
```

{button See also,AL(`Lingo_mouseDown')}

mouseDownScript

Syntax: `the mouseDownScript`

This property specifies the Lingo that is executed when the mouse button is pressed. The Lingo is written as a string, surrounded by quotation marks. It can be a simple statement or a calling script for a handler.

When the mouse button is pressed and the `mouseDownScript` property is defined, Lingo executes the instructions specified for the `mouseDownScript` property first. Unless the instructions include the `pass` command so that the `mouseDown` message can pass on to other objects in the movie, no other `on mouseDown` handlers are executed.

Setting the `mouseDownScript` property performs the same function as the `when keyDown then` command that appeared in earlier versions of Director.

When the instructions you've specified for the `mouseDownScript` property are no longer appropriate, turn them off by using the statement `set the mouseDownScript to empty`.

The `mouseDownScript` property can be tested and set. The default value is EMPTY, which means that the `mouseDownScript` property has no Lingo at all assigned to it.

Example 1:

This statement sets `the mouseDownScript` to `if the mouseDown then go to next`. When this is in effect and the user clicks the mouse button, the playback head always jumps to the next marker in the movie.

```
set the mouseDownScript ¬
   to "if the mouseDown then go to next"
```

Example 2:

This statement sets `the mouseDownScript` to `if the clickOn = 0 then beep`. When this in effect and the user clicks anywhere on the Stage, the computer beeps.

```
set the mouseDownScript ¬
   to "if the clickOn = 0 then beep"
```

Example 3:

This statement sets the `mouseDownScript to` the custom handler, *myCustomHandler*. A Lingo custom handler must be enclosed in quotation marks when used with the `mouseDownScript` property.

```
set the mouseDownScript to "myCustomHandler"
```

{button See also,AL(`Lingo_mouseDownScript')}

mouseH

Syntax: `the mouseH`

This function indicates the horizontal position of the mouse cursor. 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 cursor and checking whether the cursor is within a region of the Stage. Using `mouseH` and `mouseV` functions together, you can determine the cursor's exact location.

The `mouseH` function can be tested but not set.

Example 1:

This handler moves sprite 10 to the mouse cursor location and updates the Stage when the user clicks the mouse button:

```
on mouseDown
   set the locH of sprite 1 to the mouseH
   set the locV of sprite 1 to the mouseV
   updateStage
end
```

Example 2:

This statement tests whether the cursor is more than 10 pixels to the right or left of a starting point and sets the variable `Far` to TRUE if it is:

```
if abs(the mouseH - startH) > 10 then ¬
  set draggedEnough = TRUE
```

{button See also,AL(`Lingo_mouseH')}

mouseItem

Syntax: `the mouseItem`

This integer function gives the number of the item that is 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 commas.) 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` function can change in a handler or repeat loop. If a handler or repeat loop relies on the initial value of `the mouseItem` when the handler or repeat loop begins, call the function once and assign its value to a local variable.

Example 1:

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

   put "Please point to an item." ¬
      into member "Instructions"
```

Example 2:

This statement assigns the item under the cursor in the specified field to the variable `currentItem`:

```
set currentItem = item (the mouseItem) ¬
    of member (the mouseMember)
```

Example 3:

This handler highlights the item under the cursor when the mouse button is pressed:

```
on mouseDown
  set thisField to the member of sprite (the clickOn)
  if the mouseItem < 1 then exit
  set lastItem to 0
  repeat while the stillDown
    set MI to the mouseWord
    if MI < 1 then next repeat
    if MI <> lastWord then
      hilite item MI of field thisField
      set lastItem to MI
    end if
  end repeat
end
```

{button See also,AL(`Lingo_mouseItem')}

mouseLine

Syntax: `the mouseLine`

This integer function gives the number of the line under the pointer when the function is called and the cursor is over a field sprite. Counting starts at the beginning of the field. When the mouse is not over a field sprite, the result is -1.

The value of the `mouseLine` function can change in a handler or repeat loop. If a handler or repeat loop is using this function multiple times, it's usually a good idea to call the function once and assign its value to a local variable.

Example 1:

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 ¬
   put "Please point to a line." ¬
   into member "Instructions"
```

Example 2:

This statement assigns the number of the item under the cursor in the specified field to the variable `currentLine`:

```
set currentLine = line (the mouseLine) of ¬

  member (the mouseMember)
```

Example 3:

This handler highlights the line under the cursor when the mouse button is pressed:

```
on mouseDown
  set thisField to the member of sprite (the clickOn)
  if the mouseLine < 1 then exit
  set lastLine to 0
  repeat while the stillDown
    set ML to the mouseLine
    if ML < 1 then next repeat
    if ML <> lastLine then
      hilite word ML of field thisField
      set lastLine to ML
    end if
  end repeat
end
```

{button See also,AL(`Lingo_mouseLine')}

mouseMember

Syntax: `the mouseMember`

This function gives the cast member assigned to the sprite that is under the cursor when the function is called. When the cursor is not over a sprite, it gives the result `VOID`.

The `mouseMember` function replaces `mouseCast`, which was used in earlier versions of Director.

This function is useful for having the movie perform specific actions when the cursor rolls over a sprite and the sprite uses a certain cast member.

The value of the `mouseMember` function can change frequently. If you want to use this function multiple times in handler but want a consistent value, assign the `mouseMember` value to a local variable when the handler starts and use the
variable instead.

For casts other than cast 1, `the mouseMember` returns a value that does not distinguish between the cast member and the cast number. To obtain a result that distinguishes the cast member and the cast number, use the expression `member (the mouseMember)`. However, if the user doesn't click a sprite, this expression generates a script error.

Example 1:

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 is one example of how you can specify an action depending on which cast member is assigned to the sprite:

```
if the mouseMember = member "Off Limits"¬
then alert "Stay away from there!"
```

Example 2:

This statement assigns the cast member of the sprite under the cursor to the variable `lastMember`:

```
set lastMember = the mouseMember
```

{button See also,AL(`Lingo_mouseCast')}

mouseUp

Syntax: `the mouseUp`

This function indicates whether the mouse button is released.

- The `mouseUp` function is TRUE when the mouse button is released.
- The `mouseUp` function is FALSE when the mouse button is being pressed.

Example 1:

This handler causes the movie to beep as long as the mouse button is being pressed. The beep stops when the user clicks the mouse button:

```
on enterFrame
   repeat while the mouseUp = FALSE
      beep
   end repeat
end enterFrame
```

Example 2:

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
```

{button See also,AL(`Lingo_mouseUp')}

mouseUpScript

Syntax: `the mouseUpScript`

This property determines the Lingo that is executed when the mouse button is released. The Lingo is written as a string, surrounded by quotation marks. It 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 pass 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 does the same as using the `when mouseUp then` command that appeared in earlier versions of Director.

The `mouseUpScript` property can be tested and set. The default value is EMPTY.

Example 1:

This statement sets `the mouseUpScript` to "`go to the frame +1`". When this statement is in effect and the movie is paused, the movie always continues whenever the user releases the mouse button.

```
set the mouseUpScript to "go to the frame +1"
```

Example 2:

This statement has the movie beep when the user releases the mouse button after clicking anywhere on the Stage:

```
set the mouseUpScript ¬

   to "if the clickOn = 0 then beep"
```

Example 3:

This statement sets `the mouseUpScript` to the custom handler *myCustomHandler*. A Lingo custom handler must be enclosed in quotation marks when used with the `mouseUpScript` property.

```
set the mouseUpScript to "myCustomHandler"
```

{button See also,AL(`Lingo_mouseUpScript')}

mouseV

Syntax: `the mouseV`

This function indicates the vertical position of the mouse cursor. The value of `mouseV` is the number of pixels the cursor is from the top of the Stage.

The `mouseV` function 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` functions together, you can identify the cursor's exact location.

Example 1:

This handler moves sprite 1 to the mouse cursor location and updates the Stage when the user clicks the mouse button:

```
on mouseDown
   set the locH of sprite 1 to the mouseH
   set the locV of sprite 1 to the mouseV
   updateStage
end
```

Example 2:

This statement tests whether the cursor is more than 10 pixels above or below a starting point and sets the variable `vFar` to TRUE if it is:

```
if abs(the mouseV - startV) > 10 then ¬
   set draggedEnough = TRUE
```

{button See also,AL(`Lingo_mouseV')}

mouseWord

Syntax: `the mouseWord`

This integer function gives the number of the word under the cursor when the function is called and when the cursor 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` function can change in a handler or repeat loop. If a handler or repeat loop is using this function multiple times, it's usually a good idea to call the function once and assign its value to a local variable.

Example 1:

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 word." when it is not:

```
if the mouseWord = -1 then ¬

   put "Please point to a word." ¬

   into member "Instructions"
```

Example 2:

This statement assigns the number of the word under the cursor in the specified field to the variable `currentWord`:

```
set currentWord = word (the mouseWord) of ¬

 member (the mouseMember)
```

Example 3:

This handler highlights the word under the cursor when the mouse button is pressed:

```
on mouseDown
  set thisField to the member of sprite (the clickOn)
  if the mouseWord < 1 then exit
  set lastWord to 0
  repeat while the stillDown
    set MW to the mouseWord
    if MW < 1 then next repeat
    if MW <> lastWord then
      hilite word MW of field thisField
      set lastWord to MW
    end if
  end repeat
end
```

{button See also,AL(`Lingo_mouseWord')}

move cast

This is obsolete. Use **move member** instead.

move member

Syntax: `move member` *whichCastmember* `[, member` *whichLocation*`]`

This command moves the cast member specified by *whichCastmember* to a different location in the Cast window.

- Using the `move member` command without the optional parameter moves the cast member to the first empty location in the Cast window.
- Including the `member` *whichLocation* parameter in the `move member` command moves the cast member to the location specified by *whichLocation*.

Example 1:

This statement moves cast member Shrine to the first empty location in the Cast window:

```
move member "Shrine"
```

Example 2:

This statement moves cast member Shrine to location 20 in the Bitmaps Cast window:

```
move member "Shrine", member 20 of castLib "Bitmaps"
```

moveableSprite of sprite

Syntax: `the moveableSprite of sprite` *whichSprite*

This sprite property indicates whether a sprite is moveable.

- When the sprite can be moved by the user, `the moveableSprite of sprite` is TRUE.
- When the sprite cannot be moved by the user, `the moveableSprite of sprite` is FALSE.

You can also make a sprite moveable by using the Moveable option in the Score. However, controlling whether a sprite is moveable by using Lingo lets you turn this condition on and off as situations in the movie require. For example, referring to the "Mechanical Simulation" sample movie, you can allow the user to drag parts from the toolkit but make them unmoveable after they are on the pegboard by turning `moveableSprite of sprite` on and off at the appropriate times.

Setting `the moveableSprite of sprite` property lets you control whether sprites are moveable from other scripts.

The `moveableSprite of sprite` property can be tested and set.

Example 1:

This handler first makes sprite channel 5 a puppet and then makes sprites in that channel moveable:

```
on spriteMove
   puppetSprite 5, TRUE
   set the moveableSprite of sprite 5 to TRUE
end
```

Example 2:

This statement checks whether a sprite is moveable and displays a message if it isn't:

```
if the moveableSprite of sprite 13 = FALSE ¬
   then set the text of member "Notice" to ¬
   "You can't drag this item by using the mouse."
```

{button See also,AL(`Lingo_moveableSprite_of_sprite')}

moveToBack

Syntax: `moveToBack window "`*whichWindow*`"`

This command moves the window specified by *whichWindow* behind all other windows.

Example:

These statements move the first window in `the windowList` behind all other windows:

```
set myWind=getat(the windowList, 1)
moveToBack myWind
```

If the first record of `the windowList` was Demo Window, the long version of the `moveToBack` would be:

```
moveToBack window "Demo Window"
```

moveToFront

Syntax: `moveToFront window` *"whichWindow"*

This command moves the window specified by *whichWindow* in front of all other windows.

Example:

This statement moves the first window in `the windowList` in front of all other windows:

```
set myWind = getAt(the windowList, 1)
moveToFront myWind
```

If the first record of `the windowList` was Demo Window, the long version of `moveToFront` would be:

```
moveToFront window "Demo Window"
```

movie

Syntax: `the movie`

This string function returns the name of the currently open movie.

Example:

This statement assigns the name of the current movie to the field cast member Current Movie:

```
put the movie into member "Current Movie"
```

{button See also,AL(`Lingo_movie')}

movieFileFreeSize

Syntax: `the movieFileFreeSize`

This function returns the number of unused bytes in the current movie.

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`

This function returns the number of bytes in the current movie.

Example:

This statement displays the number of bytes in the current movie:

```
put the movieFileSize
```

movieName

Syntax: `the movieName`

This function indicates the simple name of the current movie. The `movieName` function is equivalent to the `movie` function.

Example:

This statement displays the name of the current movie in the Message window:

```
put the movieName
```

{button See also,AL(`Lingo_movieName')}

moviePath

Syntax: `the moviePath`

This function indicates the pathname of the folder that the current movie is located in. The `moviePath` function is equivalent to the `pathName` function.

For pathnames that work on both Windows and Macintosh computers, use the `@` pathname operator.

Example 1:

This statement displays the pathname of the current movie's folder:

```
put the moviePath
```

Example 2:

This statement plays a sound file `crash.aif` stored in the Sounds subfolder of the current movie's folder:

```
sound playFile 1, the moviePath&"Sounds/crash.aif"
```

Note: If you choose to specify a subfolder location, as in this example, using "/" will ensure that the path is understood on both Macintosh and Windows computers. Only on a Macintosh can you use ":" to separate subfolders.

{button See also,AL(`Lingo_moviePath')}

movieRate of sprite

Syntax: `the movieRate of sprite` *whichSprite*

This 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 is normal forward play, -1 is reverse, and 0 is 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 of sprite` exceeds 1. The severity of dropping frames depends on factors such as the performance of the computer the movie is playing on, whether the digital video sprite is stretched, and so on.

This property can be tested and set.

Example:

This statement sets the rate for a digital video in sprite channel 9 to normal playback speed:

```
set the movieRate of sprite 9 to 1
```

This statement causes the digital video in sprite channel 9 to play in reverse:

```
set the movieRate of sprite 9 to -1
```

{button See also,AL(`Lingo_movieRate_of_sprite')}

movieTime of sprite

Syntax: `the movieTime of sprite` *whichSprite*

This sprite property determines the current time of a digital video movie playing in the channel specified by *channelNumber*. The value of `the movieTime` is measured in ticks.

The `movieTime of sprite` property can be tested and set.

Example 1:

This statement displays the current time of the QuickTime movie in channel 9 in the Message window:

```
put the movieTime of sprite 9
```

Example 2:

This statement sets the current time of the QuickTime movie in channel 9 to the value in the variable `Poster`:

```
set the movieTime of sprite 9 to Poster
```

{button See also,AL(`Lingo_movieTime_of_sprite')}

mPerform

This method is now obsolete. It was used in earlier versions of Director to send an arbitrary message to any Lingo object.

mPut

This method is now obsolete. It was used in earlier versions of Director to put data into an object's internal array.

mRespondsTo

This method is now obsolete. It was used in earlier versions of Director to determine whether an XObject could respond to a specific message.

multiSound

Syntax: `the multiSound`

This system property is TRUE when the system supports more than one sound channel. A Windows computer must have a multichannel sound card for the `multiSound` property to be TRUE.

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"
```

name of cast

This is obsolete. Use **name of member** instead.

name of castLib

Syntax: `the name of castLib` *whichCast*

This cast member property returns the name of the specified cast. This property can be tested and set.

Example:

This statement changes the name of the cast Buttons to Interface:

```
set the name of castLib "Buttons" to "Interface"
```

name of member

Syntax: `the name of member` *whichCastmember*

This cast member property determines the name of the specified cast member.

- When *whichCastmember* is a string, it is used as the cast name.
- When *whichCastmember* is an integer, it is used as the cast number.

The name is a descriptive string assigned by the user. Setting this property is equivalent to entering a name in the **Cast Member Properties** dialog box.

The `name` cast member property can be tested and set.

Example 1:

This statement changes the name of the cast member named On to Off:

```
set the name of member "On" to "Off"
```

Example 2:

This statement sets the name of cast member 15 to Background Sound:

```
set the name of member 15 to "Background Sound"
```

Example 3:

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`:

```
set itsName = the name of member (i + 1)
```

{button See also,AL(`Lingo_name_of_member')}

name of menu

Syntax: `the name of menu` *whichMenu*

This menu property returns a string containing the name of the specified menu. The expression *whichMenu* can evaluate to either a menu number or a menu name.

The `name of menu` property can be tested but cannot be set directly. Use the `installMenu` command to set up a custom menu bar.

For more information about user interfaces, see Chapter 9, "Creating User Interface Controls," in *Learning Lingo*.

Example 1:

This statement assigns the name of menu number 1 to the variable `firstMenu`:

```
set firstMenu = the name of menu 1
```

Example 2:

The following handler returns a list of menu names, one per line:

```
on menuList
   put EMPTY into list
   repeat with i = 1 to the number of menus
      put the name of menu i & RETURN after list
   end repeat
   return list
end menuList
```

{button See also,AL(`Lingo_name_of_menu')}

name of menuItem

Syntax: `the name of menuItem` *whichItem* `of menu` *whichMenu*

This menu item property determines the text that appears in the menu item specified by *whichItem* in the menu specified by *whichMenu*. The *whichItem* expression can be either a menu item name or a menu item number; *whichMenu* can be either a menu name or a menu number.

The `name of menuItem` property can be tested and set.

For more information about user interfaces, see Chapter 9, "Creating User Interface Controls," in *Learning Lingo*.

Example 1:

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 2:

This statement causes a specific file name to follow the term Open in the File menu:

```
set the name of menuItem "Open" of menu fileMenu ¬
   to "Open" & fileName
```

{button See also,AL(`Lingo_name_of_menuItem')}

name of window

Syntax: `the name of window "whichWindow"`

This property determines the name of the specified window in `the windowList`. (The `title of window` property determines the title that appears in a window's title bar.)

It can be tested and set.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

Example:

This statement changes the name of window Yesterday to Today:

```
set the name of window "Yesterday to "Today"
```

name of xtra

Syntax: `the name of xtra whichXtra.`

This property indicates the name of the specified Xtra. It can be tested and set.

Example:

The following statement changes the name of the Xtra Editor to Text Whiz:

```
set the name of xtra "Editor" to "Text Whiz"
```

{button See also,AL(`Lingo_name_of_xtra')}

netAbort

Syntax: `netAbort(URL)`

or

`netAbort(netID)`

This command cancels a network operation without waiting for a result.

If it is available, using a network ID is the most efficient way to stop a network operation.

In some cases, when a network ID is not available, you can use a URL to stop the transmission of data for that URL. If the data transmission is complete, this command will have no effect.

For more information about Shockwave movies and the internet, see Chapter 14, "Shockwave, the Internet, and Lingo," in *Learning Lingo*.

Example:

This statement passes a network ID to `netAbort` to cancel a particular network operation.

```
on mouseUp
  netAbort(myNetID)
end
```

{button See also,AL(`Lingo_netAbort')}

netDone

Syntax: `netDone()`

or

`netDone(netID)`

This function indicates whether `getNetText, preloadNetThing, gotoNetMovie, gotoNetPage,` or `putNetText` are finished.

- Use `netDone()` to test the last network operation.
- Use `netDone(netID)` to test the net operation identified by *netID*.

The `netDone` function returns TRUE after a background loading operation is finished or when the operation is terminated by a browser error. The default value is TRUE.

The `netDone` function returns FALSE after a background loading operation is started and is in progress.

For more information about Shockwave movies and the internet, see Chapter 14, "Shockwave, the Internet, and Lingo," in *Learning Lingo*.

Example 1:

This handler uses the `netDone` function to test whether the last network operation is finished. If the operation is finished, text returned by `netTextResult` is displayed in the field cast member Display Text:

```
on exitFrame
  if netDone() = TRUE then
    put netTextResult() into member 'Display Text'
  end if
end
```

Example 2:

This handler uses a specific network ID as an argument for `netDone` to check on 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
```

{button See also,AL(`Lingo_netDone')}

netError

Syntax: `netError()`

or

`netError(netID)`

This function determines whether an error has occurred in a network operation.

- Use `netError()` to test the last network operation.
- Use `netError(netID)` to test the network operation specified by *netID*.

If the operation completed successfully, the function returns `OK`.

If the operation failed, the function returns an error number.

If no background loading operation has started, the function returns an empty string.

For more information about Shockwave movies and the internet, see Chapter 14, "Shockwave, the Internet, and Lingo," in *Learning Lingo*.

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
```

{button See also,AL(`Lingo_netError')}

netLastModDate

Syntax: `netLastModDate()`

This function returns the "date last modified" string from the HTTP header for the specified item. The string's format is "Thu, Jan 30, 1997 12:00:00 AM GMT"

The `netLastModDate` function can be called only after `netDone` or `netError` reports that the operation is complete and before the next operation starts. After the next operation starts, the Director movie or projector discards the results of the previous operation to conserve memory.

For more information about Shockwave movies and the internet, see Chapter 14, "Shockwave, the Internet, and Lingo," in *Learning Lingo*.

Example:

These statements check the date of a file downloaded from the internet:

```
if netDone() then
  set theDate = netLastModDate()
  if char 6 to 11 of theDate <> "Jan 30" then
    alert "The file is outdated."
  end if
end if
```

{button See also,AL(`Lingo_netLastModDate')}

netMIME

Syntax: `netMIME()`

This function provides the MIME type of the internet file that the last network operation returned (the most recently downloaded ftp or HTTP item).

The `netMIME` function can be called only after the `netDone` or `netError` reports that the operation is complete and before the next network operation starts. After the next operation starts, the Director movie or projector discards the results of the previous operation to conserve memory.

For more information about Shockwave movies and the internet, see Chapter 14, "Shockwave, the Internet, and Lingo," in *Learning Lingo*.

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.¬
              Mime type " & myMimeType & " is unsupported."
    end case
  else
    go the frame
  end if
end
```

{button See also,AL(`Lingo_netMIME')}

netPresent

Syntax: `netPresent()`

This property determines whether the Xtras needed for accessing the internet are available.

Example:

This statement sends an alert if the Xtras are not available:

```
if netPresent() = FALSE then
  alert
end if
```

{button See also,AL(`Lingo_netPresent_xtra')}

netStatus

Syntax: `netStatus` *`msgString`*

This command displays the specified string in the status area of the browser window.

In the authoring environment, `netStatus` displays the string in the Message window.

The `netStatus` command doesn't work in projectors or with the Director Shockwave ActiveX control in Internet Explorer 3.0.

For more information about Shockwave movies and the internet, see Chapter 14, "Shockwave, the Internet, and Lingo," in *Learning Lingo*.

Example:

```
on exitFrame

  netStatus "This is a test"

end
```

netTextResult

Syntax: `netTextResult()`

or

`netTextResult(netID)`

This function returns the text obtained by the last network operation. If the last network operation was `getNetText`, the text is the text of the file on the network.

The `netTextResult` function can be called only after `netDone` or `netError` indicates that the operation is complete and before the next operation starts.

After the next operation starts, Director discards the results of the previous operation to conserve memory.

For more information about Shockwave movies and the internet, see Chapter 14, "Shockwave, the Internet, and Lingo," in *Learning Lingo*.

Example:

This handler uses the `netDone` function to test whether the last network operation is finished. If the operation is finished, text returned by `netTextResult` is displayed in the field cast member Display Text:

```
on exitFrame
  if netDone() = TRUE then
    put netTextResult() into member "Display Text"
  end if
end
```

{button See also,AL(`Lingo_netTextResult')}

new

Syntax:

```
new(type)
```

or

```
new(type, castLib whichCast)
```

or

```
new(type, member whichCastMember of castLib whichCast)
```

or

```
set x = new(parentScript arg1, arg2, ...)
```

or

```
new(script parentScriptName , value1, value2 , ...)
```

or

```
new(xtra "xtraName")
```

This function creates a new cast member, child object, or Xtra.

For cast members, the parameter type 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.

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 conditions.

The child object has all the handlers of the parent script. The child object 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 the child object is a value, it can be assigned to variables, placed in lists, and passed as a parameter.

Being able to assign individual property values to child objects is the primary advantage of using `on new` handlers.

You display information about a child object by using the `put` command to display information about it in the Message window.

For more information about parent scripts and child objects, see Chapter 12, "Parent Scripts and Child Objects," in *Learning Lingo*.

For more information about authoring from Lingo, see Chapter 13, "Authoring from Lingo," in *Learning Lingo*.

Example 1:

This handler creates a new bitmap cast member and assigns it to the variable `Background`:

```
on makeBitmap
   set Background = new(#bitmap)
end makeBitmap
```

Example 2:

These statements use an `on new` handler to create a child object of a parent script. The parent script is a script cast member named Bird, which contains these handlers:

```
on new me
  return me
end

on fly me
  put "I am flying"
end fly
```

Example 3:

These statements create a child object called `myBird`, and make it fly by calling the fly handler in the Bird parent script:

```
set myBird to new(script "Bird")
fly myBird
```

Example 4:

This statement uses a new Bird parent script, which contains the property variable `speed`:

```
property speed
on new me, initSpeed
   set speed to initSpeed
   return me
end
on fly me
   put "I am flying at " & speed & "mph"
end
```

Example 5:

The following statements create two child objects called `myBird1` and `myBird2`. When the fly handler is called from the child object, the speed of the object is displayed in the Message window:

```
set myBird1 to new (script "Bird", 15)
set myBird2 to new(script "Bird", 25)
fly myBird1
fly myBird2
```

This message appears in the Message window:

```
-- "I am flying at 15 mph"
-- "I am flying at 25 mph"
```

{button See also,AL(`Lingo_new')}

next

Syntax: `next`

This keyword refers to the next marker in the movie. The `next` keyword is equivalent to the phrase `the marker (+ 1)`.

For more information about navigation, see Chapter 3, "Navigation," in *Learning Lingo*.

Example 1:

This statement sends the playback head to the next marker in the movie:

```
go next
```

Example 2:

This handler has the movie move to the next marker in the Score when the right arrow key is pressed and 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
```

{button See also,AL(`Lingo_next')}

next repeat

Syntax: `next repeat`

This keyword causes Lingo to go to the next step in a repeat loop in a script. This is different from the `exit repeat` keyword.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

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
```

not

Syntax: `not` *logicalExpression*

This logical operator performs a logical negation on a logical expression.

- When the expression specified by *logicalExpression* is TRUE, the result is FALSE.
- When the expression specified by *logicalExpression* is FALSE, the result is TRUE.

This logical operator has a precedence level of 5.

Example 1:

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 2:

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 3:

This handler sets `the checkMark of menuItem` for the item Bold in the Style menu to the opposite of its current setting:

```
on resetMenuItem
   set the checkMark of menuItem "Bold" ¬
      of menu "Style" to not (the checkMark ¬
      of menuItem "Bold" of menu "Style")
end resetMenuItem
```

{button See also,AL(`Lingo_not')}

nothing

Syntax: `nothing`

This command does nothing at all. It is useful for making the logic of an `if...then` statement more obvious. Also, a nested `if...then...else` statement that contains no explicit command for the `else` clause may require `else nothing`. Otherwise, Lingo interprets the `else` clause as part of the preceding `if` clause.

Example 1:

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 the moveable of sprite 1 = TRUE ¬
      then set the text of member "Notice" = ¬
      "Drag the ball"
      else nothing
   else set the text of member "Notice" = ¬
      "Click again"
   end if
end mouseDown
```

Example 2:

This handler instructs the movie to do nothing as long as the mouse button is being pressed:

```
on mouseDown
   repeat while the stillDown
      nothing
   end repeat
end mouseDown
```

{button See also,AL(`Lingo_nothing')}

number of cast

This is obsolete. Use **number of member** instead.

number of castLib

Syntax: `the number of castLib` *whichCast*

This cast member property indicates the number of the specified cast. For example, 2 is `the number of castLib` for Cast 2. The 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 the name of castLib n &&"contains"&&the ¬
   number of members of castLib n&&"cast members."
end repeat
```

number of castLibs

Syntax: `the number of castLibs`

This cast member 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 the name of castLib n &&"contains"&&the ¬
   number of members of castLib n&&"cast members."
end repeat
```

number of castmembers

This is obsolete. Use **number of members** instead.

number of chars in

Syntax: `the number of chars in` *chunkExpression*

This chunk function returns a count of the characters in a chunk expression.

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.

Spaces and control characters such as Tab and Return count as characters.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example 1:

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 2:

This statement sets the variable `charCounter` to the number of characters in the word i located in the string Names:

```
set charCounter to the number of chars in word i ¬
   of member "Names"
```

{button See also,AL(`Lingo_number_of_chars_in')}

number of items in

Syntax: `the number of items in` *chunkExpression*

This chunk function 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.

Example 1:

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 2:

This statement sets the variable `itemCounter` to the number of items in the field Names:

```
set itemCounter = the number of items in field "Names"
```

{button See also,AL(`Lingo_number_of_items_in')}

number of lines in

Syntax: `the number of lines in` *chunkExpression*

This chunk function 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.

Example 1:

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 2:

This statement sets the variable `lineCounter` to the number of lines in the field Names:

```
set lineCounter to the number of lines in member "Names"
```

{button See also,AL(`Lingo_number_of_lines_in')}

number of member

Syntax: `the number of member` *whichCastmember*

This cast member property indicates the cast number of the cast member specified by *whichCastmember*.

- When *whichCastmember* is a string, the string is used as the cast member name.
- When *whichCastmember* is an integer, the integer is used as the cast member number.

The `number of member` property can be tested but not set.

Example 1:

This statement assigns the cast number of the cast member Power Switch to the variable `whichCastmember`:

```
put the number of member "Power Switch" into ¬
   whichCastmember
```

Example 2:

This statement assigns the cast member Red Balloon to sprite 1:

```
set the memberNum of sprite 1 ¬
   to the number of member "Red Balloon"
```

{button See also,AL(`Lingo_number_of_member')}

number of members

Syntax: `the number of members`

This property indicates the number of the last cast member in the current movie. Any empty cast slots are also counted, so the actual number of cast members may be fewer than the `number of members` value.

The `number of members` property can be tested but not set.

Example:

The following handler returns a string containing a list of all the cast member names, one per line:

```
on castList whichCast
   put EMPTY into list
   repeat with i = 1 to the number of members¬
   of castLib whichCast
      put the name of member I of castLib whichCast¬
      & RETURN after list
   end repeat
   return list
end castList
```

{button See also,AL(`Lingo_number_of_members')}

number of members of castLib

Syntax: `the number of members of castLib` *whichCast*

This cast member property indicates the number of the last cast member in the specified cast. This property can be tested but not set.

Example:

These statements use the Message window to display 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.

```
i = 0
repeat i to the number of members ¬
of castLib "Central Casting"
   put "Cast member"&&i&&"is a"&&¬
   (the type of member I of ¬
   castLib "Central Casting" &&"cast member."
end repeat
```

number of menuItems

Syntax: `the number of menuItems of menu` *whichMenu*

This menu property indicates the number of menu items in the custom menu specified by *whichMenu*. The *whichMenu* parameter can be a menu name or a menu number.

The `number of menuItems` menu property can be tested but not set directly. Use the `installMenu` command to set up a custom menu bar.

For more information about user interfaces, see Chapter 9, "Creating User Interface Controls," in *Learning Lingo*.

Example 1:

This statement sets the variable `fileItems` to the number of menu items in the custom File menu:

```
set fileItems = the number of menuItems of menu "File"
```

Example 2:

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`:

```
set itemCount = the number of menuItems of menu i
```

{button See also,AL(`Lingo_number_of_menuItems')}

number of menus

Syntax: `the number of menus`

This menu property indicates the number of menus installed in the current movie.

The `number of menus` menu property can be tested but not set. Use the `installMenu` command to set up a custom menu bar.

For more information about user interfaces, see Chapter 9, "Creating User Interface Controls," in *Learning Lingo*.

Example 1:

This statement determines whether any custom menus are installed in the movie and installs the menu Menubar if no menus are already installed:

```
if the number of menus = 0 then ¬
   installMenu (the number of member "Menubar")
```

Example 2:

This statement causes the Message window to display the number of menus that are in the current movie:

```
put the number of menus
```

{button See also,AL(`Lingo_number_of_menus')}

number of words in

`the number of words in` *chunkExpression*

This function tells how many words are in the chunk expression specified by *chunkExpression*.

Chunk expressions refer to 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.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example 1:

This statement has the Message window display 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 2:

This handler reverses the order of words in the string specified by the argument `wordList`:

```
on reverse wordList
   put EMPTY into list
   repeat with i = 1 to the number of words ¬
      in wordList
      put word i of wordList & " " before list
   end repeat
   delete char (the number of chars in list) of list
   return list
end reverse wordList
```

{button See also,AL(`Lingo_number_of_words_in')}

number of xtras

Syntax: `the number of xtras`

This property returns the number of Lingo Xtras available to the movie. These Xtras might have been opened by the `openxlib` command or might be present in the standard Xtras folder. This property can be tested but not set.

Example:

This statement causes the Message window to display the number of Lingo Xtras that are available to the movie:

```
put the number of xtras
```

{button See also,AL(`Lingo_number_of_xtras')}

numChannels of member

Syntax: `the numChannels of member "`*whichCastmember*`"`

This Shockwave Audio (SWA) property returns the number of channels within the specified SWA streaming cast member. Allowable values are 1 for mono and 2 for stereo.

This property is only available after the SWA streaming cast member is 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:

```
set myVariable to the numChannels of member "Duke Ellington"
if myVariable = 1 then
   set the text of member "Channel Display" = "Mono"
else
   set the text of member "Channel Display" = "Stereo"
end if
```

numToChar

Syntax: `numToChar(`*integerExpression*`)`

This function displays a string containing the single character whose ASCII sequence number is the value of *integerExpression*. It 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 can refer to different keys on different computers.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example 1:

This statement causes the Message window to display the character whose ASCII number is 65:

```
put numToChar(65)
```

The result is the letter "A."

Example 2:

This handler takes any arbitrary string, removes any non-alphabetic characters, and returns only capital letters:

```
on ForceUppercase input
  set output to EMPTY
  set num to length(input)
  repeat with i = 1 to num
    set theASCII to charToNum(char 1 of input)
    if theASCII = min(max(96, theASCII), 123) ¬
    then set theASCII to theASCII - 32
    if theASCII = min(max(63, theASCII), 91) then ¬
    put numToChar(theASCII) after output
    delete char 1 of input
  end repeat
  return output
end ForceUppercase
```

{button See also,AL(`Lingo_numToChar')}

objectP

Syntax: `objectP(`*expression*`)`

This function indicates whether the expression specified by *expression* is an object produced by a parent script, Xtra, or **XObject**.

- When `objectP` is TRUE, the expression is such an object.
- When `objectP` is FALSE, the expression is not such an object.

The "P" in `objectP` stands for "predicate."

It is good practice to use `objectP` to determine which items are **XObjects** when you create **XObjects** by using `mNew` or disposing of **XObjects** by using `mDispose`.

Example 1:

This statement checks whether `modemPort` is an **XObject** and displays the result in the Message window:

```
put objectP(modemPort)
```

Example 2:

This handler checks whether `externalFile` is an **XObject** and disposes of it if it is:

```
on stopMovie

   if objectP(externalFile) then ¬

      externalFile(mDispose)

end stopMovie
```

{button See also,AL(`Lingo_objectP')}

of

The word `of` is part of many Lingo properties, such as the `foreColor of sprite`, `the number of member`, `the name of menu`, and so on.

offset

Syntax: `offset(`*stringExpression1, stringExpression2)*

This function returns the first place that the first character of *stringExpression1* occurs in *stringExpression2*.

- When *stringExpression1* is found in *stringExpression2*, the result is the number that indicates the position of the first occurrence.
- When *stringExpression1* is not found in *stringExpression2*, the result is 0.

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 "□" the same character on the Macintosh.

Example 1:

This statement causes the Message window to display the beginning position of the string "media" within the string "Macromedia":

```
put offset("media","Macromedia")
```

The result is 6.

Example 2:

This statement causes the Message window to display 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 3:

This handler replaces one string with another. It doesn't check for upper- or lowercase letters.

```
on SearchAndReplace input, oldString, newString
set output to ""
  repeat while input contains oldString
    set position to offset(oldString, input) - 1
    put char 1 to position of input after output
    put newString after output
    delete char 1 to (position ¬
    + length(oldString)) of input
  end repeat
  put input after output
  return output
end SearchAndReplace
```

{button See also,AL(`Lingo_offset')}

offset rect

Syntax: `offset (`*rectangle, horizontalChange, verticalChange)*

This 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 zero, the offset is toward the right of the Stage; when *horizontalChange* is less than zero, the offset is toward the left of the Stage.
- When *verticalChange* is greater than zero, the offset is toward the top of the Stage; when *verticalChange* is less than zero, the offset is toward the bottom of the Stage.

The values for *verticalChange* and *horizontalChange* are in pixels.

on

Syntax:

```
on handlerName [argument1] [, arg2] [, arg3] ...
    [statements]
end handlerName
```

This keyword indicates the beginning of a handler. Handlers are collections of Lingo statements that you can execute by simply using the handler name. A handler can accept arguments as input values and return a value as a function result.

Handlers can be defined in Score scripts, movie scripts, and scripts of cast members. A handler in a script of a cast member can only be called by other handlers in the same script. A handler in a Score script or 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 the shared cast.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

{button See also,AL(`Lingo_on')}

on activateWindow

Syntax: `on activateWindow`

statement(s)

`end`

This event handler contains statements that run when the movie is running as a movie in a window and the window becomes active and comes to the foreground when the user clicks the inactive window.

An `on activateWindow` handler is a good place for Lingo that you want executed every time the movie becomes active. Place such a handler in a movie script.

Clicking the main movie (the main Stage) does not generate an `on activateWindow` handler.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

Example:

This handler plays the sound file Hurray when the window that the movie is playing in becomes active:

```
on activateWindow
   puppetSound 2, "Hurray"
end
```

{button See also,AL(`Lingo_on_activateWindow')}

on alertHook

Syntax: `on alertHook me` *err, msg*

return *value*

`end`

This event handler contains statements that determine whether a projector displays an error alert when an error occurs.

me refers to the parent script instance

err describes the source of the error. Possible values are:
"File read error"
"File error"
"Script syntax error"
"Script runtime error"

msg specifies whether an alert appears when an error occurs.
When value is 0, an error alert appears
When value is 1, no error alert appears and the projector continues playing, if possible.

The `on alertHook` handler must be placed in a parent script.

Example:

This handler has the projector display no error alert when an error occurs:

```
on alertHook me, err, msg
  return 1
end
```

on beginSprite

Syntax: `on beginSprite`

statement(s)

`end`

This event handler contains statements that run when the playback head moves to a frame that has a sprite that was not previously encountered. The event is generated before `prepareFrame`.

Director creates instances of any behavior scripts attached to the sprite when the `beginSprite` message is sent.

This event is passed the sprite script reference `me` if used in a behavior. The message is sent to sprite scripts and frame scripts.

If a sprite begins in the first frame that plays in a movie, the `beginSprite` message is sent after the `prepareMovie` message but before the `prepareFrame` and `startMovie` messages.

Example:

This handler plays the sound cast member Stevie Wonder when the sprite begins:

```
on beginSprite me
  puppetSound "Stevie Wonder"
  updateStage
end
```

on closeWindow

Syntax:

```
on closeWindow
    statement(s)
end
```

This event handler contains statements that run when the user closes the window for a movie in a window by clicking the window's close box.

The `on closeWindow` handler is a good place to put Lingo that you want executed every time the movie's window closes.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

Example:

This handler has Director forget the current window when the window that the movie is playing in closes:

```
on closeWindow
   -- perform general housekeeping here
   forget the activeWindow
end
```

on cuePassed

Syntax:

```
on cuePassed(channelID, cuePointNumber, cuePointName)
    statement(s)
end
```

or

```
on cuePassed(me, channelID, cuePointNumber, cuePointName)
    statement(s)
end
```

This event handler contains statements that run each time a sound or sprite passes a cue point in its media.

- The optional `me` parameter is the `scriptInstanceRef` of the script being invoked.
- *channelID* is either the number of the sound or sprite channel for the file that the cue point occurred in. 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.
- *cuePointNumber* is the ordinal number of the cue point that triggers the event in the list of the cast member's cue points.
- *cuePointName* is the name of the cue point that was encountered.

The message is passed - in order - to sprite, cast member, frame, and movie scripts.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement 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
```

{button See also,AL(`Lingo_on_cuePassed')}

on deactivateWindow

Syntax:

```
on deactivateWindow
    statement(s)
end
```

This 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.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

Example:

This handler plays the sound file Snore when the window that the movie is playing in is deactivated:

```
on deactivateWindow
   puppetSound 2, "Snore"
end
```

on endSprite

Syntax: `on endSprite`

statement(s)

`end`

This 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 the Score script that a behavior is in.

Director destroys instances of any behavior scripts attached to the sprite when the `endSprite` event occurs.

The event handler is passed the sprite script or frame script reference `me` if used in a behavior.

The `endSprite` message is sent after the `exitFrame` message if the playback head plays to the end of the frame.

Example:

This handler runs when the playback head exits a sprite:

```
on endSprite me
  -- clean up
  set gNumberOfSharks = gNumberOfSharks - 1
  puppetSound(5,0)
end
```

{button See also,AL(`Lingo_on_endsprite')}

on enterFrame

Syntax: `on enterFrame`

statement(s)

`end enterFrame`

This event handler contains statements that run each time the playback head enters the frame that the `on enterFrame` handler is attached to. The `on enterFrame` handler is equivalent to the `on stepMovie` handler used in earlier versions of Director.

Place `on enterFrame` handlers in sprite, frame, or movie scripts.

- To assign the handler to an individual sprite, put the handler in the sprite script.
- 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 then `exitFrame`.

Note: This event is passed the sprite script or frame script reference `me` if used in a behavior.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

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
```

{button See also,AL(`Lingo_on_enterFrame')}

on EvalScript

Syntax:

```
on EvalScript
    statement(s)
end
```

This event handler in a Shockwave movie contains statements that run when the handler receives an `EvalScript` message from a browser.

The `EvalScript` message can include a string that Director can interpret as a Lingo statement. Lingo cannot accept nested strings Therefore, 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 a JavaScript or VBScript in a browser.

Expose only those behaviors through `on EvalScript` that you want people to be able to control. For security reasons, the movie author should not give complete access to the outside.

Although Lingo is not case-sensitive regarding the term `EvalScript`, it is best to match the capitalization used in JavaScript and use an initial uppercase letter "E" when writing `EvalScript`.

Example 1:

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 2:

This handler takes an argument that could be a number or symbol:

```
on EvalScript aParam
    if word 1 of aParam = "myHandler" then
        do aParam
    end if
end
```

Example 3:

This handler would normally require 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 sring(aParam)
end
```

on exitFrame

Syntax: `on exitFrame`

statement(s)

`end exitFrame`

This 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 sprite, frame, or movie scripts.

- When you want to assign the handler to an individual sprite, put the handler in the sprite script.
- When you want to assign the handler to an individual frame, put the handler in the frame script.
- When you want 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.

Note: This event is passed the sprite script or frame script reference `me` if used in a behavior.

- The order of frame events is `prepareFrame, enterFrame`, and then `exitFrame`.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

Example 1:

This handler turns off all puppet conditions when the playback head exits the frame:

```
on exitFrame me
   repeat with i = 48 down to 1
      set the puppet of sprite i = FALSE
   end repeat
end exitFrame
```

Example 2:

This handler sends the playback head to a specified frame if the value in the variable `vTotal` exceeds 1000 when the playback head exits the frame:

```
on exitFrame
   if vTotal > 1000 then go to frame "Finished"
end
```

{button See also,AL(`Lingo_on_exitFrame')}

on getBehaviorDescription

Syntax:

```
on getBehaviorDescription me
    statements
end
```

This 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 the score script that defines the behavior.

Example:

This statement displays "Vertical MultiLine textField Scrollbar" in the description pane.

```
on getBehaviorDescription me
   return "Vertical Multiline textField Scrollbar"
end
```

{button See also,AL(`Lingo_on_getBehaviorDescription')}

on getPropertyDescriptionList

Syntax: `on getPropertyDescriptionList me`

This 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 the Score script that defines the behavior. Behaviors that don't contain an `on getPropertyDesdriptionList` 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 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. Possible values for these settings are:

`#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 #cursor #graphic`
`#comment`	A descriptive string that appears to the left of the parameter's editable field in the Parameters dialog box.

For more information, see Chapter 15, "Authoring Behaviors," in *Learning Lingo*.

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 me

   set 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:#boolean,#commet:"Proportional:"]

   return description

end
```

{button See also,AL(`Lingo_getPropertyDescriptionList')}

on idle

Syntax:

```
on idle
    statement(s)
end idle
```

This event handler contains statements that run whenever the movie has no other events to handle.

This is a useful location for Lingo statements that you want to execute as frequently as possible. Some common cases are updating values in global variables and displays that tell 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. This makes it easier to turn off the `on idle` handler 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.

Cast members that were loaded during an `idle` event remain compressed until the movie uses them. When the movie plays back, it may have noticeable pauses while it decompresses the cast members.

The idle message is sent to frame scripts and movie scripts.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

Example:

This handler updates the time being displayed in the movie whenever there are no other events to handle:

```
on idle
   put the short time into member "Time"
end idle
```

{button See also,AL(`Lingo_on_idle')}

on keyDown

Syntax:

```
on keyDown
    statement(s)
end
```

This event handler contains statements that run when a key is pressed.

When a key is pressed, Lingo searches these locations, in order, for an `on keyDown` handler: primary event handler, editable field sprite script, script of a field cast member, frame script, and movie script. For sprites and cast members, `on keyDown` handlers work only for editable strings. A `keyDown` on a different type of cast member, such as a bitmap, has no effect. When pressing a key should always have the same response throughout the movie, set `the keyDownScript`.

Lingo 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 have occur when the user presses keys.

Where you place an `on keyDown` handler can affect when it runs.

- When you want the handler to apply to a specific editable field sprite, put the handler in a sprite script.
- When you want the handler to apply to an editable field cast member in general, put the handler in a script of the cast member.
- When you want the handler to apply to an entire frame, put the handler in a frame script.
- When you want the handler to apply throughout the entire movie, put the handler in a movie script.

You can override an `on keyDown` handler by placing an alternate `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.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

Example:

This handler checks whether the Return key was pressed and sends the playback head to another frame if it was:

```
on keyDown
   if the key = RETURN then go to frame "AddSum"
end keyDown
```

{button See also,AL(`Lingo_on_keyDown')}

on keyUp

Syntax:

```
on keyUp
    statement(s)
end
```

This event handler contains statements that run when a key is released. The `on keyUp` handler is similar to the `on keyDown` handler.

When a key is released, Lingo searches these locations, in order, for an `on keyUp` handler: primary event handler, editable field sprite script, script of a field cast member, frame script, and movie script. For sprites and cast members, `on keyUp` handlers work only for editable strings. A `keyUp` on a different type of cast member, such as a bitmap, has no effect. When releasing a key should always have the same response throughout the movie, set `the 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 have occur when the user releases keys.

Where you place an `on keyUp` handler can affect when it runs:

- When you want the handler to apply to a specific editable field sprite, put the handler in a sprite script.
- When you want the handler to apply to an editable field cast member in general, put the handler in a script of the cast member.
- When you want the handler to apply to an entire frame, put the handler in a frame script.
- When you want the handler to apply throughout the entire movie, put the handler in a movie script.

You can override an `on keyUp` handler by placing an alternate `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.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

Example:

This handler checks whether the Return key was released and sends the playback head to another frame if it was:

```
on keyUp
   if the key = RETURN then go to frame "AddSum"
end keyUp
```

{button See also,AL(`Lingo_on_keyUp')}

on mouseDown

Syntax:

```
on mouseDown
    statement(s)
end
```

This event handler contains statements that run when the mouse button is pressed.

When the mouse button is pressed, Lingo searches these locations, in order, for an `on mouseDown` handler: primary event handler, sprite script, script of a cast member, 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.

When pressing the mouse button should always have the same response throughout the movie, set `the mouseDownScript`. For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

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.

- When you want the handler to apply to a specific sprite, put the handler in a sprite script.
- When you want the handler to apply to a cast member in general, put the handler in a script of the cast member.
- When you want the handler to apply to an entire frame, put the handler in a frame script.
- When you want the handler to apply throughout the entire movie, put the handler in a movie script.

You can override an `on mouseDown` handler by placing an alternate `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.

Note: This event is passed the sprite script or frame script reference `me` if used in a behavior.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

Example 1:

This handler checks whether the user clicks anywhere on the Stage and sends the playback head to another frame if he or she does:

```
on mouseDown
   if the clickOn = 0 then go to frame "AddSum"
end mouseDown
```

Example 2:

This handler, assigned to a sprite script, plays a sound when the sprite is clicked:

```
on mouseDown
   puppetSound "Crickets"
end mouseDown
```

{button See also,AL(`Lingo_on_mouseDown')}

on mouseEnter

Syntax:

```
on mouseEnter
    statement(s)
end mouseEnter
```

This event handler contains statements that run when the mouse cursor 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.

The `mouseEnter` message is sent to the sprite script and then the cast member script; it is not sent to movie scripts.

Note: This event is passed the sprite script or frame script reference `me` if used in a behavior.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example:

This statement plays a sound when the mouse first touches the sprite:

```
on mouseEnter me
  puppetSound "inSound"
end mouseEnter
```

{button See also,AL(`Lingo_on_mouseEnter')}

on mouseLeave

Syntax:

```
on mouseLeave
    statement(s)
end mouseLeave
```

This 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.

The `mouseLeave` message is sent to the sprite script and then the cast member script; it is not sent to movie scripts.

Note: This event is passed the sprite script or frame script reference `me` if used in a behavior.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example:

This statement plays a sound when the mouse leaves the sprite:

```
on mouseLeave me
  puppetSound "outSound"
end mouseLeave
```

{button See also,AL(`Lingo_on_mouseLeave')}

on mouseUp

Syntax:

```
on mouseUp
    statement(s)
end mouseUp
```

This event handler contains statements that are activated when the mouse button is released.

When the mouse button is released, Lingo searches these locations, in order, for an `on mouseUp` handler: primary event handler, sprite script, script of a cast member, 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.

When releasing the mouse button should always have the same response throughout the movie, 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. The sprite's different appearance indicates that the sprite has already been clicked.

Where you place an `on mouseUp` handler can affect when it runs.

- When you want the handler to apply to a specific sprite, put the handler in a sprite script.
- When you want the handler to apply to a cast member in general, put the handler in a script of the cast member.
- When you want the handler to apply to an entire frame, put the handler in a frame script.
- When you want the handler to apply throughout the entire movie, put the handler in a movie script.

You can override an `on mouseUp` handler by placing an alternate `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.

Note: This event is passed the sprite script or frame script reference `me` if used in a behavior.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

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
   puppetSprite 10, TRUE
   set the memberNum of sprite 10 ¬
   to the number of member "Dimmed"
end mouseUp
```

{button See also,AL(`Lingo_on_mouseUp')}

on mouseUpOutside

Syntax:

```
on mouseUpOutside me
    statement(s)
end mouseUpOutside
```

This event handler is sent when the mouse is initially pressed on a sprite but is later released off the sprite.

The `mouseUpOutside` message is sent to the sprite script and then the cast member script; it is not sent to movie scripts.

Note: This event is passed the sprite script or frame script reference `me` if used in a behavior.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example:

This statement displays the mouse location when the mouse is over the sprite:

```
on mouseUpOutside me
  puppetSound "Professor Long Hair"
end mouseUpOutside
```

on mouseWithin

Syntax:

```
on mouseWithin
    statement(s)
end mouseWithin
```

This 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.

The `mouseWithin` message is sent to the sprite script and then the cast member script; it is not sent to movie scripts.

Note: This event is passed the sprite script or frame script reference `me` if used in a behavior.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example:

This statement displays the mouse location when the mouse is over the sprite:

```
on mouseWithin
  put the mouseH into field "Display"
end mouseWithin
```

{button See also,AL(`Lingo_on_mouseWithin')}

on moveWindow

Syntax:

```
on moveWindow
   statement(s)
end
```

This event handler contains statements that run when a window is moved, such as when the user drags the movie to a new location on the Stage. The `on moveWindow` handler is a good place to put Lingo that you want executed every time the movie's window moves.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

Example:

This handler plays the sound file `Honk` when the window that the movie is playing in moves:

```
on moveWindow
   puppetSound 2, "Honk"
end
```

on openWindow

Syntax: `on openWindow`

statement(s)

`end`

This event handler contains statements that run when Director opens a window. The `on openWindow` handler 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
```

on prepareFrame

Syntax:

```
on prepareFrame
    statement(s)
end prepareFrame
```

This event handler contains statements that run before the current frame is drawn.

The `on prepareFrame` handler is a useful place to change sprite properties before the sprite is drawn.

The `on prepareFrame` handler receives the reference `me` if used in a behavior.

The `prepareFrame` message is sent to sprite scripts, the script of the cast member, frame scripts, and movie scripts.

Example:

This handler sets the `locH` of the sprite that the behavior is attached to:

```
on prepareFrame me
  set the locH of sprite the spriteNum of me = the mouseH
end
```

{button See also,AL(`Lingo_on_prepareFrame')}

on prepareMovie

Syntax:

```
on prepareMovie
    statement(s)
end prepareMovie
```

This 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 scripts in the first frame that plays.
- Prepares the first frame that plays. This includes drawing the frame, playing any sounds, and performing transitions and palette effects.

New global variables used for sprite behaviors in the first frame must be initialized in the `on prepareMovie` handler. Global variables already set by the previous movie do not need to be reset.

A `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 to computer conditions such as color depth.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

Example:

This handler creates a global variable when the movie starts:

```
on prepareMovie
   global currentScore
   set currentScore = 0
end prepareMovie
```

on resizeWindow

Syntax:

```
on resizeWindow
    statement(s)
end resizeWindow
```

This event handler contains statements that run when a movie is running as a movie in a window, and the user resizes the window by dragging the window's grow 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 positioning sprites and cropping digital video.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

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
   set the loc of sprite 3 to centerPlace
end
```

on rightMouseDown

Syntax:

```
on rightMouseDown
   statements
   end rightMouseDown
```

This event handler contains statements that run when the right mouse button on a Windows computer is pressed. For Macintosh computers, the statements are activated when the mouse button and Control key are pressed at the same time, provided that the `emulateMultiButtonMouse` property is set to TRUE. If the `emulateMultiButtonMouse` property isn't set to TRUE, this event handler has no effect on the Macintosh.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

Example:

This handler opens the window Help when the user clicks the right mouse button in Windows:

```
on rightMouseDown
   open window "Help"
end
```

on rightMouseUp

Syntax:

```
on rightMouseUp
  statements
end rightMouseUp
```

This event handler contains statements that run when the right mouse button on a Windows computer is released.

For Macintosh computers, the statements are activated if the mouse button is released while the Control key is pressed, provided that the `emulateMultiButtonMouse` property is set to TRUE. If the `emulateMultiButtonMouse` property isn't set to TRUE, this event handler has no effect on the Macintosh.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

Example:

This handler opens the window Help when the user releases the right mouse button in Windows:

```
on rightMouseUp
   open window "Help"
end
```

on runPropertyDialog

Syntax: `on runPropertyDialog me` *currentInitializerList*

This event handler contains Lingo that defines specific values for a behavior's parameters in the Parameters dialog box. The `runPropertyDialog` message is sent when the Parameters dialog box opens.

The current settings of 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 based on the property list returned by the `on getPropertyDescriptionList` handler.

Example:

This handler overrides the behavior's values for the Parameters dialog box for the behavior. Normally the Parameters dialog box allows the user to set the mass and the gravitational constants. However, this handler assigns these parameters constant values:

```
property mass

property gravitationalConstant

on runPropertyDialog me iList

   --force mass to 10

   setaProp currentInitializerList, #mass, 10

   -- force gravitationalConstant to 9.8

   setaProp currentInitializerList, #gravitationalConstant, 9.8

   return iList

end
```

{button See also,AL(`Lingo_on_runPropertyDialog')}

on startMovie

Syntax:

```
on startMovie
    statement(s)
end startMovie
```

This 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.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

Example:

This handler makes sprites invisible when the movie starts:

```
on startMovie
   repeat with counter = 10 to 50
      set the visible of sprite counter to FALSE
   end repeat
end startMovie
```

{button See also,AL(`Lingo_on_startMovie')}

on stepFrame

Syntax:

```
on stepFrame
    statement(s)
end stepFrame
```

This event handler contains statements that run when the playback head enters a frame or the Stage is updated. This handler only works in scripts attached to objects in `the actorList` because these are the only objects that receive a `stepFrame` message.

An `on stepFrame` handler is a useful location for Lingo that you want to attach to a specific set of objects. The most common use of this handler is for Lingo instructions that need to run frequently for child objects, such as Lingo that creates animation.

The `stepFrame` message is sent before the `prepareFrame` message.

Assign objects to `the actorList` if you want the objects to respond to `stepFrame` messages.

For more information about child objects and using the `on stepFrame` handler, see Chapter 12, "Parent Scripts and Child Objects," in *Learning Lingo*.

Example:

If the child object is assigned to `the actorList`, the `on stepFrame` handler in this parent script sends a move message to the child object's `on move` handler each time the playback head enters a frame.

```
property mySprite

on new me, theSprite
  set mySprite = theSprite
  return me
end

on move me
  set the loc of sprite mySprite = point(random(640),random(480))
  return me
end

on stepFrame me
  move me
end
```

on stepMovie

Syntax:

```
on stepMovie
    statement(s)
end stepMovie
```

This event handler, which was used in earlier versions of Director, does the same as the `on enterFrame` handler. Use the `on enterFrame` event handler instead.

{button See also,AL(`Lingo_on_stepMovie')}

on stopMovie

Syntax:

```
on stopMovie
    statement(s)
end stopMovie
```

This 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 jumps to another movie. It isn't called when the window is closed or the window is deleted by the `forget window` command.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

Example:

This handler clears global variables and closes two resource files when the movie stops:

```
on stopMovie
   set gCurrentScore = 0
   closeResFile "Special Fonts"
   closeResFile "Special Cursors"
end stopMovie
```

{button See also,AL(`Lingo_on_stopMovie')}

on streamStatus

Syntax: `on streamStatus` *URL, state, bytesSoFar, bytesTotal, error*

```
    statements
end streamStatus
```

The `on streamStatus` handler determines how much of an object has downloaded from the internet. The handler is only called if `tellStreamStatus (TRUE)` has been called. The handler is called periodically to report the progress of streams that are being retrieved from the internet.

Parameters for the `on streamStatus` event handler have the following uses:

URL	The internet address of the data being retrieved.
state	The state of the stream being downloaded. Possible values are: Connecting, Started, InProgress, Complete, and Error
bytesSoFar	The number of bytes retrieved from the network so far.
bytesTotal	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	Contains the error code if the stream state is "Error"; 0 otherwise.

Network streams can be initiated using Lingo commands, or by linking media from a URL, or by using an external cast member from a URL.

Use `streamStatus` with `gotoNetMovie, getNetText, preloadNetThing,` and `downloadNetThing.`

Place the streamStatus handler in movie scripts.

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
```

on timeOut

Syntax:

```
on timeOut
end timeOut
```

This event handler contains statements that run when no one uses the keyboard or mouse for the length of time set in `the timeOutLength`. This is a useful location for Lingo that you want to execute when the user does nothing for a specified length of time.

When a timeout should always have the same response throughout the movie, set `the timeOutScript`.

An `on timeOut` handler must be placed in a movie script.

For more information, see the chapter, "Script Basics," in *Learning Lingo*.

Example:

This handler plays the movie "Attract Loop" after users do nothing for the time set in the `timeOutLength` property. This can be a way to respond when users have gone away from the computer:

```
on timeOut

   play movie "Attract Loop"

end timeOut
```

{button See also,AL(`Lingo_on_timeOut')}

on zoomWindow

Syntax:

```
on zoomWindow
    statement(s)
end zoomWindow
```

This event handler contains statements that run when a movie that is running as a movie in a window is resized when the user clicks the minimize/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 intended to rearrange sprites when window dimensions change.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

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
   set the loc of sprite 3 to centerPlace
end
```

open

Syntax: `open [`*whichDocument* `with]` *whichApplication*

This command launches the application specified by the string *whichApplication*. By specifying *whichDocument*, you can specify a document that the application opens at the same time. When either is in a different folder than the current movie, you must specify the pathname.

If the computer is using MultiFinder, it must have enough memory to run both Director and the other application at the same time.

Example 1:

This statement checks whether the computer is a Macintosh (by checking whether it isn't a Windows computer) and then opens the application MacWrite if it is:

```
if the machineType <> 256 then open "MacWrite"
```

Example 2:

This statement opens the MacWrite application, which is in the folder Applications on the drive myDrive, and the document named Storyboards:

```
open "Storyboards" with myDrive & "Applications:" ¬
   & "MacWrite"
```

{button See also,AL(`Lingo_open')}

open window

Syntax: `open window "`*whichWindow*`"`

This command opens the window or movie 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 more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

Example:

This statement opens the window Control Panel and brings it to the front:

```
open window "Control Panel"
```

{button See also,AL(`Lingo_open_window')}

openDA

This Lingo element is obsolete.

openResFile

Syntax: `openResFile` *whichFile*

On the Macintosh, this command opens the resource file specified by the string *whichFile*. When the file is in a different folder than the current movie, *whichFile* must specify a pathname.

In earlier versions of Director, this command was necessary to make additional fonts and cursors available in your movies. However, you can now provide custom cursors by importing the cursor as a cast member and using the `cursor` property.

When the file is already open, `openResFile` has no effect. It is good practice to close any open file as soon as you are finished using it.

The `openResFile` command doesn't support URLs as file references.

Do not use `openResFile` to open another application. (Its code resources will interfere with those of Director.) Use a resource mover like ResEdit to move the resources you need to a separate resource file.

Example:

This statement opens the resource file `Special Fonts`:

```
openResFile "Special Fonts"
```

This statement opens the resource file `Special Icons`, which is in another folder:

```
openResFile the pathName&"Special Icons"
```

{button See also,AL(`Lingo_openResFile')}

openXlib

Syntax: `openXlib` *whichFile*

This command opens the Xlibrary file specified by the string expression *whichFile*. If the file is in a different folder than 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. When the file is already open, `openXlib` has no effect.

The `openXlib` command doesn't support URLs as file references.

Xlibrary files contain Xtras and XObjects as XCOD resources. Unlike `openResFile`, `openXlib` makes these Xtras and XObjects known to Director.

In Windows, the DLL extension is optional.

Example 1:

This statement opens the Xlibrary file `Video Disc Xlibrary`:

```
openXlib "Video Disc Xlibrary"
```

Example 2:

This statement opens the Xlibrary file Xtras, which is in a different folder than the current movie:

```
openXlib "My Drive:New Stuff:Transporter Xtras"
```

{button See also,AL(`Lingo_openXlib')}

optionDown

Syntax: `the optionDown`

This function determines whether the Option key on the Macintosh or the Alt key on a PC keyboard is being pressed.

- When the Option or Alt key is being pressed, `the optionDown` is TRUE.
- When the Option or Alt key is not being pressed, `the optionDown` is FALSE.

In Windows, `the optionDown` doesn't work in projectors. Avoid using `the optionDown` if you intend to distribute the movie as a Windows projector.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

For a demonstration of `the optionDown` function, see the sample movie **Keyboard Lingo**.

Example:

This handler checks whether the Option key or Alt key is being pressed and calls the handler named `doOptionKey` if it is:

```
on keyDown

   if the optionDown then doOptionKey(the key)

end keyDown
```

{button See also,AL(`Lingo_optionDown')}

or

Syntax: *logicalExpression1* `or` *logicalExpression2*

This operator performs a logical OR operation on two logical expressions.

- When either expression or both expressions are TRUE, the result is TRUE.
- When both expressions are FALSE, the result is FALSE.

This is a logical operator with a precedence level of 4.

Example 1:

This statement causes the Message window to display 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 2:

This statement checks whether the contents of the field cast member named State are either AK or HI, and displays an alert if they are:

```
if member "State" = "AK" or member "State" = "HI" ¬
   then alert "You're off the map!"
```

{button See also,AL(`Lingo_or')}

otherwise

Syntax: `otherwise` *statement*

This optional keyword precedes instructions that Lingo carries out when none of the earlier conditions in a case statement are met.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

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 beep.

```
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
```

pageHeight of member

Syntax: `the pageHeight of member` *whichCastmember*

This 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.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example:

This statement gets the height of the visible portion of the field cast member Today's News:

```
put the pageHeight of member "Today's News"
```

palette of cast

This is obsolete. Use **palette of member** instead.

palette of member

Syntax: `the palette of member` *whichCastMember*

This cast member property determines which palette is associated with the cast member specified by *whichCastMember*. This property applies to bitmap cast members only.

The `palette of member` property can be tested and set.

Example:

This statement displays the palette assigned to the cast member Leaves in the Message window:

```
put the palette of member "Leaves"
```

paletteMapping

Syntax: `the paletteMapping`

This movie property determines whether Director remaps the movie's palette. Its effect is similar to the Remap Palettes When Needed check box in the **Movie Properties** dialog box.

- When `the paletteMapping` is TRUE, the movie remaps palettes for cast members whose palette is different than the current movie palette.
- When `the paletteMapping` is FALSE, the movie doesn't remap palettes for cast members.

To display different graphics with different palettes simultaneously, you can set `paletteMapping` to TRUE. When it's 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, find the closest match for each pixel in the new palette.

The non-matching bitmap's colors won't be exactly the same as the original colors, but the colors will be relatively close.

Unfortunately, this consumes processor time. It's usually worth the effort to adjust the bitmap's palette in advance.

This feature can give undesirable results when `paletteMapping` is TRUE and a bitmap sprite spans a series of frames. If the palette changes in the middle of the 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 refreshed area of the screen appears in remapped colors.

Example:

This statement has the movie always remap the movie's palette when needed:

```
set the paletteMapping = TRUE
```

paletteRef

Syntax: `the paletteRef`

This 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 differs from `the palette of member`, which returns a positive number for cast palettes and negative numbers for built-in Director palettes.

The `paletteRef` property can be tested and set.

Example:

This statement assigns the Macintosh system palette to the bitmap cast member Shell:

```
set the paletteRef of member "Shell" to #systemMac
```

param

Syntax: `param(`*parameterPosition*`)`

This function provides the value of a parameter in a list. The variable *parameterPosition* represents the parameter's position in the list.

For more information about lists, see Chapter 10, "Working with Lists," in *Learning Lingo*.

Example:

This handler calculates the average value of a list of parameters:

```
on avg first, second, third
   set n = paramCount()
   set sum = 0.0
      repeat with i = 1 to n
         set sum = param(i) + sum
      end repeat
   return sum/n
end avg
```

This statement passes the handler three values and displays the result in the Message window:

```
put avg(1,2,3)
--> 2.0
```

{button See also,AL(`Lingo_param')}

paramCount

Syntax: `the paramCount`

This 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`

This command passes an event message to the next location in the message hierarchy. The `pass` command jumps 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.

Including the `pass` command in a handler causes the event to be passed to other objects in the hierarchy even though the handler would otherwise intercept the event.

Passing an event message to other locations in the message hierarchy lets you execute more than one handler for a given event.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

Example:

This statement passes the event to the cast member level.

```
on mouseUp me

   if (movie_needs_to_know) then pass

end
```

{button See also,AL(`Lingo_pass')}

pasteClipBoardInto

Syntax: `pasteClipBoardInto member` *whichCastmember*

This command pastes the contents of the Clipboard into the cast member specified by *whichCastMember*. When you paste into an occupied Cast window location, the old cast member is completely erased. For instance, 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 strings 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 in projectors. It's best used during authoring.

Example:

This statement pastes the contents of the Clipboard into the bitmap cast member Shrine:

```
pasteClipBoardInto member "Shrine"
```

pathName

Syntax: `the pathName`

This function returns a string containing the full pathname of the folder in which the current movie is located.

The `pathName` function returns a URL if the movie is on the internet.

You can write pathnames that work on both Windows and Macintosh computers by using the `@` operator.

Example 1:

This statement checks whether the pathname contains the term System and causes the computer to beep if it does:

```
if the pathName contains "System" then beep
```

Example 2:

These statements check whether the movie is playing in Windows or on the Macintosh, and then plays the sound file Crash.aif in the Sounds subfolder of the current movie's folder. By checking which platform the movie is playing on, the movie uses the `sound playFile` statement that has the appropriate folder delimiter:

```
case (the platform) of
  "Windows,32", "Windows,16" :sound playFile 1, ¬
  the pathname&"sounds/Crash.aif"
  otherwise sound playFile 1, the pathname&¬
  "sounds:Crash.aif"
end case
```

{button See also,AL(`Lingo_pathName')}

pattern

Syntax: `the pattern of member` *whichCastmember*

This shape cast member property determines the pattern associated with the specified shape. Possible values are the numbers that correspond to the chips 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. This property can be tested and set.

This can be useful in Shockwave movies for changing images by changing the tiling applied to a shape, allowing you to save memory required by larger bitmaps.

For more information about authoring from Lingo, see Chapter 13, "Authoring from Lingo," in *Learning Lingo*.

Example 1:

The following statements make the shape cast member Target Area a filled shape and assign it the pattern numbered 0, which is a solid color:

```
set the filled of member "Target Area" = TRUE

set the pattern of member "Target Area" = 0
```

Example 2:

This handler cycles through eight tiles, with each tile's number offset from the previous. This lets you create animation using smaller bitmaps:

```
on exitFrame
  set currentPat to the pattern of member "Background Shape"
  set nextPat to 57 + ((currentPat - 56) mod 8)
  set the pattern of member "Background Shape" to nextPat
  go to the frame
end
```

pause

Syntax: `pause`

This command was used in earlier versions of Director to halt the playback head.

In general, using `pause` is not recommended. It's better to use the statement `go to the frame` in a frame script for the frame that you want to stay in. This has the movie appear to pause, but lets the movie continue to respond to events.

For more information about navigation, see Chapter 3, "Navigation," in *Learning Lingo*.

Example:

The following `on mouseUp` handler for a button alternately pauses and continues the animation, like the pause button on a videocassette recorder:

```
on mouseUp
   if the pauseState = TRUE then
      go to the marker + 1
   else
      pause
   end if
end mouseUp
```

{button See also,AL(`Lingo_pause')}

pause member

Syntax: `pause member (*"whichCastmember"*)`

This command pauses the playback of a Shockwave Audio (SWA) streaming cast member. When the sound is paused, the `state of member` property equals 4.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This handler could be used for a play/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
   set whatState = the state of member "soundSWA"
   if whatState = 3 then
      pause member "soundSWA"
   else
      play member "soundSWA"
   end if
end
```

pausedAtStart of member

Syntax: `the pausedAtStart of member` *whichDigitalVideo trueOrFalse*

This digital video cast member property specifies whether the Paused at Start check box in the **Digital Video Cast Member Properties** dialog box is checked or not.

- When the `pausedAtStart of member` property is TRUE, the Paused at Start check box is checked.
- When the `pausedAtStart of member` property is FALSE, the Paused at Start check box is not checked.

The `pausedAtStart of member` property can be tested and set.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

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:

```
set the pausedAtStart of member "Rotating Chair" = TRUE
```

pauseState

Syntax: `the pauseState`

This property returns TRUE when the movie is currently paused.

Example:

This statement checks whether the movie is currently paused and causes the movie to continue if it is:

```
if the pauseState = TRUE then go to the marker + 1
```

{button See also,AL(`Lingo_pauseState')}

percentPlayed of member

Syntax: `the percentPlayed of member` *"whichCastmember"*

This cast member property returns the percentage of the specified Shockwave Audio (SWA) file that has actually played.

This property can be tested only after the SWA sound is playing or has been preloaded using the `preLoadBuffer` command. This property cannot be set.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

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
   set whatState = the state of member "Frank Sinatra"
   if whatState  > 1 AND whatState < 9 then
      put the percentPlayed of member "Frank Sinatra"
      into member "Percent Played"
   end if
end
```

percentStreamed of member

Syntax: `the percentStreamed of member` *"whichCastmember"*

This Shockwave Audio (SWA) streaming sound cast member property indicates the percent of the SWA file already streamed from an ftp or HTTP server.

This differs from the `percentPlayed` property in that it also includes the amount of the file that has been buffered, but not yet been played.

This property is only available after the SWA sound is playing or has been preloaded using the `preLoadBuffer` command.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This example displays the percentage of SWA streaming cast member Ray Charles that has streamed and puts the value in a field:

```
on exitFrame
   set whatState = the state of member "Ray Charles"
   if whatState  > 1 AND whatState < 9 then
      put the percentStreamed of member "Ray Charles"
      into member "Percent Streamed Displayer"
   end if
end
```

perFrameHook

Syntax: `the perFrameHook`

The `perFrameHook` property is obsolete. It was used in earlier versions of Director to send messages to XObjects. You should now use the `actorList` property and the `on stepFrame` handler to send messages to a set of child objects.

{button See also,AL(`Lingo_perFrameHook')}

pi

Syntax: `pi()`

This function gives the value of pi (¹), the ratio of a circle's circumference to its diameter. The value of ¹ is given as a floating-point number to the number of decimal places set by the `floatPrecision` property.

Example:

This statement uses the pi function as part of an equation for calculating the area of a circle:

```
set vArea = pi()*power(vRadius,2)
```

picture of cast

This is obsolete. Use **picture of member** instead.

picture of member

Syntax: `the picture of member` *`whichCastmember`*

This 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:

```
set the picture of member whichCastmember = the picture ¬
of member whichCastmember
```

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 `picture of member` property can be tested and set.

For more information about parent scripts and child objects, see Chapter 12, "Parent Scripts and Child Objects," in *Learning Lingo*.

Example:

This statement sets the variable named `pictHolder` to the image in the cast member named Sunset:

```
set pictHolder = the picture of member "Sunset"
```

{button See also,AL(`Lingo_picture_of_member')}

pictureP

Syntax: `pictureP(`*pictureValue*`)`

This function tells the state of the `picture of member` property for the specified cast member.

- When `the picture of member` property is TRUE, `pictureP` is TRUE (1).
- When the cast member is not a picture data type, `pictureP` is FALSE (0).

Because `pictureP` doesn't directly check whether a cast member has a picture, you must test whether a cast member has a picture by checking the cast member's `picture of member` property.

Example:

The first statement assigns the value of the `picture of 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`

This property indicates the platform type that the projector was created for. It can be tested but not set.

Possible values are the following:

Possible value	Corresponding platform
`Macintosh,68k`	Original 68K Macintosh
`Macintosh,PowerPC`	PPC Macintosh
`Windows,16`	Windows 3.1 or earlier
`Windows,32`	Windows 95 or WinNT

Example:

This statement checks whether the movie was created in Windows 95 and assigns the cast Win95 Art the name Interface if it is:

```
if the platform contains "Windows,32" then set the name ¬
of castLib "Win95 Art" to "Interface"
```

play

Syntax: `play [frame]` *whichFrame*

`play movie` *whichMovie*

`play frame` *whichFrame* `of movie` *whichMovie*

This command causes the playback head to jump to the specified frame of the specified movie. 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 pathname.

The `play` command is similar to the `go to` command, but with the `play` command, when the sequence being played is over, the playback head automatically returns to the frame where the `play` command was called. If the `play` command is issued from a frame script, the playback head returns to the next frame; if the `play` command comes from a sprite script or handler, the playback head returns to the same frame. A sequence is over when the playback head reaches the end of the movie, or when the `play done` command is given.

To play a movie from a URL, it's a good idea to use the `downloadNetThing` or `preLoadNetThing` command to download the file to a local disk first, and then use the `play` command to play the movie on the local disk. This minimizes problems with waiting for the file to download.

The `play` command can also be used for playing several movies from a single handler. The handler is suspended while each movie plays but resumes when the movie is over. 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.

If a `play done` command isn't used to indicate the end of a segment started by a `play` command, memory gets used up because the original calling script isn't deleted. If you aren't sure that each `play` command has a matching `play done` command, consider avoiding `play` commands. Instead, use a global list to record where the movie should return to.

For more information on about navigation, see Chapter 3, "Navigation," in *Learning Lingo*.

Example 1:

This statement moves the playback head to the marker named blink:

```
play "blink"
```

Example 2:

This statement moves the playback head to the next marker:

```
play marker(1)
```

Example 3:

This statement moves the playback head to a separate movie:

```
play movie "My Drive:More Movies:" & newMovie
```

{button See also,AL(`Lingo_play')}

play done

Syntax: `play done`

This command indicates that the sequence being played is complete when the current movie or sequence was started using the `play` or `go to` commands. The `play done` command causes the playback head to return to where the sequence was started from. If the `play` command is issued from a frame script, the playback head returns to the next frame; if the `play` command is issued from a sprite script, the playback head returns to the same frame.

If a `play done` command isn't used to indicate the end of a segment started by a `play` command, memory gets used up because the original calling script isn't deleted. If you aren't sure that each `play` command has a matching `play done` command, consider avoiding `play` commands. Instead, use a global list to record where the movie should return to.

Note: The `play done` command has no effect in a movie that is playing in a window.

Example:

This handler has the playback head return to the frame of the movie that was playing before the current movie started:

```
on exitFrame
  play done
end
```

{button See also,AL(`Lingo_play_done')}

play member

Syntax: `play member` *"whichCastmember"*

This command begins playback of a Shockwave Audio (SWA) streaming cast member.

If the sound has not been preloaded using the `preLoadBuffer` command, the SWA sound preloads before playing begins. When the sound is playing, the `state of member` property equals 3.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This handler begins the playback of cast member Big Band:

```
on mouseDown
  play member "Big Band"
end
```

point

Syntax: `point(`*horizontal*`, `*vertical*`)`

This function 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.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example 1:

This statement sets the variable `lastLocation` to the point (250, 400):

```
set lastLocation = point(250, 400)
```

Example 2:

This statement adds 5 pixels to the horizontal coordinate of the point assigned to the variable `myPoint`:

```
set the locH of myPoint to the locH of myPoint + 5
```

Example 3:

These statements set a sprite's Stage coordinates to `mouseH` and `mouseV` plus 10 pixels. The two statements are equivalent:

```
set the loc of sprite (the clickOn) to ¬
point(the mouseH, the mouseV) + point(10, 10)

set the loc of sprite (the clickOn) to point(the mouseH, the mouseV) + 10
```

Example 4:

This handler moves a named sprite to the location that the user clicks.

```
on mouseDown
  -- Set these variables as needed for your own movie
  set theSprite to 1 -- Set the sprite that should move
  set steps to 40 -- Set the number of steps to get there
  set initialLoc to the loc of sprite theSprite
  set delta to (the clickLoc - initialLoc) / steps
  repeat with i = 1 to steps
    set the loc of sprite theSprite to ¬
    (initialLoc + (i * delta))
     updateStage
  end repeat
end mouseDown
```

{button See also,AL(`Lingo_point')}

power

Syntax: `power(`*base, exponent*`)`

This 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

Syntax: `preLoad`

`preLoad` *toFrameNum*

`preLoad` *fromFrame*, *toFrameNum*

This command preloads cast members in the specified frame or range of frames into memory. Preloading stops when memory is full or when all of the specified cast members have been preloaded.

When used without arguments, the `preLoad` command causes a preload of all cast members used from the current frame to the last frame of a movie.

When used with one argument, *toFrame*, the `preLoad` command causes a preload of all cast members used in the range of frames from the current frame to the frame *toFrame*, as specified by frame number or label name.

When used with two arguments, *fromFrame* and *toFrame*, the `preLoad` command causes a preload of all cast members used in the range of frames from the frame *fromFrame* to the frame *toFrame*, as specified by frame number or label name.

The `preLoad` command also returns the number of the last frame successfully loaded. To access this value, use the `result` function.

Example 1:

This statement preloads the cast members used from the current frame to the frame that has the next marker:

```
preLoad marker (1)
```

Example 2:

This statement preloads the cast members used from frame 10 to frame 50:

```
preLoad 10, 50
```

{button See also,AL(`Lingo_preLoad')}

"

preLoad of cast

This is obsolete. Use **preLoad of member** instead.

preLoad of member

Syntax: `the preLoad of member` *whichCastmember*

This digital video cast member property determines whether the digital video cast member specified by *whichCastmember* can preload into memory.

- When the digital video cast member can be preloaded into memory, `the preLoad of member` is TRUE.
- When the digital video cast member cannot be preloaded into memory, `the preLoad of member` is FALSE.

Setting `the preLoad of member` to TRUE has the same effect as selecting Enable Preload in the **Digital Video Cast Member Properties** dialog box.

Example:

This statement causes the Message window to display whether the QuickTime movie "Rotating Chair" can be preloaded into memory:

```
put the preLoad of member "Rotating Chair
```

preLoadBuffer member

Syntax: `preLoadBuffer member "`*whichCastmember*`"`

This 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.

After the `preLoadBuffer` command is successful, `the state of member` equals 2. Most SWA cast member properties can be tested only after the `preLoadBuffer` command has completed successfully. Some of these properties are the following: `cuePointNames`, `cuePointTimes`, `currentTime`, `duration`, `percentPlayed`, `percentStreamed`, `bitRate`, `sampleRate`, and `numChannels`.

Example:

This statement loads the cast member Mel Torme into memory:

```
preLoadBuffer (member "Mel Torme")
```

preLoadCast

This is obsolete. Use **preLoadMember** instead.

preLoadEventAbort

Syntax: `the preLoadEventAbort`

This property specifies whether pressing keys or clicking the mouse can stop the preloading of cast members.

- When the `preLoadEventAbort` property is TRUE, pressing keys or clicking the mouse can stop the preloading of cast members.
- When the `preLoadEventAbort` property is FALSE, pressing keys or clicking the mouse cannot stop the preloading of cast members.

The default value is FALSE. The setting of this property affects the current movie.

The `preLoadEventAbort` property can be tested and set.

Example:

This statement lets the user stop the preloading of cast members by pressing keys or clicking the mouse:

```
set the preLoadEventAbort = TRUE
```

{button See also,AL(`Lingo_preLoadEventAbort')}

preLoadMember

Syntax: `preLoadMember`

`preLoadMember` *whichCastmember*

`preLoadMember` *fromCastmember, toCastmember*

This command preloads cast members. Preloading stops when memory is full or when all of the specified cast members have been preloaded.

When used without arguments, the `preLoadMember` command preloads all cast members in the movie.

When used with the *whichCastmember* argument, the `preLoadMember` command preloads that cast member.

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.

The `preLoadMember` command returns the cast member number of the last cast member successfully loaded. To obtain this value, use the `result` function.

Example 1:

This statement preloads cast member 20:

```
preLoadMember 20
```

Example 2:

This statement preloads cast member Shrine and the ten cast members after it in the Cast window :

```
preLoadMember "Shrine", (the number of member "Shrine" + 10)
```

preLoadMode of CastLib

Syntax: `the preLoadMode of castLib` *whichCast*

This cast property determines the specified cast's preload mode. This has the same effect as setting Load Cast in the **Cast Properties** dialog box.

Possible values are the following:

0-When Needed

1-Before Frame One

2-After Frame One

A `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 causes Director to load the members of the cast Buttons before the movie enters frame one:

```
set the preLoadMode of castLib "Buttons" = 1
```

{button See also,AL(`Lingo_preLoadMode_of_CastLib')}

preLoadMovie

Syntax: `preLoadMovie` *whichMovie*

This command preloads the 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 the `preloadNetThing` or the `downloadNetThing` command.

- Using `preloadNetThing` loads the cast members directly to cache.
- Using `downLoadNetThing` loads the movie to a local disk first, from which you can load the movie into memory. This can minimize problems with waiting for the file to download.

Example:

This statement preloads the movie Introduction:

```
preLoadMovie "Introduction"
```

preloadNetThing

Syntax: `preloadNetThing (URL)`

This command preloads a file from the internet to the browser's 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, HTML page, ftp server location, or graphic.

The `preloadNetThing` command loads the file while the current movie continues playing. Use `netDone` to find out whether preloading is finished.

After an item is preloaded, it can be displayed immediately because it is taken from the local browser's 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 affects the behavior of the `preLoadNetThing` command.

The `preloadNetThing` command does not parse an HTML file's EMBED tags. Thus, even if an HTML page contains embedded pictures and Shockwave movies, `preloadNetThing` downloads the page's HTML. You still must preload other objects embedded in the page.

For more information about Shockwave movies and the internet, see Chapter 14, "Shockwave, the Internet, and Lingo," in *Learning Lingo*.

Example 1:

This statement preloads the file `Roses.pict` from the internet:

```
preLoadNetThing("http://www.flowers.com/Roses.pict")
```

Example 2:

This statement uses `preloadNetThing` as a function and returns the network ID for the operation:

```
set mynetid = # ¬ preloadNetThing("http://www.yourserver.com/menupage/¬
default.html")
```

{button See also,AL(`Lingo_preloadNetThing')}

preLoadRAM

Syntax: `the preLoadRAM`

This property specifies the amount of RAM that can be used for preloading a digital video. It can be set and tested.

This is useful for managing memory, so that digital video cast members are not given more than a certain limit of memory, and other types of cast members can still be preloaded. When `the preLoadRAM` is FALSE, all available memory can be used for preloading digital video cast members.

Example:

This statement allocates the amount of RAM available for preloading to 3 times the size of the cast member Interview:

```
set the preLoadRAM to 3 * (the size of member "Interview")
```

{button See also,AL(`Lingo_preLoadRAM')}

preLoadTime of member

Syntax: `preLoadTime of member` *whichCastmember*

This property specifies the amount of the Shockwave Audio (SWA) streaming cast member to download before playback begins or when a `preLoadBuffer` command is used. The value is in seconds.

This property can be set only when the SWA streaming cast member is stopped. The default value is 5 seconds.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This handler sets 6 seconds as the amount of time to download from the SWA streaming cast member Louis Armstrong. The actual preload occurs when a `preLoadBuffer` or `play` command is issued:

```
on mouseDown

   stop member "Louis Armstrong"

   set the preLoadTime of member "Louis Armstrong " = 6

end
```

prepareFrame

See **on prepareFrame** event handler.

prepareMovie

See **on prepareMovie** event handler.

previous

See: go previous command

printFrom

Syntax: `printFrom` *fromFrame* `[,` *toFrame*`] [,` *reduction*`]`

This command prints whatever is displayed on the Stage in each frame starting at the frame specified by *fromFrame*. Optionally, you can supply *toFrame*, and the reduction (100, 50, or 25 percent).

When printing at less than 100 percent, the document prints as a bitmap, so text does not print as sharply as it would at full size.

Example:

This statement prints what is on the Stage in every frame starting at 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 percent:

```
printFrom label("Intro"), ("Tale"), 50
```

property

Syntax: `property [property1][, property2][, property3] [...]`

This keyword declares that the properties specified by *property1*, *property2*, and so on are 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.

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: `the property of me`.

For behaviors, properties defined in one behavior script are available to other behaviors attached to the same sprite. If a behavior has an ancestor that needs the sprite's number, declare `spriteNum` as a property in the ancestor script.

A child object's property can be directly manipulated from outside the object's parent scripts through syntax similar to that for manipulating other properties. For example, this sets the `motionStyle` property of a child object:

```
set the motionStyle of myBouncingObject to #frenetic
```

The `count` function can 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`.

Example 1:

This statement allows each child object created from a single parent script to have its own location and velocity setting:

```
property location, velocity
```

This handler assigns a random `color` property if the object doesn't already have a `color` property:

```
on DoesItNeedAColor theObject
  repeat with i = 1 to count (theObject)
    if getPropAt (theObject, i) = #color then exit
  end repeat
  set colors to [#red, #green, #blue]
  set thisColor to getAt(colors, random(count(colors)))
  setaProp (theObject, #color, thisColor)
end
```

This handler declares `spriteNum` a property in a behavior's ancestor script:

```
-- script Elder

property spriteNum

on new me, spriteNumber

   set the spriteNum of me = spriteNumber

   return me

end
```

In the original behavior script, this sets up the ancestor:

```
property ancestor

on beginSprite me
```

```
    set ancestor = new(script "Elder",the spriteNum of me)
end
```

{button See also,AL(`Lingo_property')}

proxyServer

Syntax: `proxyServer` *`serverType, "ipAddress", portNum`*

or

`proxyServer()`

This command sets the values of an ftp or HTTP proxy server.

- The *serverType* parameter can be `#ftp` or `#http`.
- The *ipAddress* parameter is a string containing the IP address.
- The *portNum* parameter is 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.

For more information about Shockwave movies and the internet, see Chapter 14, "Shockwave, the Internet, and Lingo," in *Learning Lingo*.

Example 1:

This statement sets up an HTTP proxy server at IP address 197.65.208.157 with port 5:

```
proxyServer #http,"197.65.208.157",5
```

Example 2:

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 3:

This statement returns the IP address string of an HTTP proxy server:

```
put proxyServer(#http)
```

Example 4:

This statement disables turns off an ftp proxy server:

```
proxyServer #ftp,#stop
```

puppet of sprite

Syntax: `the puppet of sprite` *whichSprite*

This sprite channel property determines whether the sprite channel specified by *whichSprite* is under control by Lingo.

Note: A sprite channel under control by Lingo is referred to as a puppet.

- 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.

While the playback head is in the same sprite, setting the sprite channel's `puppet of 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 of sprite`, `locH of sprite`, and `width of sprite`-from Lingo after the playback head exits the sprite.

Setting the `puppet of sprite` property is equivalent to using the `puppetSprite` command. For example, the statement:

```
set the puppet of sprite 1 to TRUE
```

has the same effect as:

```
puppetSprite 1, TRUE
```

The `puppet of sprite` property can be tested and set. The default value is FALSE.

For more information about using puppets, see Chapter 5, "Controlling Score Channels from Lingo," in *Learning Lingo*.

Example 1:

This statement makes the sprite numbered i + 1 a puppet:

```
set the puppet of sprite (i + 1) to TRUE
```

Example 2:

This statement records whether sprite 5 is a puppet by assigning the value of `the puppet of sprite` 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:

```
put the puppet of sprite 5 into isPuppet
```

{button See also,AL(`Lingo_puppet_of_sprite')}

puppetPalette

Syntax: `puppetPalette` *whichPalette* `[,` *speed*`] [,` *nFrames*`]`

This command causes the palette channel to act as a puppet. When the palette channel is a puppet, Lingo can 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 the expression *whichPalette*. If *whichPalette* evaluates to a string, it specifies the cast name of the palette. If *whichPalette* evaluates to an integer, it specifies the cast number of the palette.

Optionally, you can fade in the palette by replacing *speed* with an integer expression, with 1 being slowest and 60 being fastest. You can also fade in the palette over several frames by replacing *nFrames* with an integer expression 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.

For more information about using puppets, see Chapter 5, "Controlling Score Channels from Lingo," in *Learning Lingo*.

Example 1:

This statement makes Rainbow the movie's palette:

```
puppetPalette "Rainbow"
```

Example 2:

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")
```

{button See also,AL(`Lingo_puppetPalette')}

puppetSound

Syntax:

```
puppetSound whichChannel, "whichCastMember"
puppetSound "whichCastMember"
puppetSound member "whichCastMember"
puppetSound 0
puppetSound whichChannel, 0
```

This command makes the sound channel a puppet and plays the sound cast member specified by *whichCastMember*. When the sound is a puppet, Lingo can override any sounds assigned in the Score's sound channels.

For sound cast members, 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.

For more information about using puppets, see Chapter 5, "Controlling Score Channels from Lingo," in *Learning Lingo*.

Example 1:

This statement plays the sound Wind under control of Lingo:

```
puppetSound "Wind"
```

Example 2:

This statement turns off the sound playing in channel 2:

```
puppetSound 2, 0
```

{button See also,AL(`Lingo_puppetSound')}

puppetSprite

Syntax: `puppetSprite` *whichChannel, state*

This command sets whether the sprite channel specified by *whichSprite* is under control by Lingo.

Note: A sprite channel under control by Lingo is referred to as a puppet.

- When *state* is TRUE, Lingo controls the sprite channel and the Score is ignored.
- When *state* is FALSE, the sprite channel is controlled by the Score.

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 taken from whatever the channel's settings are when the `puppetSprite` command is executed. Subsequent control of the sprite properties through Lingo can change these properties.

- 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 of sprite`, `locH of sprite`, and `width of sprite`-from Lingo, after the playback head exits the sprite.

Use the command `puppetSprite` *whichSprite*`, FALSE` to return control to the Score when finished with controlling a sprite channel from Lingo. Otherwise, unpredictable results can occur when the playback head is in frames that aren't intended to be puppets.

For more information about using puppets, see Chapter 5, "Controlling Score Channels from Lingo," in *Learning Lingo*.

Example 1:

This statement makes the sprite in channel 15 a puppet:

```
puppetSprite 15, TRUE
```

Example 2:

This statement removes the puppet condition from the sprite in the channel numbered i + 1:

```
puppetSprite i + 1, FALSE
```

{button See also,AL(`Lingo_puppetSprite')}

puppetTempo

Syntax: `puppetTempo` *framesPerSecond*

This command causes the tempo channel to act as a puppet. When the tempo channel is a puppet, Lingo can override the tempo setting in the Score and change the tempo assigned to the movie.

The `puppetTempo` command sets the tempo to the number of frames specified by *framesPerSecond*. The maximum frames per second is 60.

You do not need to turn off the puppet tempo condition to have subsequent tempo changes in the Score take effect.

For more information about using puppets, see Chapter 5, "Controlling Score Channels from Lingo," in *Learning Lingo*.

Example 1:

This statement set the movie's tempo to 30 frames per second:

```
puppetTempo 30
```

Example 2:

This statement increases the movie's old tempo by ten frames per second:

```
puppetTempo oldTempo + 10
```

{button See also,AL(`Lingo_puppetTempo')}

puppetTransition

Syntax: `puppetTransition member` *whichCastMember*

`puppetTransition member` *castmemberReference*

`puppetTransition` *whichTransition* `[,` *time*`]¬`

`[,` *chunkSize*`] [,` *changeArea*`]`

This command performs the specifed 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 one of the following values:

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 (*) in the table will not work on monitors that are set to 32 bits.

Replace *time* with the number of 1/4 seconds used to complete the transition. The minimum is 0; the maximum is 120 (30 seconds). Replace *chunkSize* with the number of pixels in each chunk of the transition. The minimum is

1; the maximum is 128. Smaller chunk sizes give smoother transitions but are slower.

There is no direct relationship between a low time and a fast transition. The actual speed of the transition depends on the relation of *chunkSize* and *time*. As an example, if *chunkSize* is one pixel, the transition takes a long time no matter how low the time, because the computer has to do a lot of work. To make transitions occur faster you should use a larger chunk size, instead of setting a shorter time.

Replace *changeArea* with a value that determines whether the transition occurs only in the changing area. The *changeArea* variable is an area within which sprites have changed.

- To have the transition occur only in the areas that change, replace *changeArea* with FALSE, which is the default setting.
- To have the transition occur over the entire Stage, replace *changeArea* with TRUE.

For more information about using puppets, see Chapter 5, "Controlling Score Channels from Lingo," in *Learning Lingo*.

Example:

This statement performs a wipe from right transition. Because no value is specified for `changeArea`, the transition occurs only on the changing area, which is the default:

```
puppetTransition 1
```

This statement performs a wipe from right transition that lasts 1 second, has a chunk size of 20, and occurs over the entire Stage:

```
puppetTransition 2, 4, 20, TRUE
```

{button See also,AL(`Lingo_puppetTransition')}

purgePriority of cast

This is obsolete. Use **purgePriority of member** instead.

purgePriority of member

Syntax: `the purgePriority of member` *whichCastMember*

This 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 is deleted. The following `purgePriority` settings are available:

0-Never purge

1-Purge last

2-Purge next

3-Purge normal

Normal allows Director to purge cast members from memory at random. Next, Last, and Never allows some control over purging. However, if you set several cast members to Last or Never, your movie may run out of memory.

Setting `purgePriority of member` 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 by assigning a low purge priority to cast members that are frequently used in the course of the movie. This reduces the number of times that Director reloads the cast member when the movie plays.

Example:

This statement sets the purge priority of cast member Background to 2, which makes it one of the first cast members to be purged when memory is needed:

```
set the purgePriority of member "Background" to 2
```

put

Syntax: `put` *expression*

This command evaluates the expression specified by *expression* and displays the result in the Message window. This can be used as a debugging tool by tracking the values of variables as the movie plays.

Example 1:

This statement displays the time in the Message window:

```
put the time

-- "9:10 AM"
```

Example 2:

This statement displays the value assigned to the variable bid in the Message window:

```
put bid

-- "Johnson"
```

{button See also,AL(`Lingo_put')}

put...after

Syntax: `put` *expression* `after` *chunkExpression*

This command evaluates a Lingo expression, converts the value to a string, and inserts the resulting string after a specified chunk in a container. (If *chunkExpression* specifies a nonexistent target chunk, the string value is inserted as appropriate into the container.) The previous contents of the container remain.

Chunk expressions can refer to 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.

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"
```

{button See also,AL(`Lingo_put_after')}

put...before

Syntax: `put` *expression* `before` *chunkExpression*

This command evaluates a Lingo expression, converts the value to a string, and inserts the resulting string before a specified chunk in a container. (If *chunkExpression* specifies a nonexistent target chunk, the string value is inserted as appropriate into the container.) The previous contents of the container remain.

Chunk expressions can refer to any character, word, item, or line in any container. Containers include fields (field cast members) and variables that hold strings, and specified characters, words, items, lines, and ranges in containers.

Example:

This statement sets the variable `Animal List` to the string "fox dog cat" and then inserts the word *elk* before the second word of the list:

```
put "fox dog cat" into Animal List

put "elk " before word 2 of Animal List
```

The result is the string "fox elk dog cat".

{button See also,AL(`Lingo_put_before')}

put...into

Syntax: `put` *expression* `into` *chunkExpression*

This 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 can refer to any character, word, item, or line in any container. Containers include field cast members and variables that hold strings, and specified characters, words, items, lines, and ranges in containers.

Note: Earlier versions of Director also used the `put...into` command to assign values to variables. It is no longer supported for this purpose. Use the `set` command to assign values to variables.

Example:

This statement changes the second line of field cast member Review Comments to "Reviewed by Agnes Gooch":

```
put "Reviewed by Agnes Gooch" into line 2 of ¬
member "Review Comments"
```

{button See also,AL(`Lingo_put_into')}

putImageIntoCastMember

Syntax: `putImageIntoCastMember (buttonCastMember, "imageString", castMember)`

For buttons created in the Button Editor, this function takes the specified image string of the specifed button cast member and creates a bitmap cast member.

Example:

This statement takes the image named ImageRollover from the Button Editor values for cast member 1 and creates the bitmap cast member Panic in castLib 2.

```
putImageIntoCastMember(member 1, "ImageRollover", member ¬
"Panic" of castLib 2)
```

quickTimePresent

Syntax: `the quickTimePresent`

This function determines whether the QuickTime extension is currently loaded into memory.

- When the extension is present, the `quickTimePresent` function is TRUE (1).
- When the extension is not present, the `quickTimePresent` function is FALSE (1).

Example:

This statement determines whether the QuickTime extension is in memory and sets `the movieRate` for a QuickTime sprite indicated by `QTSprite`:

```
if the quickTimePresent = 1 then ¬
   set the movieRate of sprite QTSprite
```

quit

Syntax: `quit`

This command exits from Director or a projector to the Windows Desktop or Macintosh Finder.

Example:

This statement causes the computer to exit to the Desktop when the user presses Control+Q:

```
if the key = "q" and the commandDown then quit
```

{button See also,AL(`Lingo_quit')}

QUOTE

Syntax: `QUOTE`

This character constant represents the quote character. It is needed to refer to the literal quote character in a string, because the quote character itself is used by Lingo scripts to delimit strings.

Example:

This statement inserts quote characters in the string:

```
put "Can you spell" && QUOTE & "Macromedia" ¬
   & QUOTE & "?"
```

The result is quotes around the word Macromedia, as in the following string:

Can you spell "Macromedia"?

ramNeeded

Syntax: `ramNeeded (`*firstFrame, lastFrame*`)`

This function determines, in bytes, the memory needed 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. Divide by 1024 to convert bytes to kilobytes (K).

Example 1:

This statement sets the variable `frameSize` to the number of kilobytes needed to display frames 100 to 125 of the movie:

```
put ramNeeded (100, 125) into frameSize
```

Example 2:

This statement determines whether the memory needed to display frames 100 to 125 is more than the available memory and branches to a movie using cast members that have lower color depth if it is:

```
if ramNeeded (100, 125) > freeBytes then ¬
   play frame "8-bit"
```

{button See also,AL(`Lingo_ramNeeded')}

random

Syntax: `random(`*integerExpression*`)`

This function returns a random integer in the range from 1 to the value specified by *integerExpression*.

The `random` function is useful when you want to randomly vary values in a movie. Some possible uses are varying the path through a game, assigning random numbers, or changing the color or position of sprites.

In cases when you want a set of possible random numbers to start with a number other than 1, subtract the appropriate amount from the `random` function. For example, the expression `random(n) - 1` uses a range from 0 to the number `n`.

Example 1:

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:

```
set the foreColor of sprite 10 = random(256) - 1
```

Example 2:

This handler randomly chooses which of two movie segments to play in the "Noh Tale" movie:

```
on selectMovie
  if random(2) = 2 then play frame "11a"
  else
    play frame "11-b" of movie "NT.OTher Movie"
  end if
end
```

Example 3:

The following statements produce results in a desired range:

This statement produces a random multiple of five between five and one hundred:

```
set theScore to 5 * random(20)
```

This statement produces a random multiple of five between zero and one hundred:

```
set theScore to 5 * (random(21) - 1)
```

This statement generates integers between -10 and +10:

```
set dirH to random(21) - 11
```

This statement produces a random two-point decimal value:

```
set theCents to random(100)/100.0 - .01
```

randomSeed

Syntax: `the randomSeed`

This property specifies seed for generating random numbers. Using the same seed produces the same sequence of random numbers.

The `randomSeed` property can be tested and set.

Example:

This statement displays the random seed number in the Message window:

```
put the randomSeed
```

rect

Syntax: `rect(left, top, right, bottom)`

`rect(point1, point2)`

This function has two uses:

- When you use four arguments, the `rect` function defines a rectangle that has the sides 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.
- When you use two arguments, the `rect` function defines a rectangle that encloses the points specified by *point1* and *point2*.

You can refer to `rect` components by list syntax or property syntax. For example, the following two phrases are equivalent:

```
set targetWidth the right of targetRect - the left of targetRect
set targetWidth to getAt(targetRect, 3) - getAt(targetRect, 1)
```

You can perform arithmetic operations on rects. If you add a single value to a rect, Lingo adds it to each element in the rect.

Example 1:

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 2:

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 rect as the rectangle created in the previous example:

```
put rect(firstPoint, secondPoint)
```

Example 3:

These statements add and subtract values for rects:

```
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 4:

This statement adds 80 to each coordinate in a rect:

```
put rect(60, 40, 120, 200) + 80
-- rect(140, 120, 200, 280)
```

Example 5:

This statement divides each coordinate in a rect by 3:

```
put rect(60, 40, 120, 200) / 3
-- rect(20, 13, 40, 66)
```

{button See also,AL(`Lingo_rect')}

rect of member

Syntax: `the rect of member` *whichCastmember*

This function gives the left, top, right, and bottom coordinates for the rectangle of any graphic cast member such as a bitmap, shape, movie, or digital video. The coordinates are returned as a rect.

The `rect of member` property is measured from the upper left corner of the cast member's bounding rectangle, not from the upper left corner of the Stage or of the cast member in the Paint window. The upper left corner of the bounding rectangle is given as (0, 0).

For an Xtra cast member that has an image, the `rect of member` is a rect that has its upper left corner at (0,0). This differs from Director 5, which returned a rect whose registration point was at (0,0).

The `rect of member` property can be tested. It can be set for field cast members only.

Example 1:

This statement displays the coordinates of bitmap cast member 20:

```
put the rect of member 20
```

Example 2:

This statement sets the coordinates of bitmap cast member Banner:

```
set the rect of member "Banner" = rect(100, 150, 300, 400)
```

{button See also,AL(`Lingo_rect_of_member')}

rect of sprite

Syntax: `the rect of sprite whichSprite`

This function gives the left, top, right, and bottom coordinates for the rectangle of any graphic sprite such as a bitmap, shape, movie, or digital video. The coordinates are returned as a rect.

The `rect of sprite` property can be tested and set.

Example:

This statement displays the coordinates of bitmap sprite 20:

```
put the rect of sprite 20
```

rect of window

Syntax: `the rect of window` *whichWindow*

This window property determines the left, top, right, and bottom coordinates of the window specified by *whichWindow*. The coordinates are given as a rect.

The `rect of window` property can be tested and set.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

Example:

This statement displays the coordinates of the window Control Panel:

```
put the rect of window "Control Panel"
```

regPoint of member

Syntax: `the regPoint of member` *whichCastMember*

This cast member property specifies the registration point of a bitmap cast member. The registration points are listed as horizontal and vertical coordinates in a point that has the form `point` (*horizontal, vertical*).

You can use the `regPoint` property to animate individual graphics within a film loop by changing a cast member's `regPoint` in relation to other objects on the Stage.

The `regPoint of member` property can be tested and set.

Example 1:

This statement displays the registration points of the bitmap cast member Desk in the Message window:

```
put the regPoint of member "Desk"
```

Example 2:

This statement changes the registration points of the bitmap cast member Desk to the values in the list:

```
set the regPoint of member "Desk" = ¬
   point(300, 400)
```

relative

See: @ (pathname operator)

repeat while

Syntax:

```
repeat while testCondition
    [statements...]
end repeat
```

This keyword structure repeatedly executes the statements as long as the condition specified by *testCondition* is TRUE. Some possible uses for this structure are for 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 clicks or releases the mouse button.

Only one handler runs 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.

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.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

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
```

{button See also,AL(`Lingo_repeat_while')}

repeat with

Syntax: `repeat with` *counter* `=` *start* `to` *finish*

[statements...]

`end repeat`

This keyword structure executes the Lingo specified by *statements* 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 goes through the repeat loop.

The `repeat with` structure is useful for repeatedly applying the same effect to a series of sprites or calculating a series of numbers, such as a number, to some exponent.

Only one handler runs 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.

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.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

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
```

{button See also,AL(`Lingo_repeat_with')}

repeat with...down to

Syntax: `repeat with` *variable* `=` *startValue* `down to` *endValue*

This keyword counts down by increments of 1 from *startValue* to *endValue*.

Only one handler runs 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.

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.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

Example:

This handler contains a repeat loop that counts down from 20 to 15:

```
on countDown
 repeat with i = 20 down to 15
  set the memberNum of sprite 6 to (10 + i)
  updateStage
 end repeat
```

repeat with...in list

Syntax: `repeat with` *variable* `in` *someList*

This keyword assigns successive values from the specified list to the variable.

Only one handler runs 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.

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.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

Example:

This statement displays four values in the Message window:

```
repeat with x in [1, 2, 3, 4]
   put i
end repeat
```

restart

Syntax: `restart`

This command restarts the Macintosh computer. It is equivalent to choosing Restart in the Macintosh Finder's Special menu. The restart command has no effect in Windows.

Example:

This statement restarts the Macintosh when the user presses Command-period:

```
if the key = "r" and the commandDown then restart
```

{button See also,AL(`Lingo_restart')}

result

Syntax: `the result`

This function gives 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.

Example 1:

The following handler returns a random roll for two dice:

```
on diceRoll
   return random(6) + random(6)
end diceRoll
```

Example 2:

The two statements:

```
diceRoll
   set roll to the result
```

are equivalent to this statement:

```
set roll to diceRoll()
```

Example 3:

Note that

```
set roll to diceRoll
```

does not call the handler because there are no parentheses following `diceRoll`; `diceRoll` here is considered a variable reference.

{button See also,AL(`Lingo_result')}

return

Syntax: `return` *expression*

This keyword is used in handlers that return values. It returns the value of *expression* and exits from the handler. The expression can be an integer, floating-point number, string, object, or symbol.

When calling a handler that serves as a user-defined function and has a return value, you must use parentheses around the argument list. This is necessary even when there are no arguments, as in the `diceRoll` function handler discussed under the entry for the `result` function.

Think of `return` as similar to the `exit` command, except that it also gives back a value to whatever called the handler. The `return` command in a handler immediately exits that handler, but it can return a value to the Lingo that called it.

Working on both object-oriented scripting and the `return` function at the same time can be difficult to understand at first. It's easier to work first with `return` for making functions and exiting handlers. Later, it will be more obvious that the `return me` line in an `on new` handler is a way to pass back a reference to the object that was created so that it can be assigned to a variable name.

The `return` function isn't the same as the character constant `RETURN`, which is a carriage return. The usage 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.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

Example 1:

This handler returns a random multiple of five between five and a hundred:

```
on GetRandomScore
  set theScore to 5 * random(20)
  return theScore
end GetRandomScore
```

You would call this handler with a statement similar to the following:

```
set thisScore to GetRandomScore()
```

In this example, the variable `thisScore` is assigned the returned value from the function `GetRandomScore`. The same thing happens with a parent script-by returning the object reference, the variable name in the calling code provides a handle to subsequently refer to that object.

Example 2:

In Windows, this statement creates a two-character string named CRLF that provides the additional line feed:

```
set CRLF = RETURN&numToChar(10)
```

{button See also,AL(`Lingo_return')}

RETURN

Syntax: `RETURN`

This character constant represents the Return key.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example:

This statement causes a paused movie to continue when the user presses the Return key:

```
if the key = RETURN then go to the frame + 1
```

This statement uses the Return character constant to insert a return between two lines in an alert:

```
alert "Last line in the file." & RETURN & ¬
   "Click OK to exit."
```

In Windows, writing to a file requires 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:

```
set CRLF = Return&numToChar(10)
```

right of sprite

Syntax: `the right of sprite` *whichSprite*

This sprite property indicates the number of pixels that the right edge of the sprite specified by *whichSprite* is from the left edge of the Stage.

The `right of sprite` property can be tested but not set directly. Edit the dimensions of a sprite by editing the sprite's `rect` property.

Note: Sprite coordinates are expressed relative to the upper left corner of the Stage.

Example:

This statement calls the handler `offRightEdge` when the right edge of sprite 3 is past the right edge of the Stage:

```
if the right of sprite 3 > (the stageRight ¬
   - the stageLeft) then offRightEdge
```

{button See also,AL(`Lingo_right_of_sprite')}

rightMouseDown

Syntax: `the rightMouseDown`

This system property indicates the current state of the right mouse button on a Windows computer. On the Macintosh, if the `emulateMultiButtonMouse` property is set to TRUE, this property indicates whether the user is pressing the mouse button and the Control key.

- When `rightMouseDown` is TRUE, the right mouse button (Windows) or the mouse button and Control key (Macintosh) is being pressed. (On the Macintosh, `rightMouseDown` is TRUE only if the `emulateMultiButtonMouse` property is TRUE).
- When `rightMouseDown` is FALSE, the right mouse button (Windows) or the mouse button and Control key (Macintosh) is not being pressed.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example:

This statement checks whether the right mouse button is being pressed and plays the sound Oops in Sound Channel 2 if it is:

```
if the rightMouseDown then puppetSound 2, "Oops"
```

rightMouseUp

Syntax: `the rightMouseUp`

This system property indicates the current state of the right mouse button on a Windows computer. On the Macintosh, if the `emulateMultiButtonMouse` property is set to TRUE, this property indicates whether the user is pressing the mouse button and the Control key.

- When `rightMouseUp` is `TRUE`, the right mouse button (Windows) or the mouse button and Control key (Macintosh) is currently not being pressed. (On the Macintosh, `rightMouseDown` is TRUE only if the `emulateMultiButtonMouse` property is TRUE).
- When `rightMouseUp` is `FALSE`, the right mouse button (Windows) or the mouse button and Control key (Macintosh) is currently being pressed.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example:

This statement checks whether the right mouse button is released and plays the sound Click Me if it is:

```
if the rightMouseUp then puppetSound 2, "Oops"
```

rollOver

Syntax: `rollOver(`*whichSprite*`)`

or

`the rollover`

This function indicates whether the cursor is currently over the bounding rectangle of the sprite specified by *whichSprite*.

- When the cursor is currently over the sprite, `rollOver` returns `TRUE` (`1`).
- When the cursor isn't currently over the sprite, `rollOver` returns `FALSE` (`0`).

The rollover has two possible syntaxes.

- 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. It 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 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 `the rollover` can change while Lingo is running a handler. You can make sure that a handler uses a consistent rollover value by assigning `the rollover` to a variable when the handler starts.

When the cursor is over the location of a sprite that has been removed, the rollover still occurs. Avoid this problem by not performing rollovers over these locations or by relocating the sprite up above the menu bar before deleting it.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example 1:

This statement changes the content of field cast member Message to "This is the place." when the cursor is over sprite 6:

```
if rollOver(6) then ¬

put "This is the place." into field "Message"
```

Example 2:

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 roll the mouse:

```
on exitFrame
  set currentLocation = the rollover
  case rollover(currentLocation) of
    1: go to frame "Left"
    2: go to frame "Middle"
    3: go to frame "Right"
  end case
end exitFrame
```

{button See also,AL(`Lingo_rollOver')}

romanLingo

Syntax: `the romanLingo`

This property specifies whether Lingo uses a single-byte or double-byte interpreter.

- When `the romanLingo` is TRUE, Lingo uses a single-byte interpreter.
- When `the romanLingo` is FALSE, Lingo uses a double-byte interpreter.

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, `the romanLingo` is set when starting up Director 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 to get faster execution of your Lingo scripts.

Example:

This statement sets `the romanLingo` to TRUE, which causes Lingo to use a single-byte character set:

```
set the romanLingo to TRUE
```

runMode

Syntax: `the runMode`

This function returns the run mode of a movie. Possible values are:

`Author`	The movie is running in Director.
`Projector`	The movie is running as a projector.
`Plugin`	The movie is running as a Shockwave plug-in or other scripting environment such as LiveConnect or Active X.

Example:

This statement determines whether or not external parameters are available and obtains them if they are:

```
if the runMode = "Plugin" then
   -- decode the embed parameter
   if externalParamName(swURL) = swURL then
      put externalParamValue(swURL) into myVariable
   end if
end if
```

runPropertyDialog

See **on runPropertyDialog**.

sampleRate of member

Syntax: `sampleRate of member "`*whichCastmember*`"`

This property of the Shockwave Audio (SWA) streaming cast member returns the sample rate of the original file that has been SWA-encoded. This property is only available after the SWA sound is playing or after the file has been preloaded using the `preLoadBuffer` command. The value is in samples per second.

This property can be tested but not set.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

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:

```
put the sampleRate of member "Paul Robeson" into member "Sound Quality"
```

sampleSize of member

Syntax: `the sampleSize of member` *whichCastmember*

This sound cast member property determines the sample size of the specified cast member. The result is usually 8- or 16-bit. This property can 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`:

```
set soundSize = the sampleSize of member "Voice Over"
```

save castLib

Syntax: `save castLib` *whichCast* `{,` *pathName*`:`*newFileName*`}`

This command saves any changes to the cast. Including the optional *pathName*:*newFileName* parameter saves the file to a new file that uses the *pathName*:*newFileName* parameter as its file name. (When the *pathName*:*newFileName* parameter isn't included, changes to the cast are saved in the cast's original file.) This command does not work with compressed files.

The `saveCastLib` command doesn't support URLs as file references.

Example:

This statement causes Director to save the revised version of the cast Buttons in the new file `UpdatedButtons` in the same folder:

```
save castLib "Buttons" , "UpdatedButtons"
```

{button See also,AL(`Lingo_save_castLib')}

saveMovie

Syntax: `saveMovie [`*pathName*`:`*fileName*`]`

This 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 `saveMovie` command doesn't support URLs as file references.

Example:

This statement saves the current movie to the file `Update`:

```
saveMovie "Update"
```

score

Syntax: `the score`

This movie property determines which Score is associated with the current movie. The property can be tested and set.

Example:

This statement assigns the film loop cast member Waterfall to the Score of the current movie:

```
set the score to the media of member "Waterfall"
```

scoreColor of sprite

Syntax: `the scoreColor of sprite` *whichSprite*

This 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.

The `scoreColor of sprite` property can be tested and set.

Example:

This statement causes the Message window to display the value for the Score color assigned to sprite 7:

```
put the scoreColor of sprite 7
```

scoreSelection

Syntax: `the scoreSelection`

This movie property determines which channels are selected in the Score window. The selection is in a list consisting of the starting channel number, the ending channel number, the starting frame number, and the ending frame number. Specify sprite channels by their channel number. Use the following numbers to specify the other channels:

To specify:	Use:
Frame script channel	0
Sound channel 2	-1
Sound channel 1	-2
Transition channel	-3
Palette channel	-4
Tempo channel	-5

You can select discontinuous channels, but you can't select discontinuous frames. This property can be tested and set.

Example 1:

This statement selects sprite channels 15 through 25 in frames 100 through 200:

```
set the scoreSelection = [[15, 25, 100, 200]]
```

Example 2:

This statement selects sprite channels 15 through 25, and sprite channels 40 through 50, in frames 100 through 200:

```
set the scoreSelection = [[15, 25, 100, 200] , [40, 50, 100, 200]]
```

Example 3:

This statement selects the frame script in frames 100 through 200:

```
set the scoreSelection = [[0, 0, 100, 200]]
```

script of menuItem

Syntax: `the script of menuItem` *whichItem* `of menu` *whichMenu*

This menu item property 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 the menu is installed, the script is set to the text following the "Å" character in the menu definition.

The script property can be tested and set.

For more information about user interfaces, see Chapter 9, "Creating User Interfaces," in *Learning Lingo*.

Example:

This statement makes the handler named `goHandler` the handler that is executed when the user chooses the command Go from the custom menu Control:

```
set the script of menuItem "Go" of menu "Control" ¬
   to "goHandler"
```

{button See also,AL(`Lingo_script_of_menuItem')}

scriptInstanceList of sprite

Syntax: `the scriptInstanceList of sprite` *spriteNum*

This sprite property gives a list of script references attached to a sprite. It is useful for:

- Attaching a behavior to a sprite.
- Determining which behaviors are attached to a sprite.
- Finding a behavior script reference to use with the call command.

The value of `the scriptInstanceList` is a list.

This property can be tested and set.

Example 1:

This handler displays the list of script references attached to a sprite:

```
on showScriptRefs me
  put the scriptInstanceList of sprite the spriteNum of me
end
```

Example 2:

These statements attach the script Big Noise to sprite 5:

```
set x = new(script "Big Noise")¬
   add(the scriptInstanceList of sprite 5,x)
```

{button See also,AL(`Lingo_scriptInstanceList')}

scriptNum of sprite

Syntax: `scriptNum of sprite` *whichSprite*

This sprite property indicates the number of the script attached to the sprite specified by *whichSprite*. If the sprite has multiple scripts attached,
`the scriptNum of sprite` 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.)

The `scriptNum of sprite` property can be tested, but not set.

Example:

This statement displays the number of the script attached to sprite 4:

```
put the scriptNum of sprite 4
```

scriptsEnabled of member

Syntax: `the scriptsEnabled of member` *whichCastmember*

This movie cast member property determines whether scripts in a linked movie are enabled.

- When `the scriptsEnabled of member` is TRUE (1), the linked movie's scripts are enabled.
- When `the scriptsEnabled of member` is FALSE (0), the linked movie's scripts aren't enabled.

This property is available for Director movie cast members only. Although the property can be tested and set for Director movies, it can't be tested or set for other cast members.

Example:

This statement turns off scripts in the linked movie Jazz Chronicle:

```
set the scriptsEnabled of member "Jazz Chronicle" = FALSE
```

scriptText of cast

This is obsolete. Use **scriptText of member** instead.

scriptText of member

Syntax: `the scriptText of member` *whichCastmember*

This 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 the movie is converted into a projector or compressed for Shockwave. Such movies lose their values for the `scriptText of member` property. Therefore, the movie's values for `the scriptText of member` can't be used by a projector. However, Director can set new values for `the scriptText of member` inside the projector.

The `scriptText of member` property can be tested and set.

For more information about authoring from Lingo, see Chapter 13, "Authoring from Lingo," in *Learning Lingo*.

Example:

This statement makes the contents of field cast member 20 the script of cast member 30:

```
set the scriptText of member 30 = the text of member 20
```

scriptType of member

Syntax: `the scriptType of member` *whichScript*

This script cast member property indicates the specified script's type. Possible values are `#MOVIE`, `#SCORE`, and `#PARENT`.

For more information about authoring from Lingo, see Chapter 13, "Authoring from Lingo," in *Learning Lingo*.

Example:

This statement makes script member Main Script a movie script:

```
set the scriptType of member "Main Script" to #movie
```

{button See also,AL(`Lingo_scriptType_of_member')}

scrollByLine

Syntax: `scrollByLine member` *whichCastmember, amount*

This command scrolls the specified field cast member up or down by the number of lines specified in *amount*. (Lines are lines separated by carriage returns, not lines caused by line wrapping.)

- When *amount* is positive, the field scrolls down.
- When *amount* is negative, the field scrolls up.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example 1:

This statement scrolls the field cast member Today's News down five lines:

```
scrollByLine member "Today's News", 5
```

Example 2:

This statement scrolls the field cast member Today's News up five lines:

```
scrollByLine member "Today's News", -1
```

scrollByPage

Syntax: `scrollByPage member` *whichCastMember, amount*

This command scrolls the specified field cast member up or down by the number of pages specified in *amount*.

- When *amount* is positive, the field scrolls down.
- When *amount* is negative, the field scrolls up.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example 1:

This statement scrolls the field cast member Today's News down one page:

```
scrollByPage member "Today's News", 1
```

Example 2:

This statement scrolls the field cast member Today's News up one page:

```
scrollByPage member "Today's News", -1
```

scrollTop of member

Syntax: `the scrollTop of member` *whichCastmember*

This property of text and field cast members 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 `the scrollTop of member` while the movie plays, you can change the section of the field that appears in the scrolling field.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example:

This repeat loop makes the field Credits appear to accelerate scrolling by continuously increasing the value of `the scrollTop of member`:

```
set the scrollTop of member "Credits" = 1
repeat with count = 1 to 150
   set the scrollTop of member "Credits" = the scrollTop ¬
   of member "Credits" + count
end repeat
```

searchCurrentFolder

Syntax: `the searchCurrentFolder`

This global property determines whether Director searches the current folder when searching file names.

- 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.

The `searchCurrentFolder` property can be tested and set.

Example 1:

This statement causes the Message window to display whether the `searchCurrentFolder` property is on:

```
put the searchCurrentFolder
```

The result is the number 1, which is the numeric equivalent of TRUE.

Example 2:

This statement sets the `searchCurrentFolder` property to TRUE, which causes Director to search the current folder when resolving file names:

```
set the searchCurrentFolder to TRUE
```

searchPath

Syntax: `the searchPath`

This property provides a list of the pathnames that are searched when Director resolves file names. When Director cannot find the file in the current folder, it searches for it in the folders listed in `searchPath`.

The `searchPath` property doesn't support URLs as file references.

The `searchPath` content is a regular list that you can handle the same as any other list by using commands such as `add`, `addAt`, `append`, `deleteAt`, and `setAt`. Items in the list are separated by commas. Trailing colons and backslashes are allowed but not necessary.

Adding a large number of paths to `the searchPath` slows searching. Try to minimize the number of paths in the list.

The `searchPath` property works the same as `the searchPaths`. It can be tested and set.

Setting the value of `searchPath` automatically changes the value of `searchPaths` also.

Example 1:

This statement displays the pathnames that Director searches when resolving file names:

```
put the searchPath
```

Example 2:

This statement assigns two folders to `the searchPath` in Windows. This version includes optional trailing backslashes:

```
set the searchPath = ["c:\director\projects\",¬ "d:\cdrom\sources\"]
```

This statement is the same, except that trailing backslashes have been omitted:

```
set the searchPath = ["c:\director\projects", "d:\cdrom\sources"]
```

Example 3:

This statement assigns two folders to `the searchPath` on a Macintosh. This version includes optional trailing colons:

```
set the searchPath = ["hard drive:director:projects:",¬
"cdrom:sources:"]
```

This statement is the same, except that trailing colons have been omitted:

```
set the searchPath = ["hard drive:director:projects",¬
"cdrom:sources"]
```

{button See also,AL(`Lingo_searchPath')}

searchPaths

Syntax: `the searchPaths`

This global property is a list of paths that Director searches. Each item in the list is a fully qualified pathname as it appears on the current platform at runtime.

The value of `searchPaths` is a regular list that you can handle the same as any other list by using commands such as `add`, `addAt`, `append`, `deleteAt`, and `setAt`.

The `searchPaths` property doesn't support URLs as file references.

Adding a large number of paths to `the searchPaths` slows searching. Try to minimize the number of paths in the list.

The `searchPath` property works the same as `the searchPaths`. It can be tested and set.

Setting the value of `searchPaths` automatically changes the value of `searchPath` also.

Example:

These statements cause Director to search inside a folder named Sounds, which is in the same folder as the current Director movie:

```
put the moviePath & "Sounds" into soundsPath

add the searchPaths, soundPath
```

{button See also,AL(`Lingo_searchPaths')}

selection

Syntax: `the selection`

This function returns a string containing the highlighted portion of the current editable field. It is useful for testing what a user has selected in a field.

The `selection` function only determines which string of characters are selected; you cannot use `the 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."
```

{button See also,AL(`Lingo_selection')}

selection of castLib

Syntax: `the selection of castLib` *whichCast*

Or

`set the selection of castLib` *whichCast* `=` `[`*startMember1*`,` ¬
endMember1`]` `,` `{[`*startMember2*`,` *endMember2*`]` `,` ¬
`[`*startMember3*`,` *endMember3*`]...}`

This cast property determines which cast members are selected in the specified Cast window. The specified range appears as a list of the starting and ending cast member numbers for the selected range. You can specify more than one selection by specifying additional ranges of cast members. (Specifying more than one selection is done by dragging while pressing the Control key (Windows) or the Command key (Macintosh). This property can be tested and set.

Example:

This statement selects cast members 1 through 10 in `castLib` number 1:

```
set the selection of castLib 1 = [1, 10]
```

This statement selects cast members 1 through 10, and 30 through 40, in `castLib` number 1:

```
set the selection of castLib 1 = [1, 10], [30,40]
```

selEnd

Syntax: `the selEnd`

This field property specifies the ending character of a selection. It is used with `the selStart` to determine a selection from the currently editable field, counting from the beginning character.

The `selEnd` field property can be tested and set, and the default value is 0.

Example 1:

These statements select "cde" from the field "abcdefg":

```
set the selStart to 3
set the selEnd to 5
```

Example 2:

This statement calls the handler `noSelection` when `the selEnd` is the same `as the selStart`:

```
if the selEnd = the selStart then noSelection
```

Example 3:

This statement makes a selection 20 characters long:

```
set the selEnd to the selStart + 20
```

{button See also,AL(`Lingo_selEnd')}

selStart

Syntax: `the selStart`

This field property specifies the starting character of a selection. It is used with `the selEnd` to determine a selection from the currently editable field, counting from the beginning character.

The `selStart` field property can be tested and set. The default value is 0.

Example 1:

These statements select "cde" from the field "abcdefg":

```
set the selStart to 3
set the selEnd to 5
```

Example 2:

This statement calls the handler `noSelection` when `the selEnd` is the same as `the selStart`:

```
if the selEnd = the selStart then noSelection
```

Example 3:

This statement makes a selection 20 characters long:

```
set the selEnd to the selStart + 20
```

{button See also,AL(`Lingo_selStart')}

sendAllSprites

Syntax: `sendAllSprites (#`*customEvent, args*`)`

This command makes a custom event message available to all sprites, not just the sprite that was involved in the event. As with any other message, the message is available to every script attached to the sprite, unless the `stopEvent` command is used.

The symbol operator (`#`) must precede the custom event message when you write this command.

After the message has passed to all sprites, the event follows the regular message hierarchy: script of cast member, frame script, and then movie script.

When you use the `sendAllSprites` command:

- Replace `#`*customEvent* with the message.
- Replace *args* with any arguments to be sent with the message.

Example:

This handler sends the custom message `bumpCounter` and the argument `2` to all sprites when the user clicks the mouse:

```
on mouseDown me

  sendAllSprites (#bumpCounter, 2)

end
```

{button See also,AL(`Lingo_sendAllSprites')}

sendSprite

Syntax: `sendSprite (`*whichSprite*`, #`*customMessage*`, `*args*`)`

This command sends a custom event message to all scripts attached to a specified sprite.

The symbol operator (`#`) must precede the custom message when you write this command.

Messages sent using `sendSprite` are available to all scripts attached to the sprite . The messages then follow the regular message hierarchy: script of the cast member, frame script, and movie script.

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
```

{button See also,AL(`Lingo_sendSprite')}

set...to, set...=

Syntax: `set the` *property* `to` *expression*

or

`set the` *property* `=` *expression*

or

`set` *variable* `to` *expression*

or

`set` *variable* `=` *expression*

This command evaluates the expression specified by *expression* and puts the result into the property specified by *property* or the variable specified by *variable*.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

Example 1:

This statement sets the ink effect for sprite 3 to the ink effect specified by the number 8:

```
set the ink of sprite 3 to 8
```

Example 2:

This statement sets the `soundEnabled` property to the opposite of its current state. When `the soundEnabled` is TRUE (the sound is on), this statement turns it off. When `the soundEnabled` is FALSE (the sound is off), this statement turns it on.

```
set the soundEnabled = not (the soundEnabled)
```

Example 3:

This statement sets the variable `vowels` to the string "aeiou":

```
set vowels to "aeiou"
```

{button See also,AL(`Lingo_set_to')}

setaProp

Syntax: `setaProp` *list*, *property*, *newValue*

or

`setaProp` (*childObject*, *property*, *newValue*)

This command replaces the value assigned to *property* with the value specified by *newValue*. The `setaProp` command works with property lists only. 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.
- The `setaProp` command can also set `ancestor` properties.

For more information about lists, see Chapter 10, "Working with Lists," in *Learning Lingo*.

Example:

These statements create a property list and then add the item #c:10 to the list:

```
set newList = [#a:1, #b:5]
put newList
-- [#a:1, #b:5]
setaProp newList, #c, 10
put newList
-- [#a:1, #b:5, #c:10]
```

{button See also,AL(`Lingo_setaProp')}

setAt

Syntax: `setAt` *list*, *orderNumber*, *value*

This 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 linear list, Director expands the list's blank entries to provide the number of places specified by *orderNumber*.
- When *orderNumber* is greater than the number of items in a property list, the `setAt` command gives a script error.

For more information about lists, see Chapter 10, "Working with Lists," in *Learning Lingo*.

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]

setButtonImageFromCastMember

Syntax: `setButtonImageFromCastMember (buttonCastMember, ¬ "imageString", bitmapCastMember)`

This function assigns a bitmap to an image string in a button cast member.

When you use this function:

- Replace *bitmapCastMember* with the name of the bitmap cast member.
- Replace *imageString* with the name of the image string in the button cast member.
- Replace *buttonCastMember* with the name of the button cast member.

Example:

This statement takes the bitmap cast member Coolness of castLib 2 and puts it into the image named imageNormal in the Button editor residing in button cast member Panic:

```
putImageIntoCastMember(member "Coolness", "imageNormal", ¬
member "Panic" of castLib 2)
```

setCallBack

Syntax: `setCallBack` *XCMDname, value*

This command is obsolete. It was used in earlier versions of Director to specify how Lingo handled unsupported callbacks from the HyperTalk XCMD or XFCN.

setPref

Syntax: `setPref` *prefName, prefValue*

This command writes the string specified by *prefValue* to the file specified by *prefName* on the computer's local disk.

prefName 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 inside a browser, a folder named Prefs is created inside the Plug-In Support folder. The `setPref` command can write only to that folder.

If the movie is playing outside a browser, a folder is created in the same folder as the application. The folder receives the name Application Folder, where *Application* is the name of the Application. For example, a projector named BigBand would have a folder named BigBand Folder.

Example:

This handler assigns the text in the field cast member Text Entry to the field cast member Current Preferences:

```
on mouseUp
   setPref the text of member "Current Preferences", ¬
   the text of member "Text Entry"
end
```

{button See also,AL(`Lingo_setPref')}

setProp

Syntax: `setProp` *list*, *property*, *newValue*

This 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 specifed property, `setProp` produces 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 this command gives an error when the property is not already in the list.

For more information about lists, see Chapter 10, "Working with Lists," in *Learning Lingo*.

Example:

This statement changes the value assigned to the age property of property list x to 11:

```
setProp x #age, 11
```

{button See also,AL(`Lingo_setProp')}

setTrackEnabled

Syntax: `setTrackEnabled(sprite` *whichSprite, whichTrack, trueOrFalse*`)`

This digital video sprite property sets whether the specified track of a digital video is enabled to play.

- When `setTrackEnabled` is TRUE, the specified track is enabled.
- When `setTrackEnabled` is FALSE, the specified track is disabled.

To test whether a track is already enabled, test the `trackEnabled of sprite` property.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement enables track 3 of the digital video assigned to sprite channel 8:

```
setTrackEnabled(sprite 8, 3, TRUE)
```

{button See also,AL(`Lingo_setTrackEnabled')}

shapeType

Syntax: `the shapeType of member` *whichCastmember*

This shape cast member property indicates the specified shape's type. Possible types are `#rect`, `#roundRect`, `#oval`, or `#line`. This property is useful for specifying a shape cast member's type after the shape cast member is created from Lingo.

For more information about authoring from Lingo, see Chapter 13, "Authoring from Lingo," in *Learning Lingo*.

Example:

These statements create a new shape cast member numbered 100 and then define it as an oval:

```
new(#shape, member 100)

set shapeType of member 100 = #oval
```

shiftDown

Syntax: `the shiftDown`

This function indicates whether the user is pressing the Shift key. It must always be tested in conjunction with another key.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

- When `the shiftDown` is TRUE, the user is pressing the Shift key.
- When `the shiftDown` is FALSE, the user is not pressing the Shift 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)
```

{button See also,AL(`Lingo_shiftDown')}

short

{button See also,AL(`Lingo_short')}

showGlobals

Syntax: `showGlobals`

This command causes the Message window to display all global variables. It is useful for debugging scripts.

Example:

This statement displays all global variables in the Message window:

```
showGlobals
```

{button See also,AL(`Lingo_showGlobals')}

showLocals

Syntax: `showLocals`

This command causes the Message window to display all local variables. This command can only be used within handlers or parent scripts.

Local variables in handlers are abandoned after the handler executes. This command is useful for debugging scripts.

Example:

This statement displays all local variables in the Message window:

```
showLocals
```

{button See also,AL(`Lingo_showLocals')}

showResFile

Syntax: `showResFile [`*whichFile*`]`

This command, when used on the Macintosh, displays a list of resources in the resource file specified by the string *whichFile*. The file must be already open. If the resource file is in a different folder than the current movie, *whichFile* must specify a pathname. If no file is specified, all open resource files are listed. In Windows, the `showResFile` command has no effect.

The `showResFile` command doesn't support URLs as file references.

There may be many open resource files, and the listing may be very long. To cancel the listing, click the mouse button.

Example:

This statement displays the resource file Special Fonts:

```
showResFile "Special Fonts"
```

{button See also,AL(`Lingo_showResFile')}

showXlib

Syntax: `showXlib [`*Xlibfilename*`]`

This command shows all Xtras and XObjects in *Xlibfilename* (it must be open), or all open Xlibraries if no file is specified. Xlibrary files are resource files that contain XCOD (XObjects) resources. If the file is in another folder than the current movie, specify the pathname.

Xlibrary files are resource files that contain XCOD (Xtras and XObjects) resources (Macintosh) or DLLs (Windows). Because the type of Xlibrary files on the Macintosh and in Windows differs, the list of files that the `showXlib` command generates can be different on different platforms.

The `showXlib` command doesn't support URLs as file references.

The `mDescribe` method displays online documentation for an XObject.

To use `mDescribe`:

1. **Type showXlib in the Message window and press the Return key.**
   This displays all open Xlibrary resource files and all Xtras and XObjects contained in those Xlibraries.

2. **Using the list of Xtras and XObjects displayed in the Message window, type `XObjectName(mDescribe)` and press the Return key.**
   This displays the on-line documentation for that XObject.

Example:

This statement displays the Xtras and XObjects in the VideoDisc Library:

```
showXlib "VideoDisc Xlibrary"
```

{button See also,AL(`Lingo_showXlib')}

shutDown

Syntax: `shutDown`

This command has different effects on the Macintosh and in Windows.

- On the Macintosh, the `shutDown` command closes all open applications and turns the computer off.
- In Windows 95, the `shutDown` command exits Director or the projector.
- In Windows 3.1, the `shutDown` command exits Director or the projector and then exits Windows.

Example:

This statement checks whether the user has pressed Control+R and shuts down the computer if he or she has:

```
if the key = "s" and the commandDown then shutDown
```

{button See also,AL(`Lingo_shutDown')}

sin

Syntax: `sin(`*angle*`)`

This 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
```

Note: The symbol ¹ cannot be used in a Lingo expression.

size of cast

This is obsolete. Use **size of member** instead.

size of member

Syntax: `the size of member` *castName*

This cast member property permits you to learn the size, in bytes, of a specific cast member number or name. Divide bytes by 1024 to convert to kilobytes.

Example:

```
put the size of member "Shrine" into member "How Big"
```

sort

Syntax: `sort` *list*

This command puts the items in the list specified by *list* 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.

For more information about lists, see Chapter 10, "Working with Lists," in *Learning Lingo.*

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]
sort Values
put Values
--[#a: 1, #c: 3, #d: 2]
```

sound close

Syntax: `sound close` *soundChannel*

This command stops the sound playing in the specified channel and then closes the sound channel specified by *soundChannel*.

The `sound close` command was used in earlier versions of Director. For best results, use the `puppetSound` command.

Example:

This statement stops any sound playing in and closes sound channel 1:

```
sound close 1
```

{button See also,AL(`Lingo_sound_close')}

sound fadeIn

Syntax: `sound fadeIn` *whichChannel*

`sound fadeIn` *whichChannel*`,` *ticks*

This 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.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement fades in the sound in channel 1 over 5 seconds:

```
sound fadeIn 1, 5 * 60
```

{button See also,AL(`Lingo_sound_fadeIn')}

sound fadeOut

Syntax: `sound fadeOut` *whichChannel*

`sound fadeOut` *whichChannel*`,` *ticks*

This 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 fadeout continues at a predetermined rate until the number of ticks has elapsed, or until the sound in the specified channel changes.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement fades out the sound in channel 1 over 5 seconds:

```
sound fadeOut 1, 5 * 60
```

{button See also,AL(`Lingo_sound_fadeOut')}

sound of cast

This is obsolete. Use **sound of member** instead.

sound of member

Syntax: `the sound of member` *whichCastmember*

This movie and digital video cast member property determines whether the sound for the specified movie or digital video plays.

- When `the sound of member` is TRUE (1), the sound plays.
- When `the sound member` is FALSE (0), the sound doesn't play.

This property can be tested and set for Director movies and digital video cast members.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement turns on the sound for the Director movie cast member Movie Clip:

```
set the sound of member "Movie Clip" to 1
```

sound playFile

Syntax: `sound playFile` *whichChannel*, *whichFile*

This command plays the AIFF or WAVE sound located at *whichFile* in the sound channel specified by *whichChannel*.

When the sound file is in a different folder than the movie, *whichFile* must specify the full pathname to the file.

To play sounds obtained from a URL, it's usually a good idea to use the `downloadNetThing` command to download the file to a local disk first. This can minimize problems with waiting for the file to download.

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.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example 1:

This statement plays the file named Thunder in channel 1:

```
sound playFile 1, "Thunder"
```

Example 2:

This statement plays the file named Thunder in channel 3:

```
sound playFile 3, the pathName &"Thunder"
```

{button See also,AL(`Lingo_sound_playFile')}

sound stop

Syntax: `sound stop` *whichChannel*

This command stops the playing of 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.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example 1:

These statements stop any sound playing in and closes sound channel 1:

```
sound stop 1
sound close 1
```

Example 2:

This statement checks whether a sound is playing in sound channel 1 and stops the sound if it is:

```
if soundBusy(1) then sound stop 1
```

{button See also,AL(`Lingo_sound_stop')}

soundBusy

Syntax: `soundBusy(`*whichChannel*`)`

This function determines whether a sound is playing in the sound channel specified by *whichChannel*.

- When a sound is playing in the specified sound channel, the `soundBusy` function returns TRUE.
- When no sound is playing in the specified sound channel, the `soundBusy` function returns FALSE.

Make sure that you allow enough time for the sound to start playing before using `soundBusy` to check the sound channel.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement checks whether a sound is playing in sound channel 1 and loops in the frame if it is. This would allow the sound to finish before the playback head goes to another frame:

```
if soundBusy(1) then go to the frame
```

{button See also,AL(`Lingo_soundBusy')}

soundChannel of member

Syntax: `the soundChannel of member "`*whichCastmember*`"`

This Shockwave Audio (SWA) streaming 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 unused sound channel.

Shockwave Audio streaming 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.

This property can be tested and set.

Example:

This statement has the SWA streaming cast member Frank Zappa play in sound channel 3:

```
set the soundChannel of member "Frank Zappa" to 3
```

soundEnabled

Syntax: `the soundEnabled`

This property determines whether the sound is on or off. TRUE means that the sound is on.

The `soundEnabled` property can be tested and set, and the default value is TRUE. When you set this property to FALSE, the volume setting of the sound is not changed, but you do not hear the sound.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement set turns to the opposite of its current setting. It turns the sound on if it is off and turns it off if it is on:

```
set the soundEnabled to not (the soundEnabled)
```

{button See also,AL(`Lingo_soundEnabled')}

soundLevel

Syntax: `the soundLevel`

This property determines the volume level of the sound that is played through the computer's speaker. Settings range from 0 (no sound) to 7 (maximum sound volume).

The `soundLevel` property can be tested and set. The default value is 7.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example 1:

This statement sets the variable `oldSound` equal to the current sound level:

```
put the soundLevel into oldSound
```

Example 2:

This statement sets the sound level to 5:

```
set the soundLevel to 5
```

{button See also,AL(`Lingo_soundLevel')}

sourceRect

Syntax: `the sourceRect of window` *whichWindow*

This window property specifies the coordinates of the rectangle that the movie that plays in the window specified by *whichWindow* was originally created for.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

Example:

This statement displays the original coordinates of the movie "Control Panel" in the Message window:

```
put the sourceRect of "Control Panel"
```

SPACE

Syntax: `SPACE`

This read-only constant represents the space character.

Example:

This statement displays "Age Of Aquarius" in the Message window:

```
put "Age"&SPACE&"Of"&SPACE&"Aquarius"
```

sprite

Syntax: `the` *property* `of sprite` *whichSprite*

This 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.

Example 1:

This statement sets the variable named `horizontal` to `the locH of sprite 1`:

```
set horizontal = the locH of sprite 1 into horizontal
```

Example 2:

This statement turns on the puppet condition for the sprite channel that has sprite number i + 1:

```
set the puppet of sprite (i + 1) to TRUE
```

{button See also,AL(`Lingo_sprite')}

sprite...intersects

Syntax: `sprite` *sprite1* `intersects` *sprite2*

This operator compares the position of two sprites. It is TRUE if the bounding rectangle of *sprite1* touches the bounding rectangle of *sprite2*.

If both sprites have matte ink, their actual outlines, not the bounding rectangles, are used. A sprite's outline is defined by the non-white pixels that make up its border.

This is a comparison operator with a precedence level of 5.

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"
```

{button See also,AL(`Lingo_sprite_intersects')}

sprite...within

Syntax: `sprite` *sprite1* `within` *sprite2*

This comparison operator compares the position of two sprites. It is TRUE if the bounding rectangle of *sprite1* is entirely inside the bounding rectangle of *sprite2*.

If both sprites have matte ink, their actual outlines, not the bounding rectangles, are used. A sprite's outline is defined by the non-white pixels that make up its border.

This is a comparison operator with a precedence level of 5.

Example:

This statement checks whether two sprites intersect and calls the handler `doInside` if they do:

```
if sprite 3 within 2 boundary ¬
   then doInside
```

{button See also,AL(`Lingo_sprite_within')}

spriteBox

Syntax: `spriteBox` *whichSprite, left, top, right, bottom*

This command is obsolete, Set **`the rect of sprite`** to change a sprite's bounding rectangle instead.

{button See also,AL(`Lingo_spriteBox')}

spriteNum

Syntax: `the spriteNum of me`

This property indicates the number of the channel the behavior's sprite is in.

If you attach behaviors to a sprite, `t he scriptInstanceList` automatically records which scripts the sprite has assigned. However, 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 which sprite the script is attached to. The sprite's number must be sent to the `on new` handler as an argument when the `on new` handler is called.

Example 1:

In this handler, the `spriteNum` property is automatically set for script instances that are created by the system:

```
on mouseDown me
  set the member of sprite the spriteNum of me = member ¬
    "DownPict"
end
```

Example 2:

This script includes an `on new` handler that explicitly sets the `spriteNum` property to identify the sprite that the script is attached to:

```
property spriteNum
on new me whichSprite
  set the spriteNum of me = whichSprite
  return me
end
```

{button See also,AL(`Lingo_spriteNum')}

sqrt

Syntax: `sqrt(`*number*`)`

`the sqrt of` *number*

This function yields the square root of the number specified by *number*.

- When *number* is a floating-point number, the result is a floating-point number.
- When *number* is an integer, the result is rounded to the nearest integer.

The value of number must be a decimal number that is greater than zero.

Example:

This statement displays the square root of 3.0 in the Message window:

```
put sqrt(3.0)
-- 1.7321
```

{button See also,AL(`Lingo_sqrt')}

stage

Syntax: `the stage`

This system property is used to refer to the main movie. This 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 movie to loop in its current frame:

```
tell the stage to go to the frame
```

This statement displays the current setting of the Stage:

```
put the rect of the stage
--rect (0, 0, 640, 480)
```

stageBottom

Syntax: `the stageBottom`

This function-along with `the stageLeft`, `the stageRight`, and `the 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 given by `the stageBottom - the stageTop`.

The `stageBottom` function can be tested but not set.

Example:

These two statements position sprite 3 a distance of 50 pixels from the bottom edge of the Stage:

```
put the stageBottom - the stageTop into ¬
   stageHeight
set the locV of sprite 3 to stageHeight - 50
```

Note: Sprite coordinates are expressed relative to the upper left corner of the Stage. See *Using Director* for more information.

{button See also,AL(`Lingo_stageBottom')}

stageColor

Syntax: `the stageColor`

This property determines the color of the movie background.

The value of the `stageColor` property ranges from 0 to 255 for 8-bit color, or 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.

Example:

This statement sets the variable `oldColor` to the index number of the current Stage color:

```
set oldColor = the stageColor into oldColor
```

This statement sets the Stage color to the color assigned to chip 249 on the current palette:

```
set the stageColor to 249
```

{button See also,AL(`Lingo_stageColor')}

stageLeft

Syntax: `the stageLeft`

This function-along with `the stageRight`, `the stageTop`, and `the stageBottom`-indicates where the Stage is positioned on the desktop. It equals 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 zero.

The `stageLeft` 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 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
```

{button See also,AL(`Lingo_stageLeft')}

stageRight

Syntax: `the stageRight`

This function-along with `the stageLeft`, `the stageTop`, and `the 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 given by `the stageRight - the stageLeft`.

The `stageRight` 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:

```
set stageWidth = the stageRight - the stageLeft

set the locH of sprite 3 to stageWidth - 50
```

{button See also,AL(`Lingo_stageRight')}

stageTop

Syntax: `the stageTop`

This function-along with `the stageBottom`, `the stageLeft`, and `the 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 zero.

The `stageTop` function can be tested but not set.

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
```

Sprite coordinates are expressed relative to the upper left corner of the Stage.

{button See also,AL(`Lingo_stageTop')}

startMovie

See: on startMovie movie handler.

starts

Syntax: *string1* `starts` *string2*

This comparison operator compares two strings.

- When *string1* starts with *string2*, the condition is TRUE (1).
- When *string1* does not start with *string2*, the condition is FALSE (0).

The string comparison is not sensitive to case or diacritical marks; "a" and "□" are considered the same.

This is a comparison operator with a precedence level of 1.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example:

This statement has the Message window display whether the word Macromedia starts with the string Macro:

```
put "Macromedia" starts "Macro"
```

The result is 1, which is the numerical equivalent of TRUE.

{button See also,AL(`Lingo_starts')}

startTime of sprite

Syntax: `the startTime of sprite` *whichSprite*

This sprite property determines when the specified digital video sprite begins. The value of `the startTime` is measured in ticks.

It can be tested and set.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement starts the digital video sprite in channel 5 at 100 ticks into the digital video:

```
set the startTime of sprite 5 to 100
```

startTimer

Syntax: `startTimer`

This command sets the timer property to zero. It also resets all the accumulating timers for the `lastClick`, `lastEvent`, `lastKey`, and `lastRoll` functions to zero.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example:

This handler sets the timer to zero when a key is pressed:

```
on keyDown

   startTimer

end keyDown
```

{button See also,AL(`Lingo_startTimer')}

state of member

Syntax: `state of member` *"whichCastmember"*

This Shockwave Audio (SWA) streaming cast member property determines the current state of the SWA streaming file. The properties `streamName`, `URL`, and `preLoadTime` can only be changed when the SWA sound is stopped. The following properties for the SWA file return meaningful information only after the file is streaming: `cuepointNames`, `cuepointTimes`, `currentTime`, `duration`, `percentPlayed`, `percentStreamed`, `bitRate`, `sampleRate`, and `numChannels`.

The following are possible values and their corresponding meanings for the state of member:

0	Stopped
1	Preloading
2	Preload completed successfully
3	Playing
4	Paused
5	Done
9	Error
10	Insufficient CPU

Example:

This statement issues an alert if an error is detected for the SWA streaming cast member:

```
on mouseDown
   if the state of member "Ella Fitzgerald" = 9 then
      alert "Sorry, can't find an audio file to stream."
   end if
end
```

stillDown

Syntax: `the stillDown`

This function indicates whether the user is pressing the mouse button.

- When the user is pressing the mouse button, `the stillDown` is TRUE.
- When the user is not pressing the mouse button, `the stillDown` is FALSE.

This function is useful within a `mouseDown` script to test whether the mouse button is still being pressed after Director responds to the initial mouse press.

Lingo cannot test `the stillDown` when it is used inside a repeat loop. Use the `mouseDown` function inside of repeat loops instead.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

For a demonstration of modifier keys, see the sample movie **Keyboard Lingo**.

Example:

This statement checks whether the mouse button is being pressed and calls the handler `dragProcedure` if it is:

```
if the stillDown then dragProcedure
```

{button See also,AL(`Lingo_stillDown')}

stopEvent

Syntax: `stopEvent`

This command prevents Lingo from passing an event message to subsequent locations in the message hierarchy. This does the same as the `dontPassEvent` command used in earlier versions of Director, except that it now applies to sprite scripts also.

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.

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.

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.

For more information, see Chapter 2, "Script Basics," in *Learning Lingo*.

Example:

```
on mouseUp me
  global grandTotal
  if grandTotal = 500 then
    stopEvent
  end if
end
```

{button See also,AL(`Lingo_stopEvent')}

stop member

Syntax: `stop member ("whichCastmember")`

The command stops the playback of a Shockwave Audio (SWA) streaming cast member. When the cast member is stopped, the `state of member` property equals 0.

To change properties such as `streamName`, `preLoadTime`, and `URL of member`, the SWA streaming cast member must be stopped.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement stops SWA cast member Big Band from playing.

```
stop member ("Big Band")
```

stopTime of sprite

Syntax: `the stopTime of sprite` *whichSprite*

This sprite property determines when the specified digital video sprite stops. The value of `the stopTime` is measured in ticks.

It can be tested and set.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement stops the digital video sprite in channel 5 at 100 ticks into the digital video:

```
set the stopTime of sprite 5 to 100 on streamStatus
```

streamName of member

Syntax: `the streamName of member` "*whichCastmember*"

This Shockwave Audio (SWA) property specifies a URL or file name for a SWA streaming cast member. It has the same functionality as URL of member.

This property can be tested and set.

Example:

This statement links the file BigBand.swa to a SWA streaming cast member. The linked file is on the disk MyDisk within the folder named Sounds.

```
set the streamName of member "SWAstream" to ¬
"MyDisk/sounds/BigBand.swa"
```

stretch of sprite

Syntax: `the stretch of sprite` *whichSprite*

This sprite property was introduced in an earlier version of Director and used in conjunction with the `spriteBox` command. To set the rect of a sprite, it's now easier to use the `rect` sprite property.

The `stretch of sprite` property determines whether the sprite specified by *whichSprite* can be stretched by using the `spriteBox` command or the `width of sprite` and `height of sprite` properties. If it is TRUE, the bitmap sprite can be stretched.

The `stretch of sprite` property can be tested and set, and the default value is FALSE. When FALSE, the bitmap sprite always remains at its default or normal size.

The `stretch of sprite` property applies to bitmap, digital video, film loop, linked movie, OLE and Xtra cast members that have a graphic image, and picture cast members, but not to shape, field, or button sprites.

Shapes can be stretched at any time by setting their `height of sprite` and `width of sprite` properties, regardless of the setting of their `stretch` property. Field and button cast members cannot be stretched in any case.

Director requires much more processor time to draw stretched sprites than regular sprites, which can affect movie performance.

For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet.

Example:

This statement checks whether sprite 3 is stretchable and sets the sprite's width to 10 pixels if it is:

```
if the stretch of sprite 3 = TRUE then ¬
   set the width of sprite 3 to 10
```

{button See also,AL(`Lingo_stretch_of_sprite')}

string

Syntax: `string(`*expression*`)`

This function converts an integer, floating-point number, or symbol expression to a string.

Example 1:

This statement adds 2.0 + 2.5 and inserts the results into field cast member Total:

```
set field "Total" = string(2.0 + 2.5)
```

Example 2:

This statement converts the symbol #red to a string and inserts it in the field cast member Color:

```
set field "Color" = string(#red)
```

{button See also,AL(`Lingo_string')}

stringP

Syntax: `stringP(`*expression*`)`

This function determines whether the expression specified by *expression* is a string.

- When *expression* is a string, the result is TRUE.
- When *expression* is not a string, the result is FALSE.

The "P" in `stringP` stands for predicate.

Example 1:

This statement checks whether "3" is a string:

```
put stringP("3")
```

The result is 1, which is the numeric equivalent of TRUE.

Example 2:

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.

{button See also,AL(`Lingo_stringP')}

switchColorDepth

Syntax: `the switchColorDepth`

This property, when the movie plays on the Macintosh, determines whether Director automatically switches the color depth when loading a movie. The `switchColorDepth` property has no effect in Windows.

- When `the switchColorDepth` is TRUE, Director switches the monitor(s) that the Stage occupies to the color depth of the movie that is being loaded.
- When `the switchColorDepth` is FALSE, Director leaves the color depth of the monitor(s) unchanged when a movie is loaded.

When `the 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 color depth lets you use the minimum amount of memory to play movies. At minimum memory, loading cast members is more efficient and animation can occur faster.

The `switchColorDepth` property can be tested and set. The default value is the setting for the Reset Monitor to Movie's Color Depth check box in the **General Preferences** dialog box.

Example 1:

This statement sets the variable named `switcher` to the current setting of `switchColorDepth`:

```
put the switchColorDepth into switcher
```

Example 2:

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 ¬
   set the switchColorDepth to TRUE
```

{button See also,AL(`Lingo_switchColorDepth')}

symbol

Syntax: `symbol(`*stringValue*`)`

This function takes a string, *stringValue*, and returns a symbol.

Example 1:

This statement displays the symbol #hello.

```
put symbol("hello")
-- #hello
```

Example 2:

This statement displays the symbol #goodbye.

```
set x = "goodbye"
put symbol(x)
-- #goodbye
```

symbolP

Syntax: `symbolP(`*expression*`)`

This function determines whether the expression specified by *expression* is a symbol.

- When *expression* is a symbol, the result is TRUE.
- When *expression* is not a symbol, the result is FALSE.

The "P" in `symbolP` stands for predicate.

Example:

This statement checks whether #3 is a symbol:

```
put symbolP(#3)
```

TAB

Syntax: `TAB`

This character 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
```

This statement advances or retreats the playback head, 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
```

{button See also,AL(`Lingo_TAB')}

tan

Syntax: `tan(`*angle*`)`

This function yields the tangent of the specified angle. The angle must be expressed in radians as a floating-point number.

Example:

The following function yields the tangent of ¹/4:

```
tan (pi()/4.0) = 1
```

Note: The ¹ symbol cannot be used in a Lingo expression.

tell

Syntax: `tell` *object* `to` *statement*

or

`tell` *object*

statement(s)

`end tell`

This command communicates the statement or statements specified by *statement(s)* to the object specified by *object*.

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 might react to the first movie handler by executing the handler. The movie playing in the window might 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, it is important to use an object name that identifies the window by using the full pathname or its number in `the windowList`. If you use `the 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 opening and closing windows may change the order of `windowList`, it is a good idea to store the full pathname as a global variable.

A multiple-line `tell` command resembles a handler. It requires an `end tell` statement:

```
global childMovie

tell window childMovie

   go to frame 5

   set the stageColor to 100

   set the memberNum of sprite 4 to 45

   updateStage

end tell
```

When a message calls a handler, a value returned from the handler can be found in the global property `the result` after the called handler is done. For example, these statements send the window childMovie the message calcBalance and then return the result:

```
global childMovie

tell window childMovie to calcBalance

-- a handler name

set myBalance to the result

-- return value from "calcBalance" handler
```

When you use the `tell` command to send a message from a movie playing in a window to the main movie, use the system property `the stage` as the object name:

```
tell the stage to go to 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 applies only to handlers in the same script in which you are using the tell command.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

Example:

This statement causes the window Control Panel to instruct the movie Simulation to branch to another frame:

```
tell window "Simulation" to go to frame "Save"
```

tellStreamStatus

Syntax: `tellStreamStatus()`

This function enables or disables stream status reporting.

Setting the parameter for `tellStreamStatus` to different values gives these results:

- `tellStreamStatus(TRUE)` turns on the stream status handler.
- `tellStreamStatus(FALSE)` turns off the steam status handler.
- `tellStreamStatus()` determines the status of the handler.

Example 1:

This `on prepareMovie` handler calls the `on streamStatus` handler when the movie starts:

```
on prepareMovie
  tellStreamStatus(TRUE)
end
```

Example 2:

This statement determines the status of the stream status handler:

```
on mouseDown
  put tellStreamStatus()
end
```

textAlign of field

This is obsolete. Use **alignment of member** instead.

textFont of field

This is obsolete. Use **font of member** instead.

textHeight of field

This is obsolete. Use **lineHeight of member** instead.

text of cast

This is obsolete. Use **text of member** instead.

text of member

Syntax: `the text of member` *whichCastmember*

This cast member property determines the character string that is contained in the field cast member specified by *whichCastmember*.

The `text of member` property is useful for displaying messages and recording what the user types.

The `text of member property` can be tested and set.

Lingo changes to the text of a cast member remove any special formatting you have applied to individual words or lines. Altering `the text of member` reapplies global formatting.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example:

This statement places the phrase "Thank you." in the empty cast member Response:

```
if the text of member "Response" = EMPTY then ¬
   set the text of member "Response" to "Thank You."
```

This statement sets the content of cast member Notice to "You have made the right decision!":

```
set the text of member "Notice" = "You have ¬
   made the right decision!"
```

{button See also,AL(`Lingo_text_of_member')}

textSize of field

This is obsolete. Use **fontSize of member** instead.

textStyle of field

This is obsolete. Use **fontStyle of member** instead.

the

Syntax: `the` *property*

All Lingo properties and many sprite properties and functions require the keyword `the` to precede the property. The keyword `the` distinguishes the property from a variable or object name.

Properties are globally available to handlers without a global declaration. Like global variables, Lingo system properties are available between different movies in the same presentation (unless they are changed by system events). Sprite properties change when a new movie is loaded.

ticks

Syntax: `the ticks`

This function returns the current time in ticks (1/60th of a second). Counting ticks begins from the time the computer is started.

Example:

This statement converts ticks to minutes by dividing the number of ticks by 60 twice, and then sets the variable `minutesOn` to the result:

```
put the ticks/60/60 into minutesOn
```

{button See also,AL(`Lingo_ticks')}

time

Syntax:

```
the time
the short time
the long time
the abbreviated time
the abbrev time
the abbr time
```

This function returns the current time in the system clock as a string in one of three formats: short, long, or abbreviated. If no format is specified, the default 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 cause the Message window to display the time in different formats. 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"
```

Note: The three time formats vary, depending on the individual computer's time format. The examples above are for the United States.

{button See also,AL(`Lingo_time')}

timeoutKeyDown

Syntax: `the timeoutKeyDown`

When this property is TRUE, `keyDown` events set the `timeoutLapsed` property to zero. This is useful for restarting the countdown for a timeout each time a key is pressed.

The `timeoutKeyDown` property can be tested and set. The default value is TRUE.

Example:

This statement sets the variable timing to the value of the `timeoutKeyDown` property:

```
set timing to the timeoutKeyDown
```

This statement turns off the `timeoutKeyDown` property:

```
set the timeoutKeyDown to FALSE
```

{button See also,AL(`Lingo_timeoutKeyDown')}

timeoutLapsed

Syntax: `the timeoutLapsed`

This property indicates the number of ticks 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 but not set directly in Lingo.

Example:

This statement sets the field of member Countdown to the value of the `timeoutLapsed` property. Dividing `timeOutLapsed` by 60 converts it to seconds:

```
set member "Countdown" = the timeoutLapsed / 60
```

timeoutLength

Syntax: `the timeoutLength`

This property determines the number of ticks before a timeout event occurs. A timeout occurs when the `timeoutLapsed` property reaches the time specified by the `timeoutLength` property.

The `timeoutLength` property can be tested and set. The default value is 10,800 ticks, which is 3 minutes.

Example:

This statement sets `the timeOutLength` to 10 seconds:

```
set the timeoutLength to 10 * 60
```

timeoutMouse

Syntax: `the timeoutMouse`

This property determines whether `mouseDown` events reset the `timeoutLapsed` property to zero. When this property is TRUE, `mouseDown` events reset the `timeoutLapsed` property.

The `timeoutMouse` property can be tested and set. The default value is TRUE.

Example 1:

This statement records the current setting of `the timeoutMouse` by setting the variable named `timing` to `the timeoutMouse`:

```
set timing = the timeoutMouse
```

Example 2:

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:

```
set the timeoutMouse to FALSE
```

{button See also,AL(`Lingo_timeoutMouse')}

timeoutPlay

Syntax: `the timeoutPlay`

This property determines whether the `timeoutLapsed` property is reset to zero when a movie is played. When `timeoutPlay` is TRUE, playing a movie resets the `timeoutLapsed` property to zero. This allows timeouts to occur only when the animation is paused.

The `timeoutPlay` property can be tested and set. The default value is FALSE.

Example:

This statement sets `the timeoutPlay` to TRUE, which has Lingo reset the `timeoutLapsed` property to zero when a movie is played:

```
set the timeoutPlay to TRUE
```

timeoutScript

Syntax: `the timeoutScript`

This 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.

Define a primary event handler for timeouts by setting `the timeoutScript` to a string of the appropriate Lingo. The Lingo can be a simple Lingo statement or a calling statement for a handler. When the event script you've assigned is no longer appropriate, turn it off with the statement `set the timeoutScript to EMPTY`.

The `timeoutScript` property can be tested and set. The default value is EMPTY.

Example:

This statement sets `the timeoutScript` to a calling script for the handler `timeoutProcedure`:

```
set the timeoutScript to "timeoutProcedure"
```

timer

Syntax: `the timer`

This property is a free running timer that counts time in ticks (60ths of a second). It has nothing to do with the `timeOutScript` property. It is used only for convenience in timing certain events. The `startTimer` command zeroes the value of the timer property.

The `timer` property is useful for setting up delays within handlers. (The `delay` command works only in frame scripts.) For example, you can use `the timer` to synchronize pictures to a soundtrack by inserting a delay that makes the movie wait until a sound is finished.

Example:

This handler creates a 1-second delay:

```
on countTime
   startTimer
   repeat while the timer < 60
      nothing
   end repeat
end countTime
```

This statement sets the variable `startTicks` to the current value of the timer:

```
set startTicks = the timer
```

{button See also,AL(`Lingo_timer')}

timeScale of member

Syntax: `the timeScale of member` *whichCastmember*

This digital video cast member property gives the time unit per second that the digital video's frames are based on. For example, a QuickTime digital video is measured in 1/600s of a second. This property can be tested but not set.

{button See also,AL(`Lingo_timeScale_of_member')}

title of window

Syntax: `the title of window` *whichWindow*

This window property is the title of the window specified by *whichWindow*.

The `title of window` property can be tested and set.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

Example:

This statement makes Action View the title of window X:

```
set the title of window "X" to "Action View"
```

titleVisible of window

Syntax: `the titleVisible of window` *whichWindow*

This window property specifies whether the window specified by *whichWindow* displays the window title in the window's title bar.

The `titleVisible of window` property can be tested and set.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

Example:

This statement displays the title of the window Control Panel by setting the window's `titleVisible` property to TRUE:

```
set the titleVisible of window "Control Panel" to TRUE
```

to

The word `to` occurs in a number of Lingo constructs.

{button See also,AL(`Lingo_to')}

top of sprite

Syntax: `the top of sprite` *whichSprite*

This sprite property returns the top vertical coordinate of the bounding rectangle of the sprite specified by *whichSprite*. The coordinate is the number of pixels from the upper left corner of the Stage.

The `top of sprite` property can be tested but not set directly. Set `the rect of sprite` to set the top vertical coordinate of a sprite.

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 the top of sprite 3 < 0 then offTopEdge
```

{button See also,AL(`Lingo_top_of_sprite')}

trace

Syntax: `the trace`

This movie property specifies whether the movie's trace function is on or off.

- When `the trace` is TRUE, the trace function is on.
- When `the trace` is FALSE, the trace is off.
- This property can be tested and set.

Example:

This statement turns the `trace` property on:

```
set the trace = TRUE
```

traceLoad

Syntax: `the traceLoad`

This property specifies the amount of information that is displayed about cast members as they are loaded. The possible values for the `traceLoad` property have the following effect:

0-Displays no information

1-Displays cast members' names

2-Displays cast members' names, number of the current frame, movie name, and file seek offset

The `traceLoad` property can tested and set.

Example:

This statement causes the movie to display the names of cast members as they are loaded:

```
set the traceLoad to 1
```

traceLogFile

Syntax: `the traceLogFile`

This property specifies the name of the file that the Message window display is written to. You can close the file by setting the `traceLogFile` property to `EMPTY` ("").

Example:

This statement instructs Lingo to write the contents of the Message window to the file Messages:

```
set the traceLogFile = "Messages"
```

This statement closes the file that the Message window display is being written to:

```
set the traceLogFile = ""
```

trackCount(member)

Syntax: `trackCount(member` *whichCastmember*`)`

This digital video cast member property returns the number of tracks on the specified digital video cast member. This property can be tested but not set.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement determines the number of tracks in the digital video cast member Jazz Chronicles and displays the answer in the Message window:

```
put trackCount(member "Jazz Chronicle")
```

trackCount(sprite)

Syntax: `trackCount(sprite` *whichSprite*`)`

This digital video sprite property returns the number of tracks on the specified digital video sprite. This property can be read but not set.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement determines the number of tracks in the digital video sprite assigned to channel 10 and displays the answer in the Message window:

```
put trackCount(sprite 10)
```

trackEnabled

Syntax: `trackEnabled(sprite` *whichSprite*`,` *whichTrack*`)`

This digital video sprite property indicates whether the specified track of a digital video is enabled to play.

- When `trackEnabled` is TRUE, the specified track is enabled.
- When `trackEnabled` is FALSE, the specified track is disabled.

This property cannot be set. You must use the `setTrackEnabled` property.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

{button See also,AL(`Lingo_trackEnabled')}

tracking

Syntax: `the tracking of sprite` *whichSprite*

This sprite property determines whether a button is tracking over the specified sprite. A button tracks if the mouse was pressed while over the sprite, but isn't yet released yet.

This property can be tested but not set.

Example:

This statement displays the state of tracking for sprite 5:

```
put the tracking of sprite 5
```

trackNextKeyTime

Syntax: `trackNextKeyTime(sprite` *whichSprite, whichTrack*)

This digital video sprite property indicates the time of the key frame that follows the current time in the specified digital video track. This property can be tested but not set.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement has the Message window display the time of the key frame 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 trackNextKeyTime(sprite 15, 5)
```

trackNextSampleTime

Syntax: `trackNextSampleTime(sprite` *whichSprite*`,` *whichTrack*`)`

This digital video sprite property indicates the time of the next sample that follows the digital video's current time. It is useful for locating text tracks in a digital video. This property can be tested but not set.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

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 trackNextSampleTime(sprite 15, 5)
```

trackPreviousKeyTime

Syntax: `trackPreviousKeyTime(sprite` *whichSprite*`,` *whichTrack*`)`

This digital video sprite property reports the time of the prior key frame that precedes the current time. This property can be tested but not set.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement has the Message window display the time of the key frame in track 5 that precedes the current time in the digital video sprite assigned to channel 15:

```
put previousKeyTime(sprite 15, 5)
```

trackPreviousSampleTime

Syntax: `trackPreviousSampleTime(sprite` *whichSprite*, *whichTrack*)

This digital video sprite property indicates the time of the sample preceding the digital video's current time. It is useful for locating text tracks in a digital video. This property can be tested but not set.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

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 trackPreviousSampleTime(sprite 15, 5)
```

trackStartTime(member)

Syntax: `trackStartTime(member` *whichCastMember*`,` *whichTrack*`)`

This digital video cast member property gives the start time of the specified track of the specified digital video cast member. It can be tested but not set.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

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 trackStartTime(member "Jazz Chronicle", 5)
```

trackStartTime(sprite)

Syntax: `trackStartTime(sprite` *whichSprite*, *whichTrack*)

This 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.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

In the Message window, this statement tells when track 5 in sprite channel 10 starts playing. The starting time is 120 ticks (2 seconds) into the track:

```
put trackStartTime(sprite 10, 5)

-- 120
```

{button See also,AL(`Lingo_trackStartTimesprite')}

trackStopTime(member)

Syntax: `trackStopTime(member` *whichCastmember*`,` *whichTrack*`)`

This digital video cast member property provides the stop time of the specified track of the specified digital video cast member. It can be tested but not set.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

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 trackStopTime(member "Jazz Chronicle", 5)
```

trackStopTime(sprite)

Syntax: `trackStopTime(sprite, whichSprite, whichTrack)`

This digital video sprite property provides the stop time of the specified track of the specified digital video sprite. It can be tested but not set.

When a digital video movie is played, `the trackStopTime` is where playback halts or loops if the `loop` property is turned on.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

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 trackStopTime(sprite 6, 5)
```

{button See also,AL(`Lingo_trackStopTimesprite')}

trackText

Syntax: `trackText(sprite` *whichSprite*`,` *whichTrack*`)`

This digital video sprite property provides the text that is at the current time in the specified track of the digital video. 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.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

This statement assigns the text at the current time in track 5 of the digital video assigned to sprite 20 to the field cast member Archives:

```
put trackText(sprite 20, 5) into member "Archives"
```

trackType(member)

Syntax: `trackType (member` *whichCastmember*`,` *whichTrack*`)`

This 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.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

Example:

The following handler checks whether track 5 of digital video cast member Today's News is a text track and runs the handler `textFormat` if it is:

```
on checkForText
 if trackType(member "Today's News", 5) = #text ¬
 then textFormat
end
```

trackType(sprite)

Syntax: `trackType(sprite` *whichSprite*`,` *whichTrack*`)`

This digital video sprite property provides 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.

For more information, see Chapter 8, "Controlling Sound and Digital Video," in *Learning Lingo*.

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 the trackType(sprite 10, 5) = #text ¬
 then textFormat
end
```

trails of sprite

Syntax: `the trails of sprite` *whichSprite*

This property turns the trails ink effect on and off for the sprite specified by *whichSprite*. For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet.

Set `the trails` to 0 to turn trails off; set `the trails` to 1 to turn trails on.

- To erase trails, animate another sprite across these pixels or use a transition.

Example:

This statement sets the trails on for sprite 7:

```
set the trails of sprite 7 to 1
```

{button See also,AL(`Lingo_trails_of_sprite')}

transitionType of member

Syntax: `the transitionType of member` *whichCastmember*

This transition cast member property determines a transition's type, which is given as a specific number. The possible values are the same as the code numbers assigned to transitions for the `puppetTransition` command.

For more information about authoring from Lingo, see Chapter 13, "Authoring from Lingo," in *Learning Lingo*.

Example:

This statement sets the type of transition cast member 3 to 51, which is a pixel dissolve cast member:

```
set the transitionType of member 3 to 51
```

TRUE

Syntax: `TRUE`

This logical constant represents the value of a logically true expression, such as `2 < 3`. It has a numerical value of `1`.

Example:

This statement turns on the `soundEnabled` property by setting it to TRUE:

```
set the soundEnabled to TRUE
```

{button See also,AL(`Lingo_TRUE')}

tweened of sprite

Syntax: `the tweened of sprite`

This property determines whether the individual frames in a sprite are created as key frames when the sprite is created.

- If `the tweened of sprite` is TRUE, only the first frame in the new sprite is a key frame.
- If `the tweened of sprite` is FALSE, all frames in the new sprite are key frames.

The `tweened of sprite` property can be tested and set.

Note: This property does not impact playback and is only useful during Score recording.

Example:

When this statement is issued, newly created sprites only have a key frame in the first frame of the sprite span:

```
set the tweened of sprite to 1
```

type of member

Syntax: `the type of member` *whichCastmember*

`the type of member` *whichCastmember* ¬
`of castLib` *whichCast*

This cast member property indicates the specified cast member's type. This property replaces `castType`, which appeared in earlier versions of Director.

The `type of member` property can be one of the following values:

```
#bitmap            #palette
#button            #picture
#digitalVideo      #richText
#empty             #script
#field             #shape
#filmLoop          #sound
#movie             #transition
#ole
```

You can also define custom cast member types for custom cast members.

The `type of member` cast member property can be tested but not set.

For movies created in Director 5 and Director 6, `the type of member` 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 type of member. This is to provide backward compatibility for movies that were created in Director 4.

For more information about authoring from Lingo, see Chapter 13, "Authoring from Lingo," in *Learning Lingo*.

Example:

The following handler checks whether the cast member Today's News is a field cast member and runs the handler `fieldFormat` if it is:

```
on checkFormat
 if the type of member "Today's News" = #field ¬
 then fieldFormat
end
```

type of sprite

Syntax: `the type of sprite`

In earlier versions of Director, this sprite property determined a sprite's type, but is no longer supported for this purpose.

Currently, the `type of sprite` property lets you clear sprite channels during Score recording by setting the `type of sprite` value for that channel to 0.

The `type of sprite` property can be tested and set.

Example:

This statement clears sprite channel 1 when issued during a Score recording session:

```
set the type of sprite 1 to 0
```

union rect

Syntax: `union rect` *rect1, rect2*

This function returns the smallest rectangle that encloses the two rectangles *rect1* and *rect2*.

Example:

```
put union (rect (0, 0, 10, 10), ¬
rect (15, 15, 20, 20))
-- rect (0, 0, 20, 20)
```

{button See also,AL(`Lingo_union_rect')}

unLoad

Syntax: `unLoad`

`unLoad` *theFrameNum*

`unLoad` *fromFrameNum, toFrameNum*

This command clears the cast members used in a specified frame from memory. When used without an argument, the `unLoad` command clears the cast members in all the frames of a movie from memory-except any being used in the current frame.

When used with one argument, *theFrameNum*, the `unLoad` command clears from memory the cast members in that frame. Director automatically unloads the least recently used cast members to accommodate `preLoad` commands or normal cast loading.

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
```

This statement clears the cast members used from the frame labeled first to the frame labeled last:

```
unLoad "first","last"
```

{button See also,AL(`Lingo_unLoad')}

unLoadMember

Syntax: `unLoadMember member` *whichCastmember*

`unLoadMember member` *whichCastmember* `of castLib` *whichCast*

`unLoadMember member` *fromCastName*`,` *toCastName*

This command clears the specified cast members from memory. When used without an argument, `unLoadMember` causes all cast members in a movie to be cleared from memory-except for any being used in the current frame.

When used with one argument, *whichCastmember*, the `unLoadMember` command clears from memory the cast member name or number that you specify.

When used with two arguments, *fromCastName* and *toCastName*, the `unLoadMember` command unloads all cast members in the range specified.

Example 1:

This statement clears the cast member Screen1:

```
unLoadMember member "Screen1"
```

Example 2:

This statement clears from memory all cast members from cast member 1 to cast member Big Movie.

```
unLoadMember 1, member "Big Movie"
```

{button See also,AL(`Lingo_unLoadMember')}

unloadMovie

Syntax: `unloadMovie` *whichMovie*

This command removes the specified preloaded movie from memory. This can be useful for prioritizing which 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 1:

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 2:

This statement unloads the movie at http://www.cbDemille.com/SunsetBlvd.dir:

```
unLoadMovie "http://www.cbDemille.com/SunsetBlvd.dir"
```

updateFrame

Syntax: `updateFrame`

This command enters the changes that have been made to the current frame and steps 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.

This command works during a Score generation session only.

When used in the movie's last frame or any frame after that, the `updateFrame` command duplicates everything in the frame and copies it into the next frame. (You can determine the last frame number using the statement `put the lastFrame`.)
To avoid this, assign an empty frame script or Tempo setting in a frame that is well past the last frame in which you plan to record Score.

For more information about authoring from Lingo, see Chapter 13, "Authoring from Lingo," in *Learning Lingo*.

Example:

When used in the following handler, the `updateFrame` command enters the changes that have been made to the current frame and steps 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
      set horizontal = 0
      set vertical = 300
      repeat with i = 1 to numberOfFrames
         go to frame i
         set the memberNum of sprite 20 to ¬
         the number of member "Ball"
         set the locH of sprite 20 to horizontal
         set the locV of sprite 20 to vertical
         set the type of sprite 20 to 1
         set the foreColor of sprite 20 to 255
         set horizontal = horizontal + 3
         set vertical = vertical + 2
         updateFrame
      end repeat
   endRecording
end
```

{button See also,AL(`Lingo_updateFrame')}

updateLock

Syntax: `the updateLock`

This movie property determines whether the Stage is updated during Score recording

- When `the updateLock` is TRUE, the Stage is not updated.
- When `the updateLock` is FALSE, the Stage is updated.

You can keep the Stage display constant during a Score recording session by setting the `updateLock` movie property to TRUE before Lingo updates the Score. If `the updateLock` is FALSE, the Stage updates to show a new frame each time the frame is entered by the `updateLock` command.

You can also use `the updateLock` to prevent unintentional Score updating when leaving a frame, such as when temporarily leaving a frame to examine properties in another frame.

For more information about authoring from Lingo, see Chapter 13, "Authoring from Lingo," in *Learning Lingo*.

updateMovieEnabled

Syntax: `the updateMovieEnabled`

This property specifies whether changes made to the current movie are automatically saved when the movie branches to another movie.

- When `the updateMovieEnabled` is TRUE, changes to the movie are automatically saved when the movie branches to another movie.
- When `the updateMovieEnabled` is FALSE, changes to the movie are not automatically saved when the movie branches to another movie. The default value is FALSE.

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.

```
set the updateMovieEnabled = TRUE
```

updateStage

Syntax: `updateStage`

This command redraws the Stage immediately. Normally the Stage is updated only between frames, but the `updateStage` command updates the Stage any time the command is executed from a handler.

The `updateStage` command redraws sprites, performs transitions, plays sounds, and sends a `stepFrame` message.

Example:

This handler changes the sprite's horizontal and vertical locations, and redraws the Stage so that the sprite appears in the new location:

```
on moveRight whichSprite, howFar
   puppetSprite whichSprite, TRUE
   set the locH of sprite whichSprite ¬
      to the locH of sprite whichSprite + howFar
   updateStage
end moveRight
```

URL of member

Syntax: `set the URL of member` *" whichCastmember"*

This property specifies the URL for the Shockwave Audio cast member.

This property can be tested. It can be set only when the SWA streaming cast member is stopped.

For more information about Shockwave movies and the internet, see Chapter 14, "Shockwave, the Internet, and Lingo," in *Learning Lingo*.

Example:

This statement makes a file on an internet server the URL for SWA cast member Benny Goodman:

```
on mouseDown
   set the URL of member "Benny Goodman" = ¬
   "http://audio.macromedia.com/samples/classic.swa"
end
```

value

Syntax: `value(`*string)*

This function returns the numerical value of a string. This is useful when making use of a numerical string that the user has typed into a field cast member or data from Xtras and XObjects that return numerical strings.

Example 1:

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 2:

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.

{button See also,AL(`Lingo_value')}

version

Syntax: `version`

This system variable contains the version string for Macromedia Director. The same string appears the Finder's Get Info dialog box.

Example:

This statement displays the version of Macromedia Director in the Message window:

```
put version
-- "6.0"
```

video of cast

This is obsolete. Use **video of member** instead.

video of member

Syntax: `the video of member` *whichCastmember*

This digital video cast member property enables or disables playing the graphic image of the specified digital video cast member.

- When `the video of member` is TRUE (1), the digital video is enabled.
- When `the video of member` is FALSE (0), the digital video is disabled.

Only the visual element of the digital video cast member is affected. For example, when `the video of member` 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:

```
set the video of member "Interview" to FALSE
```

videoForWindowsPresent

Syntax: `the videoForWindowsPresent`

This movie property indicates whether Video for Windows is present on the computer. It can be tested but not set.

Example:

This statement checks whether Video for Windows is missing and has the playback head go the marker Alternate Scene if it isn't:

```
if the videoForWindowsPresent = FALSE then go to "Alternate Scene"
```

visible of sprite

Syntax: `the visible of sprite` *whichSprite*

This sprite property determines whether the sprite specified by *whichSprite* is visible.

- When the `visible of sprite` property is TRUE, the sprite is visible.
- When the `visible of sprite` property is FALSE, the sprite is not visible.

The `visible of sprite` property can be tested and set.

Example:

This statement makes sprite 8 visible:

```
set the visible of sprite 8 to TRUE
```

visible of window

Syntax: `the visible of window` *whichWindow*

This window property determines whether the window specified by *whichWindow* is visible.

- When the `visible of window` property is TRUE, the window is visible.
- When the `visible of window` property is FALSE, the window is not visible.

The `visible of window` property can be tested and set.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

Example:

This statement makes the window Control Panel visible:

```
set the visible of window "Control Panel" to TRUE
```

VOID

Syntax: `VOID`

This constant is 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
```

voidP

Syntax: `voidP(`*variableName*`)`

This property specifies whether `VOID` is the value of the specified variable.

- When the `voidP` property is TRUE, the variable has not been given an initial value.
- When the `voidP` property is FALSE, the variable has been given an initial value.

The `voidP` property can be tested but not set.

Example:

This statement checks whether the variable `answer` has been given an initial value:

```
put voidP(answer)
```

volume of member

Syntax: `the volume of member "`*whichCastmember*`"`

This Shockwave Audio streaming cast member property determines the volume of the specified SWA streaming cast member.

Allowable values range from 0 to 255.

This property can be tested and set.

Example:

This statement sets the volume of a Shockwave Audio streaming cast member to half the possible volume:

```
set the volume of member "SWAfile" to 128
```

volume of sound

Syntax: `the volume of sound` *whichChannel*

This sound property determines the volume of the sound channel specified by *whichChannel.* Sound channels are numbered 1, 2, 3, and so on. 1 and 2 are the channels that appear in the Score.

The value of the `volume of sound` property ranges from 0 (mute) to 255 (maximum volume).

The lower the value of `the volume of sound`, the more static or noise you're likely to hear. Using `the soundLevel` may produce less noise, although it offers less control.

Example:

This statement sets the volume of sound channel 2 to 130, which is a medium setting:

```
set the volume of sound 2 to 130
```

{button See also,AL(`Lingo_volume_of_sound')}

volume of sprite

Syntax: `the volume of sprite` *whichSprite*

This property can be used to control the volume of a digital video movie cast member. You can use a cast name or number. The values for volume range from -256 to 256. Values of zero or less are mute.

Example:

This statement sets the volume of the QuickTime movie playing in sprite channel 7 to 256, which is the maximum sound volume:

```
set the volume of sprite 7 to 256
```

{button See also,AL(`Lingo_volume_of_sprite')}

when...then

This Lingo construct is obsolete.

width of cast

This is obsolete. Use **width of member** instead.

width of member

Syntax: `the width of member` *whichCastMember*

This cast member property determines the width, in pixels, of the cast member specified by *whichCastMember*. The `width of member` applies only to bitmap and shape cast members. It does not affect field or button cast members.

The `width of member` property can be tested but not set.

Example:

This statement assigns the width of member 50 to the variable `height`:

```
put the width of member 50 into height
```

{button See also,AL(`Lingo_width_of_member')}

width of sprite

Syntax: `the width of sprite` *whichSprite*

This sprite property determines the horizontal size in pixels of the sprite specified by *whichSprite*. The width applies only to bitmap and shape cast members. It does not affect field or button cast members.

The `width of sprite` property can be tested and set.

Setting this property has no effect on bitmap sprites unless the sprite's `stretch of sprite` property is set to TRUE.

Example 1:

This statement sets the width of sprite 10 to 26 pixels:

```
set the width of sprite 10 to 26
```

Example 2:

This statement assigns the width of sprite number i + 1 to the variable `howWide`:

```
put the width of sprite (i + 1) into howWide
```

{button See also,AL(`Lingo_width_of_sprite')}

window

Syntax: `window` *whichWindow*

This 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. By using windows that play movies, you can have several movies open at once and allow them to interact.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

Example 1:

This statement opens the window Control Panel:

```
open window "Control Panel"
```

Example 2:

This statement moves the window Control Panel to the front:

```
moveToFront window "Control Panel"
```

{button See also,AL(`Lingo_window')}

windowList

Syntax: `the windowList`

This property is a list of all the known movie windows.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

Example 1:

This statement displays all the known movie windows in the Message window:

```
put the windowList
```

Example 2:

This statement clears `the windowList`:

```
set the windowList = []
```

windowPresent

Syntax: `windowPresent("windowName")`

This function indicates whether the object specified by *windowName* is running as a movie in a window. The *windowName* argument must be the window's name as it appears in the `windowList` property.

- When `windowPresent` returns TRUE (1), the object is a movie in a window.
- When `windowPresent` returns FALSE (0), the object isn't a movie in a window.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

Example:

This statement tests whether the object myWindow is a movie in a window and displays the result in the Message window:

```
put windowPresent(myWindow)
```

windowType of window

Syntax: `the windowType of window` *whichWindow*

This window property specifies the display style of the window specified by *whichWindow*. Possible values for *whichWindow* are:

0	Moveable, sizeable 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	Moveable window without size box or zoom box
5	Moveable modal dialog box
8	Standard document window
12	Zoomable, nonresizeable window
16	Rounded corner window
49	Floating palette, during authoring (in Macintosh projectors, the value 49 specifies a stationary window)

In Windows, these numbers create windows with the same functionality but a Windows appearance. Other values for `the windowType of window` are possible, but use them with caution, because some modal windows can only be exited by restarting the computer.

For more information about movies in a window, see Chapter 11, "Movies in a Window," in *Learning Lingo*.

Example:

This statement sets the value of the display style of the window named Control Panel to 8:

```
set the windowType of window "Control Panel" to 8
```

word...of

Syntax: `word` *whichWord* `of` *chunkExpression*

`word` *firstWord* `to` *lastWord* `of` *chunkExpression*

This chunk expression keyword specifies a word or a range of words in a chunk expression. A word chunk is any sequence of characters delimited by spaces. (Any non-visible character-such as a Tab or Enter-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 fields (field cast members) and variables that hold strings.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example 1:

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:

```
set animalList = "fox dog cat"
put "elk" before word 2 of animalList
```

The result is the string "fox elk dog cat".

Example 2:

This statement causes the Message window to display the fifth word of the same string:

```
put word 5 of "fox elk dog cat"
```

Because there is no fifth word in this string, the Message window displays two quotation marks (""), which indicate an empty string.

{button See also,AL(`Lingo_word_of')}

wordWrap of member

Syntax: `the wordWrap of member` *whichCastmember*

This field cast member property controls line wrapping.

- When TRUE, `the wordWrap of member` allows line wrapping.
- When FALSE, `the wordWrap of member` prevents line wrapping.

For more information, see Chapter 7, "Working with Fields and User Input," in *Learning Lingo*.

Example:

- This statement turns the line wrapping off for the field cast member Rokujo:

```
set the wordWrap of member "Rokujo" to FALSE
```

xFactoryList

Syntax: `xFactoryList(`*whichLibrary*`)`

This function returns a string list of all the currently available Xtras and XObjects in the XLibrary file specified by the string *whichLibrary*. The XLibrary must have been previously opened with the `openXlib` command. If you specify `EMPTY` for *whichLibrary*, this function returns a list of all Xtras and XObjects in all open XLibraries.

The Xtras and XObjects appear one per line in the returned string list. Each line ends with a an `Enter` character.

Example 1:

This statement displays the Xtras and XObjects available in the Xlibrary named AppleCD XObj:

```
put xfactoryList("AppleCD XObj")"
```

Example 2:

This statement displays the first line of the list of all available Xtras and XObjects in all open Xlibraries:

```
put line 1 of xfactoryList(EMPTY)
```

{button See also,AL(`Lingo_xFactoryList')}

xtra

Syntax: `xtra "`*whichLingoXtra*`"`

This function returns an instance of the Lingo Xtra specified by *whichLingoXtra*.

Example:

This statement uses the `new` function to create a new instance of the Lingo Xtra HelpXtra and assigns it to the variable `tool`:

```
set tool = new(xtra "HelpXtra")
```

{button See also,AL(`Lingo_xtra')}

xtras

{button See also,AL(`Lingo_xtras')}

zoomBox

Syntax: `zoomBox` *startSprite*`,` *endSprite* `[,` *delayTicks*`]`

This command creates a zooming effect, like the expanding windows in the Finder (Macintosh). 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-Looks for *endSprite* in the current frame, otherwise,

2-Looks for *endSprite* in the next frame.

Note, however, that the `zoomBox` command does not work for an *endSprite* 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
```

Menus

For information on a command, click the name of the menu in which it appears:

File
Edit
View
Insert
Modify
Control
Xtras
Window
Help

Or browse the **alphabetical list of menu commands**.

Menu commands

Click a command or submenu listed alphabetically below:

Field
Film Loop
Filter Bitmap
Find Again
Find Cast Member
Find Handler
Find Selection
Find Text
Font
Frame Palette
Frame Script
Frame Sound
Frame Tempo
Frame Transition
Frame
General Preferences
Grids > Settings
Grids > Show
Grids > Snap To
Ignore Breakpoints
Import
Insert Frame command
Inspector > Behavior
Inspector > Memory
Inspector > Sprite
Inspector > Text
Invert Selection
Join Sprites
Keyframe (insert menu)
Keyframes (view menu)
Launch External Editor
Loop Playback
Marker > Next
Marker > Previous
Markers Window
Media Element > Bitmap
Media Element > Color Palette
Media Element > Shockwave Audio
Media Element > Text
Message
Move Backward
Run Script
Save
Save All
Save and Compact
Save As
Save As Shockwave Movie
Score Preferences
Score
Script
Select All
Selected Frames Only
Send to Back
Show Grid
Snap To Grid
Sort
Space to Time
Split Sprite
Sprite > Properties
Sprite > Script
Sprite > Tween Properties
Sprite Overlay > Show Info
Sprite Overlay > Show Paths
Sprite Overlay > Settings
Sprite Toolbar
Stage
Step Backward
Step Forward
Step Into Script
Step Script
Stop
Transform Bitmap
Text
Toggle Breakpoint
Toolbar
Tool Palette
Transform Bitmap
Tweak
Undo
Update Movies
Video
Volume

File menu

The File menu contains commands for creating Director movies and casts, opening and saving movies, importing and exporting files, creating projectors, and printing.

Click the name of a menu command for more information:

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

New > Movie command (File menu) Control-N

The New Movie command opens a new, untitled movie. Name the untitled movie in the dialog box that appears the first time you save the movie.

New > Cast command (File menu) Control-Alt-N

The New Cast command creates new internal and external casts.

To create a new cast, enter a name for the new cast, choose either Internal or External, and then click Create.

If you choose External, Used in Current Movie check box is turned on. Click the check box if you don't want the external cast used in the current movie.

{button See also,AL(`New_Cast_help')}

Open command (File menu) Control-O

The Open command opens an existing Director movie or external cast. When you choose Open, a directory dialog box appears.

The dialog box only lists movies and casts created by the Macintosh version of Director, or movies with a .DIR extension created with the Windows version of Director.

When you open a movie with linked external casts, Director opens the external casts as well. If it cannot find them in the specified location, it prompts you to choose a new location.

{button See also,AL(`Open_help')}

Close command (File menu) Control-F4

This command closes the current window.

Save command (File menu) Control-S

The Save command saves the current movie, replaces the previous version, and saves all casts (internal and external) linked to the movie.

If an external cast is the active window, then the external cast is saved, not the movie.

The first time you save a movie with linked external casts, Director prompts you to enter a name and location for each cast.

Tip: To change the movie's color depth, change the monitor's color depth before saving the movie.

{button See also,AL(`Save_help')}

Save As command (File menu) Control-Shift-S

Use Save As to name and save a movie, save the movie under a different name, or save it to a different folder. Enter the new name of the movie to name it, or choose a new folder.

The first time you save a movie with linked external casts, Director prompts you to enter a name and location for each cast.

Tip: To change the movie's color depth, change the monitor's color depth before saving the movie.

{button See also,AL(`Save_As_help')}

Save As Shockwave Movie command (File menu) Control-Shift-S

Use this command to convert the currently opened Director movie into a Shockwave movie that can be played on the Internet.

The Save As Shockwave Movie command saves a movie file in the compressed Shockwave format.

{button See also,AL(`Save_Shockwave_help')}

Save and Compact (File menu) Control-Option-S

Use Save and Compact to save the movie so that it is optimized for playback. Since this operation reorders the cast and compacts the file, it takes longer, especially if you are saving a very large file. However, this command produces smaller and more efficient movies.

{button See also,AL(`Save_Compact_help')}

Save All command (File menu)

Use Save All to save the movie and all external cast files, linked and unlinked. If the movie or any casts have not been saved before, a dialog box appears.

{button See also,AL(`Save_All_help')}

Revert command (File menu)

The Revert command opens the last saved version of the current movie. This command is dimmed if you have not made any changes or if you are working on a new untitled movie that you have not yet saved.

Import command (File menu) Control-R

When you choose Import, Director opens a cast window and displays a dialog box so you can choose the file you want to import.

Note: Director 6.0 for Windows supports the following file types in addition to those also supported by Director for Macintosh: JPEG, CompuServe GIF, TIFF, EPS, Photo CD, Windows metafiles, and FCC and FCI.

Dialog box options

Preview shows a thumbnail image of the file to be imported.

Show determines what type of files appear in the list of available files. Choose a media type that Director imports. For example, choose Bitmap Image to view only the bitmaps in the current folder.

Add All places all the files of the selected type in the current folder in the list of files to be imported.

Show Preview displays a thumbnail image of the file to be imported.

Internet opens a dialog box in which you can enter a URL. Director imports the file from the URL or creates a link, depending on the option you choose from the Media pop-up at the bottom of the dialog box.

Options provides further options when importing PICS or Scrapbook files.

Add, **Add All**, and **Remove** let you build a list of multiple files to import.

Media:

- **Standard Import** imports data directly into the movie.
- **Link to External File** gives you the option to create a link to a PICT file, AIFF sound file, or a Director movie rather than copying the contents of the file into your movie.
- **Include Original Data for Editing** is for use with external editors. When this option is selected, Director retains a copy of the original data. When you edit the cast member with an external editor, Director sends the original data to the external editor.
- **Import PICT File As PICT** is only available when you import a Macintosh PICT file. If checked, the image is imported and pasted into the cast as a PICT cast member. If not checked, Director converts the imported image to a bitmapped cast member.

Import imports all the selected files and places them in the active cast.

Note: Digital videos are always linked.

{button See also,AL(`Import_help')}

Image Options

If you import a bitmap with a color depth or color palette different than the current movie, the Image Options dialog box appears. You can choose to import the bitmap at its original color depth or at the Stage color depth. You also have the choice of importing the image's color palette, or remapping the image's colors to a palette already in the movie.

In many cases, it's easiest to change the image's color depth to the depth of the Stage and remap the image to the color palette used in the rest of the movie. If you are not concerned with the exact colors that display on Stage, remap everything to the system palette.

Dialog box options

Color Depth selects the color depth for the cast member.

- **Image** specifies the color depth of the current image. Select this option to import the image at this color depth.
- **Stage** specifies the color depth of the current Stage. Select this option to import the file at this color depth.

Palette specifies options for importing 2-, 4-, or 8-bit images.

- **Import** specifies importing the image with its color palette.
- **Remap To** makes Director replace the image's colors with the most similar solid colors in the palette you select from the pop-up.
- **Dither** blends the colors in the new palette to approximate the original colors in the graphic.

Same Settings for Remaining Images applies the current settings to all the remaining files you selected for importing.

Note: If you change 16-, 24-, or 32-bit cast members to 8-bits or less, you need to remap them to an existing color palette.

{button See also,AL(`Image_options_help')}

Export command (File menu) Control-Shift-R

Exports frames of Director movies from Director so you can save images as stills, or create digital videos. You can choose to export frames of movies as AVI, or DIB files.

Dialog box options

Export options determine which frames to export:

- **Current Frame** exports the current frame on the Stage. This is the default.
- **Selected Frames** exports the selected frames in the Score window.
- **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.

Include:

- **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.

Format specifies the exported file format from the pop-up. File formats you can export are Video for Windows (AVI) and DIB (Device Independent Bitmap).

Options is dimmed unless you choose Video for Windows from the Format pop-up. Specify the export frames per second (fps) interval in the Frame Rate box in the Video for Windows Export Options dialog box.

When you click Export, a dialog box appears, allowing you to name the file. If you are saving in AVI format, only one file will be created. If you are saving in DIB 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 "Myfi0001.dib" for Windows 3.1 and NT or "Myfile0001" for Windows '95.

{button See also,AL(`Export_help')}

Create Projector command (File menu)

The Create Projector command creates a play-only version of a Director movie, called a projector. You can package several movies, external casts, Xtras, and linked media in a single projector. Use the Create Projector dialog box to add these objects from the Source Folder to the projector's play list.

Movies created with previous versions of Director are not listed in the Create Projector dialog box. To include movies that were created with a previous version of Director, you must first open the movies in the current version of Director and save them, or use the **Update Movies** command to convert them.

If a movie is open when you choose this command, Director closes the current movie.

Dialog box options

Source Folder (on the left side of the dialog box)	From the opened folder, select the movies and external casts you want to include in the projector.
Add	Moves the selected source movie to the projector's play list (on the right side of the dialog box).
Add All	Adds all movies in the current folder to the play list.
Remove	Deletes the selected source movie from the projector's play list.
Move Up and Move Down	Moves a selected movie higher up or further down in the play list.
Create	Assembles the projector file.
Done	Dismisses the dialog box.
Create	Assembles the projector file.
Options	Opens the **Projector Options dialog box** that specifies additional preferences for creating the projector.

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Projector Options dialog box

The Projector Options dialog box specifies additional preferences for creating the projector.

Note: These settings override any movie preferences you set in **Movie Properties** and apply to all movies in the projector.

{button See also,AL(`Projector_Options_help')}

Dialog box options

Create for specifies the systems for the projector to run on:

- **Windows 3.1** creates a projector that runs on Windows 3.1. A Windows 3.1 projector will also run on Windows 95 and NT.
- **Windows 95 and NT** creates a projector that runs on all Windows 32-bit operating systems.

Playback:

- **Play Every Movie** specifies that the projector plays all movies in the play list. Otherwise, the projector only plays 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 checked, pressing Command-period will go to the next movie and Command-Q will quit.
- **Animate in Background** allows the movie to continue playing if a user clicks outside the Stage. This is useful if you are using Apple Events. If not checked, the movie stops playing if the user clicks outside the Stage.

Options:

- **Full Screen** displays the movie full 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 is not resizeable.
- **Show Title Bar** is available only if In a Window is selected. If checked, the window where the movie appears has a title bar. The window is only moveable if it has a title bar.

Stage Size:

- **Use Movie Settings** uses the same Stage size of the new movie or matches the size of the current movie.
- **Match First Movie** repositions and resizes 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.

Media:

- **Compress (Shockwave Format)** compresses the projector's movie data in the Shockwave format. This makes the projector smaller, but increases the load time.

Xtras:

- **Include Network Xtras** includes the Xtras required to connect the projector to the internet. Check this option if your movie uses any linked media from the internet.
- **Check Movies for Xtras** includes in the projector all Xtras listed in the Movie Xtras dialog. Xtras used only in Lingo do not appear in Movie Xtras.

Page Setup command (File menu) Control-Shift-P

The Page Setup command offers options for determining how a page is to be printed. The dialog box that you see depends on the type of printer you use.

Print command (File menu) Control-P

The Print command lets you print your movie in a variety of ways.

Click a dialog box option for more information:

{button See also,AL(`Print_help')}

Print specifies what part of the movie to print. 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, or Cast Thumbnails print options let you choose from a range of cast and cast members-internal or external. Information displayed in the Print dialog box depends on the selection to be printed.

When you use the Print command, cast members in each frame are merged into a single image. If you press the Alt key while clicking the Print button, each cast member in a given frame is imaged as a separate PICT. If you are printing multiple frames on a page, printing with this option causes shape elements (text, rectangles, lines, and circles) to print better. However, printing with this option may take much longer.

Frames controls which frames of your movie are printed.

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

Include lets you specify which frames to print.

- **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 10th 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.

Options displays a dialog box that lets you adjust the layout of the items you choose to print. The image at the left of the dialog box previews the layout options.

- **Scale** provides options to print 100%, 50% or 25% of original size.
- **Frame Borders** creates a border around each frame.
- **Frame Numbers** prints the frame number with each frame.
- **Registration Marks** places registration marks on every page.
- **Storyboard Format** is only available when you select 50%- or 25%-size images to print. It 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.

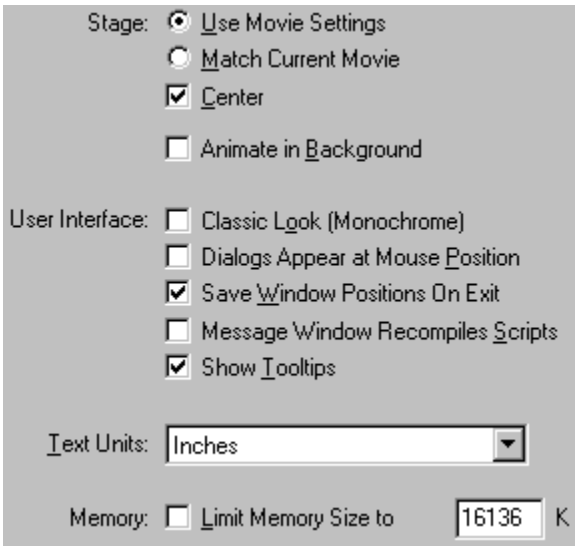
Send Mail (File menu) Control-P

Send Mail mails an open movie, together with any other information you provide, to the user you specify in a dialog. (The dialog is supplied by the mail-system you have installed, and may differ between systems.) The command is only enabled when electronic mail software is installed on your computer in Windows 95 and NT. It is not supported on Windows 3.1.

Preferences > General command (File menu) Control-U

The General Preferences command allows you to modify some of Director's default settings.

Click a dialog box option for more information:



Stage Size specifies the size and location of the Stage.

- **Use Movie Settings** sets the Stage size to the movie's Stage size and location.
- **Match Current Movie** opens the new movie in the Stage of the movie that's currently open.
- **Center** positions the Stage in the center of the screen, which is useful if the Stage size is smaller than the screen size. Otherwise, the movie plays using its original Stage position.
-
- **Animate in Background** runs your 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.

Use **Movie Properties** on the Modify menu to specify the exact size of the Stage.

User Interface

- **Classic Look (Monochrome)** switches to a black-and-white user interface. Using a black-and-white user interface improves performance if you switch color palettes, since Director doesn't have to update its color user interface to match the colors in the new palette each time you switch palettes. In addition, working with a black-and-white user interface may be less distracting as you work with the color palettes in your movie. For example, if you are working on an animation that uses multiple palettes and/or color cycling, using a black-and-white user interface may be less distracting.
- **Dialogs Appear At Mouse Position** displays dialog boxes at the mouse position. If this option is not checked, 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.
- **Message Window Recompiles Scripts** is turned on by default. If deselected, scripts should be manually recompiled using the Recompile All Scripts command before entering Lingo in the Message window.
- **Show Tooltips** controls the definitions that appear when the pointer is over tools. Turn the option off to stop definitions from appearing.

Text Units specifies inches, centimeters, or pixels for the units of measure displayed on the Ruler in the Text and Field windows.

Memory limits the amount of memory Director uses. Click Limit Memory Size to, and then enter the number of Kilobytes in the box on the right.

Preferences > Network command (File menu)

This command opens the Network Preferences dialog box. This box includes the settings and configurations that control communication between Director and the internet.

Dialog box options

Preferred Browser specifies the browser to launch when a movie running the authoring environment encounters the "`gotoNetPage`" lingo command. Use the Browse option to locate the browser.

Launch When Needed enables or disables browser launching.

Note: When Director encounters instances that require browser launching, and a preferred browser has not been specified, an alert appears with the following options: Browse (to select the preferred browser) or Disable (to disable browser launching).

Cache Size defines the amount of space on your hard disk to be used by Director to cache data from the internet.

Clear empties the cache immediately.

Check Documents specifies how often cached documents are compared to the same documents on the server. This comparison determines whether you view an updated page from the network server or a potentially out-of-date page stored in the cache.

- **Once Per Session** checks for page revisions only once during the time your start and quit the application.
- **Every Time** repeatedly checks for changes when you request a page. This option slows performance.

Proxies specify the configuration of your system's proxy server.

Ordinarily, browsers don't require proxies to interact with the network services of external sources. However, in some network configurations the connection between the browser software and a remote server is blocked by a firewall. A firewall protects information in internal computer networks from external access, and in doing so, may limit the ability to exchange information.

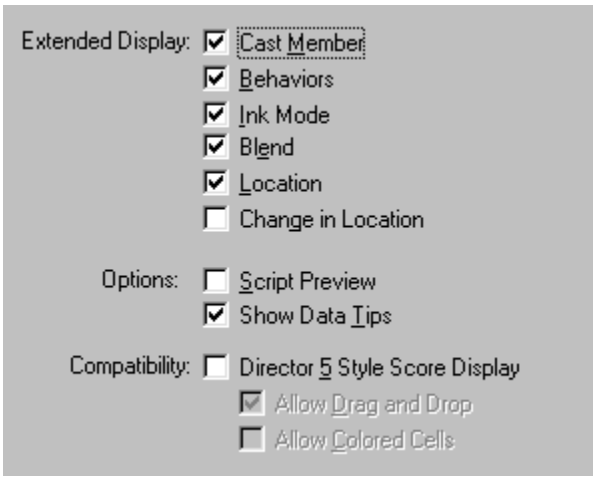
To overcome this limitation, browser software can interact with proxy software. A proxy server interacts with the firewall and acts as a conduit, providing a specific connection for each network service protocol. If you are running browser software on an internal network from behind a firewall, you will 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.

Preferences > Score Window command (File menu)

This command controls display options in the Score window.

Click a dialog box option for more information:



{button See also,AL(`Score_Window_Options_help')}

Extended Display options let you choose the notation information that appears in the numbered sprite channels when extended display is on. You can turn on extended display with the Score window's Display pop-up.

- **Cast Member** displays the cast member number and/or name.
- **Behaviors** displays the behaviors assigned to the cast member.
- **Ink Mode** displays the type of ink applied to the cast member.
- **Blend** displays the blend percentage applied to the cast member.
- **Location** shows the X & Y screen coordinates of the cast member.
- **Change in Location** indicates the change in X and Y coordinates relative to the previous cast member in that channel.

Options

- **Script Preview** displays the Script Preview box at the top of the Score.
- **Show Data Tips** displays data tips (cast member name and number) when the pointer is over a sprite for a few seconds.

Compatibility

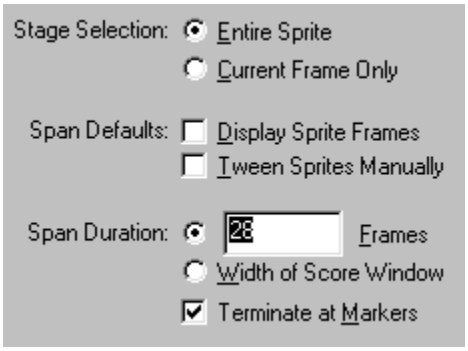
- **Director 5 Style Score Display** modifies the Director 6 Score window to look and behave like the Director 5 Score window.
- **Allow Drag and Drop** allows sections of score to be moved by dragging in the Director 5 Score. Pressing the Spacebar while the Score window is open temporarily overrides this setting.
- **Allow Colored Cells** allows you to choose a color for selected cells using the cell color selector on the left side of the Score window. Otherwise, the cell color selector is hidden.

  If you've already applied color to cells, turning off this option hides cell colors but doesn't remove them. Score window scrolling performance is faster if you hide cell colors.

Preferences > Sprite (File menu)

This command determines how sprites are displayed in the Score window and on the Stage.

Click a dialog box option for more information:



{button See also,AL(`Preferences_Sprite_help')}

Stage Selection determines whether the entire sprite or a single frame within the sprite is selected in the Score window when a sprite is selected on the Stage.

- **Entire Sprite** specifies that selecting a sprite on the Stage selects the sprite in all the frames it occupies.
- **Current Frame Only** specifies that selecting a sprite on the Stage selects only the current frame of the sprite.

Span Defaults determines the appearance and behavior of sprites yet to be created. 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**.
- **Tween Size and Position** turns on automatic tweening for size and position for all new sprites. With this option off, sprites must be manually tweened when new frames or keyframes are added to the sprite.

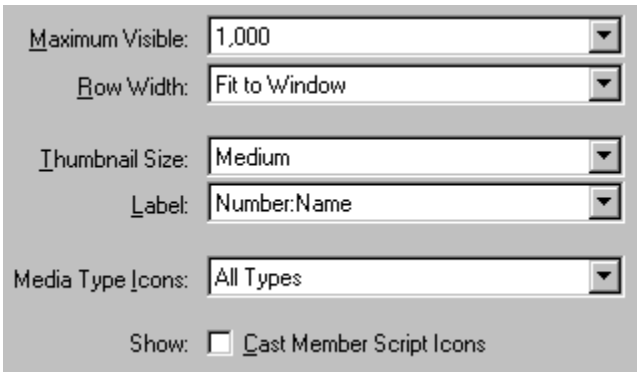
Span Duration determines the length of the sprite, or sprite span, measured in frames.

- **_ 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 encountered.

Preferences > Cast Window command (File menu)

The Cast Window Preferences command displays a dialog box that lets you control the appearance of the current cast window. Before you choose Cast Preferences, make sure that the cast you want to change is active. The title bar displays the name of the cast you are changing.

Click a dialog box option for more information:



Maximum Visible specifies the maximum number of cast members displayed in the Cast window. 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 and make better use of the vertical scroll bar. The default is 1000.

Row Width determines how many thumbnails are displayed in each row in the Cast window. 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.

Thumbnail Size sets the size of each cast thumbnail image displayed in the Cast window. Thumbnails always maintain the standard 4:3 aspect ratio.

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

Tip: If thumbnails appear fuzzy, they were probably created at a small size and are now set to a larger size. To fix this problem, change the Cast window preferences thumbnail setting to a larger size. Click OK when the alert message asks you if thumbnails should be regenerated.

Label selects the display format of the cast member ID displayed below each cast thumbnail image in the Cast window. The chosen format is also used in other windows, whenever a cast ID is displayed, including the Score. The default is Number:Name.

- **Number** displays cast number in decimal format.
- **Name** displays cast name, if one exists; otherwise displays cast number in decimal format.
- **Number:Name** displays the cast number (in decimal format) and cast name, separated by a colon, for example, "340:Jackie O". If a name does not exist, it just displays the cast number in decimal format. This is the default.

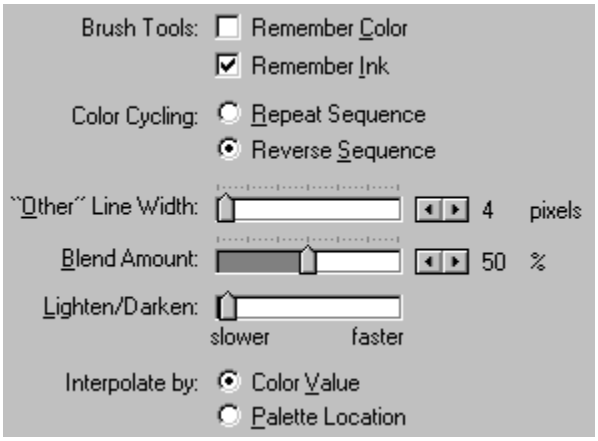
Media Type Icons determines if Director displays an icon in the lower right corner of each cast member, indicating the cast member's type.

Cast Member Script Icons makes a script indicator icon appear in the lower left corner of each cast member that has a script attached.

Preferences > Paint window command (File menu)

The Paint window Preferences command allows you to modify the settings of a number of tools and drawing methods in the Paint window.

Click a dialog box option for more information:



{button See also,AL(`Paint_Window_Options_help')}

Brush Tools allows you to set brush tools to remember the last color or ink used.

- **Remember Color** remembers the last color used with a tool and stays selected for the next time you use the paintbrush or air brush.
- **Remember Ink** remembers the last ink used with a tool and stays selected for the next time you use the paintbrush or air brush.

Color Cycling controls the way colors cycle when you draw with cycling ink.

- **Repeat Sequence** causes colors to cycle from foreground to destination and then repeat foreground to destination.
- **Reverse Sequence** causes colors to cycle from the foreground to the destination color and then destination to foreground.

"Other" Line Width allows you to set a thicker line width than the widths available in the **Paint window**. The width you set will be the width that appears when you draw a line (after selecting Other Line Width in the Tool Palette line width selector).

Blend Amount sets the opacity of the selected color when using the Blend ink effect in the Paint window. You can vary the blend value between 0 and 100 percent.

Lighten/Darken sets the rate at which artwork changes when you use the Darken or Lighten effect in the Paint window.

Interpolate by determines how colors are used when using smooth, lighten, darken, or cycle effects.

- **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.

Preferences > Editors command (File menu)

This command opens the External Editors Preferences dialog box. Use this box to associate an external editor with a bitmap, video, or sound cast member.

Initially, all cast members are set to Director's internal default editor. To switch to an external editor, highlight a cast member type and select the Edit option.

{button See also,AL(`Preferences_Editors_help')}

Exit command (File menu) Alt-F4

The Exit command exits Director.

Edit menu

The Edit menu contains standard commands for editing.

Click a command for more information:

Undo
Repeat
Cut
Copy
Paste
Paste Special
Clear
Duplicate
Select All
Invert Selection

Find > Text
Find > Handler
Find > Cast Member
Find > Selection
Find Again
Replace Again
Edit Sprite Frames
Edit Entire Sprite
Exchange Cast Members
Edit Cast Member
Launch External Editor

Undo command (Edit menu) Control-Z

Undo reverses your last action. Undo works with most commands you use while writing, drawing, and animating.

Repeat command (Edit menu) Control-Y

The Repeat command repeats your last action. Repeat works with paint effects commands.

If using **Photoshop filters** to modify cast members, this command repeats the effect of the last filter used.

Cut command (Edit menu) Control-X

The Cut command removes the selected object from its current location and places it on the Clipboard. It can then be pasted to another location.

Copy command (Edit menu) Control-C

The Copy command makes a copy of the selected colors, text, art, or sequence of art and places that copy on the Clipboard.

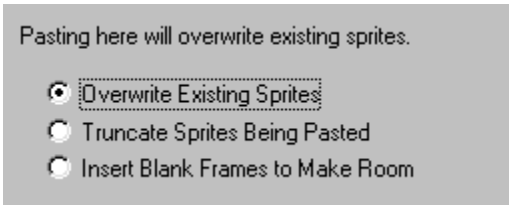
Paste command (Edit menu) Control-V

The Paste command pastes the contents of the Clipboard in a selected location.

{button See also,AL(`Paste')}

Paste Options

This dialog box appears if a paste operation will overwrite existing sprites.



Choose one of the following options:

Overwrite Existing Sprites overwrites existing sprites with the content of the Clipboard.

Truncate Sprites Being Pasted pastes the content of the Clipboard in the space available and doesn't overwrites existing sprites.

Insert Blank Frames to Make Room creates new frames for the contents of the Clipboard.

{button See also,AL(`Paste_Options')}

Paste Special command (Edit menu)

Use this command to paste a sequence of sprites at the point on the Stage where a previous sequence ended. When you use this command, Director automatically adjusts the positions of cast members on the Stage so that the first cast member in the pasted sequence follows the last cast member in the original sequence.

For example, to animate a ball bouncing across the Stage with a sequence of five sprites that describe one bounce, you can use Paste Relative to make the second sequence of sprites start where the first sequence ended. The effect is that the two sequences are chained, one after another, in one smooth, continuous motion. This works for any repetitive sequence of sprites.

When selecting cells in a sequence to be pasted relative to the original sequence, make sure the same cast member is at the beginning and end of the sequence, and that you overlap the first cell in the copy with the last cell in the original sequence.

{button See also,AL(`Paste_Special')}

Clear command (Edit menu)

Clear removes the selected frames or cast members without saving to the Clipboard. When the Score window is active, Clear removes the contents of selected sprite frames. When a Cast window is active, Clear removes selected cast members.

Delete is the keyboard shortcut for this command.

Duplicate command (Edit menu) Control-D

The Duplicate command duplicates the selected cast member and pastes the duplicate into the next available position in the cast window.

If the Paint, Text, Video, or Script window is open, this command duplicates the selected cast member and places it in the Edit window. The name and registration point of the duplicate is the same as the name and registration point of the original cast member.

This is a quick way to create a series of cast members for frame-by-frame animation. Duplicate a cast member, change it slightly, duplicate the changed cast member, alter and duplicate it again, and so on.

Tip: If the Cast window is front-most, you can select multiple cast members and use this command to duplicate the entire selection at the same time.

You can also duplicate by Option-dragging selected cast members into empty cast spaces.

{button See also,AL(`Duplicate_Cast_Member_help')}

Select All command (Edit menu) Control-A

Select All highlights all the selectable items in the active window.

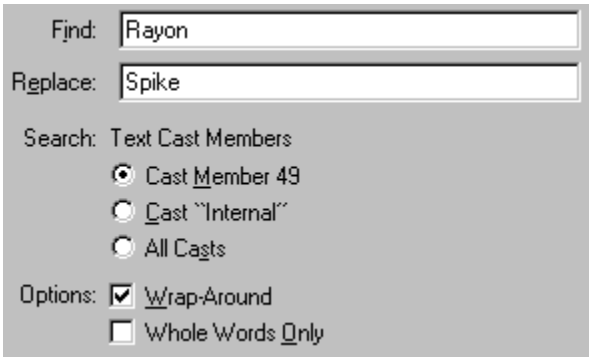
Invert Selection command (Edit menu)

When you choose Invert Selection after choosing a color or range of colors in the Color Palettes window, your selection is replaced by a new selection, which consists of all the colors that were not part of your original selection. This command only works when working with color palettes.

Find > Text command (Edit menu) Control-F

Find Text lets you quickly search for and replace text in the Text, Field, or Script windows. All searches start at the insertion point and search forward.

Click a dialog box option for more information:



{button See also,AL(`Find_Change_help')}

Find specifies the text you want to find. Searching is not case-sensitive: ThisHandler, thisHandler, and THISHANDLER are all the same for search purposes.

Replace specifies the replacement text.

Search specifies which cast members to search.

- **Cast Member _** limits the search to the current cast member.
- **Cast _** limits the search to cast members in the current cast.
- **All Casts** extends the search to all cast members in all casts.

Options

- **Wrap-Around** specifies whether or not Director returns to the beginning of text once it reaches the end. If this option is turned on but All Casts is off, Director continues searching from the top of the current text after it reaches the bottom of the window. If both options are checked, Director searches all cast members of the same type (either text 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 for occurrences of the specified whole word.

Find > Handler command (Edit menu) Control-Shift-;

Find Handler lets you view the names of all handlers in the current script or movie. You can also use this command to open the Script Window that contains the selected handler.

To find a handler:

Choose Edit > Find Handler. Select a handler and click OK to open the Script Window in which the handler is defined.

Dialog box options

Search determines which Scripts to search.

- **Current Script Only** limits the search to handlers defined in the current script. It is only available if you choose Find Handler from a Script Window.

View by determines how handlers will be listed.

- **Name** lists handlers alphabetically, by name.
- **Script Order** lists handlers in the order in which they appear in their respective scripts. Handlers from different scripts are listed in the order in which the scripts appear in the cast window.

{button See also,AL(`Find_Handler_help')}

Find > Cast Member command (Edit menu) Control-;

Use this command to find cast members by name, by type, by color palette, or by their use in the current Score. This command is useful for identifying cast members to clear from your movie or to remap to another palette.

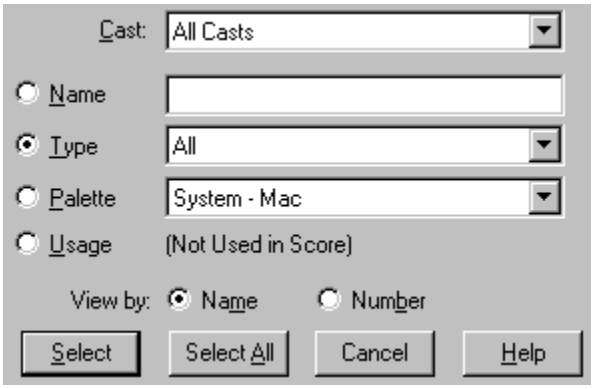
To find cast members:

Choose the cast you want to search from the Cast pop-up and then choose one of the search options. Cast member names and number appear in the bottom pane.

- Select a cast member in the list and click Select (or double-click a cast member) to close the dialog box and select the cast member in the cast.
- Click **Select All** to close the dialog box and select all matching cast members in the cast.

Tip: To quickly select cast members by name, type the first few letters of the name, and the dialog box automatically displays a list of cast members whose name begins with the letters you type.

Click a dialog box option for more information:



{button See also,AL(`Find_Cast_Member_help')}

Cast selects the cast to search.

Name searches for all cast members with names that begin with the characters you enter. Click Name and enter the cast member name you want to search for in the field to the right.

Type specifies the type of cast members to search for. Select Type and choose an option from the pop-up.

Palette searches for all cast members using a certain palette.

Usage finds all cast members not used in the Score. Keep in mind, however, that a cast member that is not used in the Score may still be used in a Lingo command.

View by determines how cast members are displayed on the list.

- **Name** displays cast members by name.
- **Number** displays cast members by number.

Select selects the cast member highlighted in the Find Cast Member dialog box.

Select All selects all the cast members in the Find Cast Member dialog box.

Find > Selection command (Edit menu) Control-H

Select a cast member in either the Cast window or the Score and use Find Selection to find the next occurrence of that cast member in the Score.

Find Again repeats the previous find, initiated in either the Cast or Score window.

Find Again command (Edit menu) Control-Alt-F

Find Again finds the next occurrence of the text you entered in the Find field in the Find dialog box.

Replace Again command (Edit menu) Control-Alt-E

Choose Replace Again to replace the next instance of the text you entered in the Find field in the Find Text dialog box.

Edit Sprite Frames command (Edit menu)

The Edit Sprite Frames command changes how a sprite is selected and how keyframes are created. 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.

Alt-double-clicking (Windows) or Option-double-clicking (Macintosh) a sprite is the equivalent of choosing this command.

{button See also,AL(`Edit_Sprite_Frames_help')}

Edit Entire Sprite command (Edit menu)

The Edit Entire Sprite command returns sprites to their normal state when Edit Sprite Frames is on. Alt-double-clicking (Windows) or Option-double-clicking (Macintosh) a frame within the sprite is the equivalent of choosing this command.

{button See also,AL(`Edit_Entire_Sprite_help')}

Exchange Cast Members command (Edit menu) Control-E

The Exchange Cast Members command replaces the cast member selected on the Stage or in the Score with the cast member selected in the Cast window. When you use Exchange Cast Members, the registration point of the new cast member lines up with the registration point of the old cast member.

{button See also,AL(`Exchange_Cast_Members_help')}

Edit Cast Member command (Edit menu)

This command is only available if there is a selection in the cast window. It displays the appropriate editing window for the selected cast member. For example, if you select a bitmap cast member and choose this command, Director opens the Paint window for the selected cast member. For cast members that don't have editing windows (such as shape, PICT, sound, movie, and film loop cast members) Director displays the cast member's **Cast Member Properties** dialog box.

Shortcut: Double-click a cast member in the cast window.

Launch External Editor command (Edit menu)

This command launches an external editor for the selected cast member.

{button See also,AL(`Launch_Editor_help')}

View menu

Click a command or submenu for more information:

Marker > Previous
Marker > Next
Display
Zoom
Grids > Show
Grids > Snap To
Grids > Settings
Rulers
Sprite Overlay > Show Info
Sprite Overlay > Show Paths
Sprite Overlay > Settings
Sprite Toolbar
Keyframes
Sprite Labels
Onion Skin

Depending on which window you have in front, the **Panel** command name changes (such as Paint Tools or Text Toolbar).

Display command (View menu)

Choose the type of display for the Score window. The same choices are also available in the Display pop-up in the Score.

{button See also,AL(`Display_submenu_help')}

Marker > Next command (View menu) Control-right

Choose an option to move to the next marker in the Score, or select the name of a marker to move there directly.

{button See also,AL(`Marker_Next_help')}

Marker > Previous command (View menu) Control-left

Choose an option to move to the previous marker in the Score, or select the name of a marker to move there directly.

{button See also,AL(`Marker_Previous_help')}

Zoom submenu (View menu)

Narrower Control-Subtract

Choose Narrower to make frames in the Score smaller and display more of them at once.

Wider Control-Add

Choose Wider to make frames in the Score wider so more detail is visible in sprite labels.

Zoom In Control-Add

Choose Zoom In to change the size of the Paint window, or the width of frames in the Score. You can zoom between 100% to 800%.

When you zoom in, a smaller representation of the 100% size artwork appears in the upper right corner of the Paint window. To return to normal size, click inside the smaller window or choose Zoom Out.

Zoom Out Control-Subtract

Choose Zoom Out to enlarge the Paint window. You can zoom from 100% to 800%.

To return to normal size, click inside the smaller window or choose Zoom In.

Zoom 100, 200, 400, 800%

Choose one of these to view the Paint window or Score at a specific scale.

Tip: You can also magnify your artwork by double-clicking the pencil tool in the Paint window's tool palette, or by Control-clicking the area you want to enlarge with any of the paint tools.

Rulers command (View menu)

Choose Rulers from the View menu to show or hide rulers in the paint or Text windows.

The default setting for the unit of measure is inches. You can change the setting for text to picas, centimeters, or pixels using **General Preferences** in the File menu.

In the Paint window, change the unit of measure by clicking the upper left corner where the vertical and horizontal rulers meet. The zero point of the rulers can be moved to a new location by dragging from the corner of the rulers. The current position of the pointer is indicated by dotted lines in the rulers.

Grids > Show command (View menu) Command-Shift-Alt-G

The commands on the Grids submenu control the grid on the Stage. Use the grid to align and place sprites on the Stage.

Show shows or hides the grid on the Stage. A checkmark indicates the grid is displayed.

{button See also,AL(`Show_Grid_help')}

Grids > Snap To command (View menu) Control-Option-G

The commands on the Grids submenu control the grid on the Stage. Use the grid to align and place sprites on the Stage.

Snap To makes all sprites move to the nearest grid line when you move them. A check mark indicates the option is on.

{button See also,AL(`Snap_Grid_help')}

Grids > Settings command (View menu)

The commands on the Grids submenu control the grid on the Stage. Use the grid to align and place sprites on the Stage.

Dialog box options

Grid Settings defines the settings for the grid. In the Grid Settings dialog box you can define the following settings:

Spacing specifies the number of pixels between the vertical and horizontal grid markings.

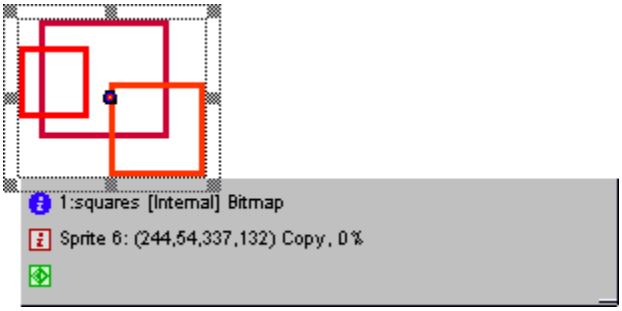
Display determines if the grid appears as lines or dots.

Color defines the color for the grid from the pop-up color palette.

{button See also,AL(`Grid_Settings_help')}

Sprite Overlay > Show Info command (View menu)

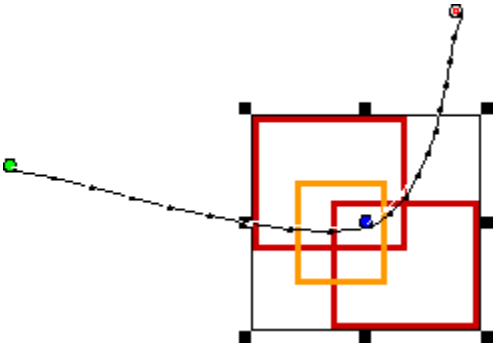
This command displays the most important sprite properties directly on Stage. Each sprite visible on Stage has its own sprite overlay. The sprite overlay is dynamically updated and always reflects the current property values of the sprite.



{button See also,AL(`Sprite_Overlay_Info')}

Sprite Overlay > Show Paths command (View menu)

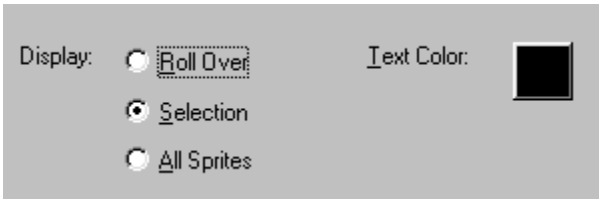
This command displays the path of moving sprites on the Stage. Keyframes appear as hollow circles. Small tick marks show the sprite's position in tweened frames.



{button See also,AL(`Sprite_Overlay_Paths')}

Sprite Overlay > Settings command (View menu)

This command opens the Overlay Setting dialog box, which determines how sprite overlay properties appear on the Stage.



Dialog box options

Display determines how sprite properties appear on the Stage:

- **Roll Over** makes sprite properties visible and active when the mouse rolls over a single sprite.
- **Selection** makes sprite properties visible and active when a sprite is selected.
- **All Sprites** makes sprite properties visible and active for all sprites on the Stage.

Text Color determines the text color displayed in the sprite overlay.

{button See also,AL(`Sprite_Overlay_Settings')}

Keyframes command (View menu)

This command suppresses display of all but the head and tail sprites in the Score.

{button See also,AL(`KeyFrames_View_help')}

Panel command (View menu) Control-Shift-H

Choose Panel from the View menu to show or hide tool panels in the Text, Paint, Color Palette, and Script windows.

The command name in the View menu varies, depending on which window is active. If the active window has a toolbar, the name of the toolbar appears. If the active window does not have a toolbar, the word Panel appears.

Active window:	View menu command:
Field window	Text Toolbar
Message window	Message Toolbar
Paint window	Paint Tools
Script Window	Script Toolbar
Score window	Sprite Toolbar
Text window	Text Toolbar
Watcher window	Panel

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Sprite Toolbar command (View menu)

This option displays the Sprite Inspector as part of the Score window.

Sprite Labels command (View menu)

This command displays the sprite label of the sprite in the Score window. The label can represent the name of the underlying castmember or the value of any property.

Choose one of the following options:

- **Keyframes** displays sprite labels at each keyframe.
- **Changes Only** displays sprite labels when a tweenable property within the sprite changes.
- **Every Frame** displays a sprite label for each frame. Use the View > Zoom menu option to adjust the Score, if necessary. This option can reduce performance in the authoring environment.
- **First Frame** displays sprite labels at the first frame of the sprite.
- **None** displays no sprite labels.

{button See also,AL(`Sprite_Labels_help')}

Onion Skin command (View menu)

Onion skinning allows you to view several cast members blended into an image in the Paint window. The blended-in cast members serve as a reference while you paint a new cast member.

To activate onion skinning, choose View > Onion Skin. The Onion Skin toolbar appears. Click the Toggle Onion Skin button in the toolbar to enable onion skinning. See the **Using onion skinning** help topic.

Click a toolbar button for more information:

{button See also,AL(`Onion_Skin_help')}

Toggle Onion Skinning is an on/off button. When off, no reference images are blended into the Paint window's drawing area. When on, selected reference images are blended into the drawing area. The current cast member is then drawn on top. **Shortcut:** Control-Alt-K enables or disables onion skinning.

Preceding Cast Members indicates the number of cast members immediately preceding the current cast member shown as reference images in the Paint window. Images further away from the current cast member are dimmer than images closer to it in the cast.

Following Cast Members indicates the number of cast members immediately following the current cast member that will be shown as reference images in the Paint window. Images further away from the current cast member are dimmer than images closer to it in the cast.

Set Background selects the current cast member to be the background cast member. This cast member will be used as a reference image if Show Background is on.

Show Background blends the background cast member into the Paint window as a reference image.

Track Background marks the current foreground and background cast members as the beginning members of a foreground and background series of cast members. From then on, as you select different foreground cast members, the corresponding member of the background series is used as a reference image.

Insert menu

Click a command or submenu for more information:

Keyframe
Remove Keyframe
Frame
Remove Frame
Marker
Media Element > Bitmap
Media Element > Shockwave Audio
Media Element > Text
Media Element > Color Palette
Control > Push Button
Control > Radio Button
Control > Check Box
Control > Field
Control > Custom Button
OLE Object
Film Loop

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Marker command (Insert menu)

The Marker command inserts a marker into the Score at the current position of the playback head.

{button See also,AL(`Marker_Insert_help')}

Frames command (Insert menu) Control-Shift-]

The Frames command inserts a specified number of frames into your movie at the location of the playback head.

The new frame is inserted for all channels and adds a frame to the movie's length.

{button See also,AL(`Insert_Frame_help')}

Remove Frame command (Insert menu) Control-[

Remove Frame deletes the frame at the location of the playback head, making the movie one frame shorter in length.

Keyframe command (Insert menu) Control-Alt-K

This command inserts a new keyframe at the location of the Playback Head.

{button See also,AL(`Keyframe_Insert_help')}

Remove Keyframe command (Insert menu)

Remove Keyframe deletes the selected key frame(s).

{button See also,AL(`Remove_Keyframe_help')}

Media Element > Bitmap command (Insert menu)

Choose Insert > Media Element > Bitmap to create a new bitmapped cast member. When you choose this command, Director opens the Paint window.

Unless you first select an empty position in the cast window, Director assigns the new cast member to the first empty position after the current cast member.

Media Element > Text command (Insert menu)

Choose Insert > Media Element > Text to create a new text cast member and open the Text window.

Unless you first select an empty position in the cast window, Director assigns the new cast member to the first empty position after the current cast member.

{button See also,AL(`New_Cast_Member_Text_help')}

Media Element > Color Palette command (Insert menu)

Choose Insert > Media Element > Color Palette command to create a new palette cast member. When you choose this command a dialog box appears and prompts you to enter the name of the new color palette. Once you enter a name and click OK, Director opens the color palettes window and you can change the new palette as needed.

Unless you first select an empty position in the cast window, Director assigns the new cast member to the first empty position after the current cast member.

{button See also,AL(`New_Cast_Member_Palette_help')}

Control > Push Button, Radio Button, or Check Box (Insert menu)

Choose one of these commands to create a button or check box cast member on the Stage. First, select the channel and frame in the Score where you want to create the button or check box, and then choose the command. A button or check box will appear on the Stage and you can begin entering text.

Unless you first select an empty position in the cast window, Director assigns the new cast member to the first empty position after the current cast member.

You must then attach a **script** to a button or check box so that it responds appropriately when clicked.

Control > Field command (Insert menu)

Use this command to create a new field cast member. First, select the channel and frame in the Score where you want to create the new field, and then select the Field command. A field will appear on the Stage and you can begin entering text.

Unless you first select an empty position in the cast window, Director assigns the new cast member to the first empty position after the current cast member.

Use fields primarily for text that must be editable in a projector. For most text applications you should use normal text.

{button See also,AL(`New_Cast_Member_Field_help')}

Control > Custom Button command (Insert menu)

This command opens the Button Editor. The Button Editor creates a single cast member that displays on, off, rollover, and disabled conditions automatically, without any scripting or references to other cast members.

Settings/Bitmaps tabs switch between panels containing different button options.

- **Settings** provides controls that determine the button text, font, button type, and initial state of the button.
- **Bitmaps** provides fields in which you paste images of the button in all its states.

Label provides a box in which you enter the text that appears on the button.

Font opens the Font dialog box. Use it to specify the font, size, style, and kerning of the button text.

Button Type provides two options, Push Button and Toggle Button.

- **Push Button** creates a button that reverts to its normal state after being clicked.
- **Toggle Button** creates a button that remains in the toggled state after being clicked.

Initial State provides options for the initial setting of the button.

- **Enabled** makes the button enabled by default.
- **Toggled** makes a toggle button on by default (not available for push buttons).

{button See also,AL(`Custom_Button_help')}

Media Element > Shockwave Audio

Creates a Streaming Shockwave Audio cast member. The cast member created by using this command does not do anything until you use Cast Member Properties to link the cast member to an external Shockwave Audio file.

{button See also,AL(`SWA_Streaming_Xtra')}

OLE Object command (Insert menu)

The OLE Object command creates cast members from OLE objects. You can create a new OLE Object using the source application, or by linking to an existing file.

Note: OLE objects only work in Windows 95 and Windows NT.

{button See also,AL(`New_OLE_Object_help')}

Film Loop command (Insert menu)

The Film Loop command changes a selected animation sequence into a repeating loop that appears as a single cast member.

{button See also,AL(`Create_Film_Loop_help')}

Modify menu

Click the name of a menu command for more information:

Cast Properties	**Font**
Cast Member > Properties	**Paragraph**
Cast Member > Script	**Borders**
Sprite > Properties	**Join Sprites**
Sprite > Script	**Split Sprite**
Sprite > Tweening	**Extend Sprite**
Frame > Tempo	**Arrange**
Frame > Palette	**Align**
Frame > Transition	**Tweak**
Frame > Sound	**Reverse Sequence**
Frame > Script	**Sort**
Movie > Properties	**Cast to Time**
Movie > Casts	**Space to Time**
Movie > Xtras	**Transform Bitmap**
Movie > Playback	**Convert to Bitmap**

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Cast Properties command (Modify menu)

Use the Cast Properties command to view properties and change settings for the currently selected cast.

Dialog box options

Name displays the name of the current cast for you to view or change.

Storage indicates whether the cast is internal or external and, if the cast is external, where it is stored.

Size displays the size of the cast in kilobytes.

Preload defines how the cast is loaded into memory when the movie runs. The choices are:

- **When Needed:** The cast does not load into memory until it is required by the movie.
- **After Frame One:** The cast is loaded when the movie exits frame one.
- **Before Frame One:** The cast is loaded before the movie plays frame one.

{button See also,AL(`Cast_Properties_help')}

Cast Member Properties command (Modify menu)

Cast Member Properties displays a dialog box containing information about the selected cast member: its name, cast position, type, and its size in kilobytes. The Cast Member Properties dialog box also displays additional information and options for each cast member type. Use the options in the dialog box to define the behavior and appearance of the selected cast members.

The type of information in the Cast Member Properties dialog box depends on the type of cast member.

Bitmap	**PICT**
Button	**Script**
Digital Video	**Shape**
Field	**Sound**
Film Loop	**Field**
Director Movie	**Text**
Palette	**Transition**
Multiple cast members selected	**Xtras**

 In the cast window, select a cast member and click the Info button as a shortcut for choosing this command. If you are editing the cast member in the paint, text, digital video, or Script Window, you can click the window's Info button as a shortcut for choosing this command.

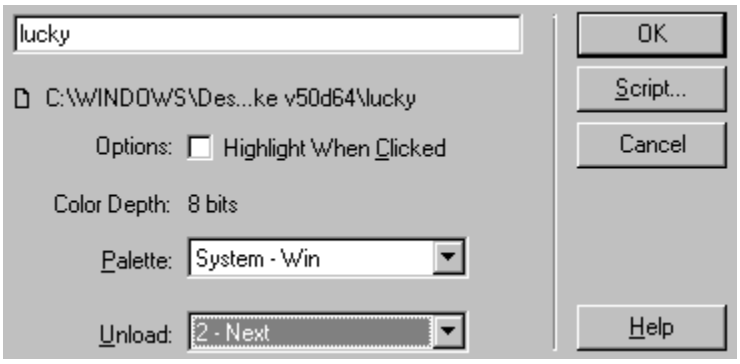
Tip: Select a cast member in the cast or on the Stage and then right-click and choose Cast Member Properties from the pop-up.

Cast Member Properties > Bitmap (Modify menu)

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- Dimensions
- The size in kilobytes

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

File Name displays the location of linked external files. This appears only if the selected cast member is linked to an external file. Click the file name to choose a new location.

Options: Highlight When Clicked makes the current cast member invert when it is clicked by the user. Use this option to create buttons. Even if Highlight When Clicked is checked, the cast member will not do anything unless it is controlled by a Lingo script.

Color Depth displays the color depth of the cast member.

Palette assigns a different palette to the cast member, while maintaining the cast member's original palette references, so the image is not changed. You can change the palette assignment at any time by choosing another palette from the menu.

Unload controls how Director removes the cast member from memory if memory is low.

- **3-Normal:** The selected cast member will be removed from memory as necessary.
- **2-Next:** The selected cast member will be among the next to be removed from memory.
- **1-Last:** The selected cast member will be the last to be removed from memory.
- **0-Never:** The selected cast member remains in memory and is never purged.

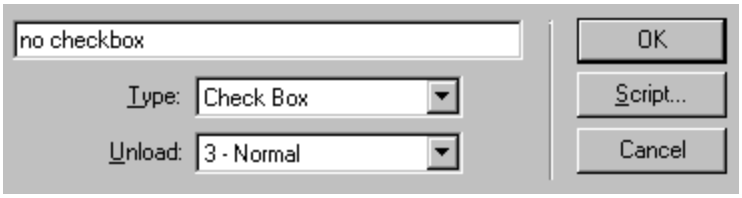
Script opens a Script Window for the cast member. The script remains attached to the cast member if the cast member is cut or copied and pasted.

Cast Member Properties > Button (Modify menu)

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

Type lists the button types-push button, check box, or radio button.

Unload controls how Director removes the cast member from memory if memory is low. Choose one of these options from the pop-up.

- **3-Normal:** The selected cast member will be removed from memory as necessary.
- **2-Next:** The selected cast member will be among the next to be removed from memory.
- **1-Last:** The selected cast member will be the last to be removed from memory.
- **0-Never:** The selected cast member remains in memory and is never purged.

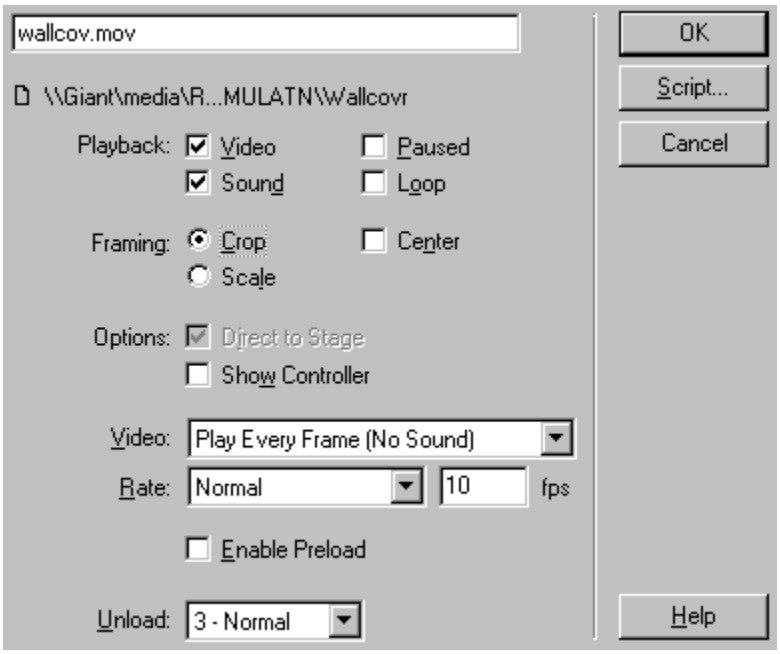
Script opens a Script Window for the cast member. The script remains attached to the cast member if the cast member is cut or copied and pasted.

Cast Member Properties > Digital Video (Modify menu)

Use the Cast Member Properties for digital video cast members to view information about the current cast member and to change optional settings. The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The length of the movie in seconds
- The size in kilobytes
- Dimensions

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script so that you don't have to update your scripts if your cast members are renumbered or sorted.

File Name displays the location of the digital video. Click the file name to choose a new location.

Video plays the video portion of the digital video. If turned off, the video portion does not play. Deselect this option and check Sound if you want to play the audio-only portion of a movie.

Sound plays the sound portion of the digital video.

Paused pauses the digital video when it first appears on the Stage (while playing the Director movie).

By default, a digital video starts playing the moment it first appears. If you check Paused, you can later start the movie using the statement:

```
set the movieRate of sprite n to R
```

where:

- `n` is the sprite number in the current frame
- `R` is a number representing the rate. For example, 0 = stop, 1 = normal speed, 2 = 2x speed, and -1 = reverse.

Loop loops the digital video from the end back to the beginning and continues to play.

{button See also,AL(`QuickTime_Cast_Info_help')}

Crop retains the movie's original size if you resize the bounding rectangle. The edges of the movie may be clipped.

Center centers the movie when you resize the bounding rectangle. If Center is not checked, the loop maintains its original position when you resize its bounding rectangle. Center is only available if Crop is checked.

Scale scales the movie if you resize the bounding rectangle.

Direct to Stage plays the movie in front of any cast members on the Stage, regardless of the channel that contains the movie. Inks are not visible on a movie that plays with this option. In general, use Direct to Stage when you want the best possible performance from a digital video and you don't need ink effects or compositing. Results may vary, so you may have to experiment. For more information, see the **Using Direct to Stage** help topic.

Show Controller displays a controller bar below the movie to allow the user to start, stop, and step through the movie. This option is only available if Direct to Stage is checked.

Video defines how the movie is synchronized. If you choose Play Every Frame, every frame of the digital video plays. The digital video's soundtrack will not play, since the movie can't play the soundtrack asynchronously while the video portion plays frame-by-frame. If you choose Sync to Soundtrack, the movie skips frames as necessary to keep up with the tempo of the soundtrack. These options are only available if Direct to Stage is checked.

Rate determines at what rate the movie plays. The options on the Rate menu are only available if Play Every Frame is checked. The following options appear on the Rate pop-up:

- **Normal**: Each frame plays at its normal rate, and no frames are skipped.
- **Maximum**: The movie plays as fast as possible while still displaying each frame.
- **Fixed**: Play the movie using a specific frame rate. Enter the number of frames per second in the field to the right. Use this option only for digital videos that use the same frame rate for each frame of the movie.

Enable Preload preloads the entire movie (or as much of the movie as can fit into available memory) using the **`preLoad`** or **`preLoadMember`** Lingo commands. If there is not enough memory to load the entire movie, Director loads only what can 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.

Unload controls how Director removes the cast member from memory if memory is low. Choose one of these options from the pop-up.

- **3-Normal:** The selected cast member will be removed from memory as necessary.
- **2-Next:** The selected cast member will be among the next to be removed from memory.
- **1-Last:** The selected cast member will be the last to be removed from memory.
- **0-Never:** The selected cast member remains in memory and is never purged.

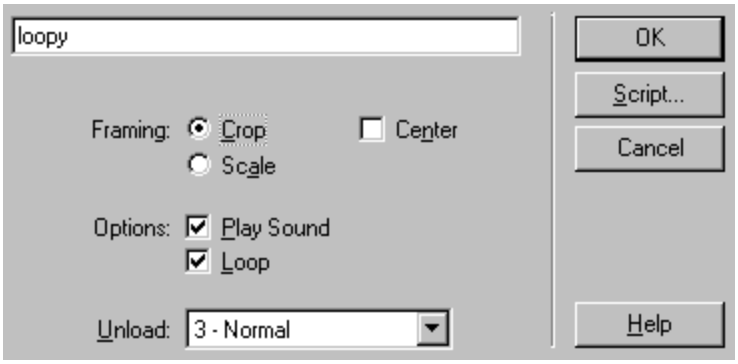
Script opens a Script Window for the cast. The script remains attached to the cast member if the cast member is cut or copied and pasted.

Cast Member Properties > Film Loop (Modify menu)

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script so that you don't have to update your scripts if your cast members are renumbered or sorted.

Crop retains the film loop's original size if you resize the bounding rectangle of the sprite. The edges of the movie may be clipped.

Scale scales the film loop if you resize the sprite bounding rectangle.

Center centers the film loop when you resize the bounding rectangle. If Center is not checked, the loop maintains its original position when you resize its bounding rectangle. Center is only available if Crop is checked.

Play Sound enables sound during playback. If not checked, sound is disabled during playback.

Loop returns the animation from the last frame back to the first and continues to play. If this option is not checked, the animation doesn't loop, and the last frame remains on the Stage.

Unload controls how Director removes the cast member from memory if memory is low. Choose one of these options from the pop-up.

- **3-Normal:** The selected cast member will be removed from memory as necessary.
- **2-Next:** The selected cast member will be among the next to be removed from memory.
- **1-Last:** The selected cast member will be the last to be removed from memory.
- **0-Never:** The selected cast member remains in memory and is never purged.

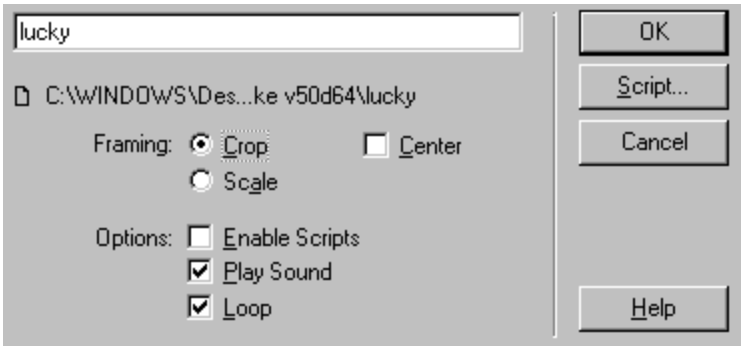
Script opens a Script Window for the cast member. The script remains attached to the cast member if the cast member is cut or copied and pasted.

Cast Member Properties > Director Movie (Modify menu)

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script so that you don't have to update your scripts if your cast members are renumbered or sorted.

File Name displays the location of the external file associated with the linked movie. Clicking the file name lets you choose a new location.

Crop retains the linked Director movie's original size if you resize the bounding rectangle. The edges of the movie may be clipped.

Scale scales the movie if you resize the bounding rectangle.

Center centers the linked Director movie when you resize the bounding rectangle. If Center is not checked, the movie maintains its original position when you resize its bounding rectangle. Center is only available if Crop is checked.

Enable Scripts activates the linked movie's scripts when the movie is used in the Score. If this option is not checked, Director ignores the movie's scripts.

Play Sound enables sound during playback. If not checked, sound is disabled during playback.

Loop returns the movie from the last frame back to the beginning and continues to play. If this option is not checked, the movie doesn't loop, and the last frame remains on the Stage when the movie finishes playing.

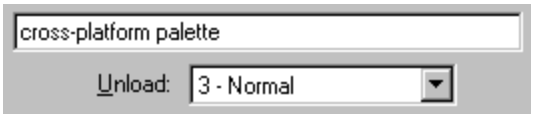
Script opens a Script Window for the cast member. The script remains attached to the cast member if the cast member is cut or copied and pasted.

Cast Member Properties > Palette (Modify menu)

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script so that you don't have to update your scripts if your cast members are renumbered or sorted.

Unload controls how Director removes the cast member from memory if memory is low. Choose one of these options from the pop-up.

- **3-Normal:** The selected cast member will be removed from memory as necessary.
- **2-Next:** The selected cast member will be among the next to be removed from memory.
- **1-Last:** The selected cast member will be the last to be removed from memory.
- **0-Never:** The selected cast member remains in memory and is never purged.

Cast Member Properties > PICT (Modify menu)

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes
- Dimensions

Click a dialog box option for more information:

Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

Unload controls how Director removes the cast member from memory if memory is low. Choose one of these options from the pop-up.

- **3-Normal:** The selected cast member will be removed from memory as necessary.
- **2-Next:** The selected cast member will be among the next to be removed from memory.
- **1-Last:** The selected cast member will be the last to be removed from memory.
- **0-Never:** The selected cast member remains in memory and is never purged.

Script opens a Script Window for the cast member. The script remains attached to the cast member if the cast member is cut or copied and pasted.

Cast Member Properties > Script (Modify menu)

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes

Click a dialog box option for more information:

Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script so that you don't have to update your scripts if your cast members are renumbered or sorted.

Type displays the script type (Movie, Score, or Parent) for the selected cast member and lets you change it. A movie script's handlers are global and can be called from other scripts. A Score script's handlers are local and cannot be called from other scripts. If you change a script into a Score script, it appears in the Script pop-up in the Score.

Cast Member Properties > Shape (Modify menu)

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes

Click a dialog box option for more information:

Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script so that you don't have to update your scripts if your cast members are renumbered or sorted.

Shape displays the current shape-rectangle, round rectangle, or oval-and lets you change the shape into any of the other available shapes.

Filled fills the currently selected shape with the current fill pattern and colors as specified in the **tool palette**.

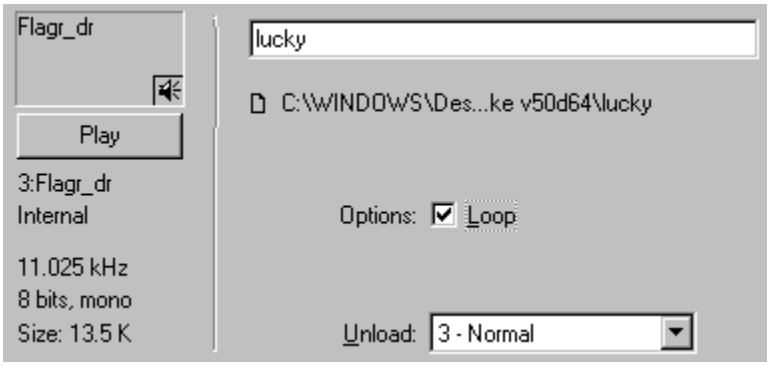
Script opens a Script Window for the cast member. The script remains attached to the cast member if the cast member is cut or copied and pasted.

Cast Member Properties > Sound (Modify menu)

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes
- Sample rate, sample size, and channels

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script so that you don't have to update your scripts if your cast members are renumbered or sorted.

File Name displays the location of linked external sound files. This appears only if the selected cast member is linked to an external file. Click the file name to choose a new location.

Loop makes the sound play continuously. If not checked, the sound plays once, even if the movie loops.

Unload controls how Director removes the cast member from memory if memory is low.

- **3-Normal:** The selected cast member will be removed from memory as necessary.
- **2-Next:** The selected cast member will be among the next to be removed from memory.
- **1-Last:** The selected cast member will be the last to be removed from memory.
- **0-Never:** The selected cast member remains in memory and is never purged.

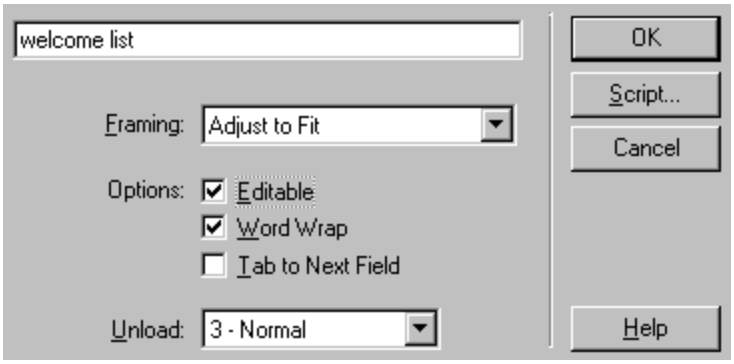
Play plays the sound at its pre-recorded sampling rate.

Cast Member Properties > Field (Modify menu)

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script so that you don't have to update your scripts if your cast members are renumbered or sorted.

Framing displays framing options for the current field.

- **Adjust to Fit** causes the field to expand vertically when text is entered that extends beyond the current size of the box.
- **Scrolling** attaches a scroll bar to the right side of the field. This is useful for a large amount of text.
- **Fixed** causes the box to retain its original size. If text is entered that extends beyond the limits of the box, the text is not displayed.
- **Limit to Field Size** sets the field's width to be fixed to the size of the field. Characters that don't fit are ignored.

Editable makes field cast members editable during movie playback. You can use this option instead of using the Lingo command `set the editable of sprite` to TRUE.

If you set a field cast member to be editable in the cast, it is always editable. Director overrides the Score's **editable check box** setting for the sprite.

Word Wrap makes words move to the next line when they reach the edge of the box. If turned off, text that extends beyond the right edge is truncated and you must use the Enter key to generate a new line.

Tab to Next Field causes the Tab key to advance the cursor to the next editable field on the Stage during playback. Note that the editable check box must be checked, or the Lingo command `set the editable of sprite to TRUE` must be specified for this option to have any effect.

Unload controls how Director removes the cast member from memory if memory is low. Choose one of these options from the pop-up.

- **3-Normal:** The selected cast member will be removed from memory as necessary.
- **2-Next:** The selected cast member will be among the next to be removed from memory.
- **1-Last:** The selected cast member will be the last to be removed from memory.
- **0-Never:** The selected cast member remains in memory and is never purged.

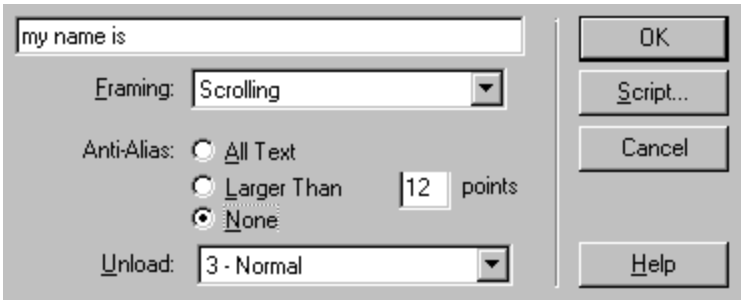
Script opens a Script Window for the cast member. The script remains attached to the cast member if the cast member is cut or copied and pasted.

Cast Member Properties > Text (Modify menu)

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script so that you don't have to update your scripts if your cast members are renumbered or sorted.

Framing displays framing options for the current text cast member.

- **Adjust to Fit** causes the text box to expand vertically when text is entered that extends beyond the current size of the box.
- **Scrolling** attaches a scroll bar to the right side of the text box. This is useful for a large amount of text.
- **Cropped** causes the text box to retain its original size. If you enter text that extends beyond the limits of the box, the text is not displayed.

Anti-Alias Text: 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.

- **All Text** anti-aliases all the text in the text block.
- **Larger Than** anti-aliases text larger than the point size entered in the points field.
- **None** turns off anti aliasing for the current cast member.

Unload controls how Director removes the cast member from memory if memory is low.

- **3-Normal:** The selected cast member will be removed from memory as necessary.
- **2-Next:** The selected cast member will be among the next to be removed from memory.
- **1-Last:** The selected cast member will be the last to be removed from memory.
- **0-Never:** The selected cast member remains in memory and is never purged.

Script opens a Script Window for the cast member. The script remains attached to the cast member if the cast member is cut or copied and pasted.

Cast Member Properties > Transition (Modify menu)

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes

Click a dialog box option for more information:

Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script so that you don't have to update your scripts if your cast members are renumbered or sorted.

Unload controls how Director removes the cast member from memory if memory is low. Choose one of these options from the pop-up:

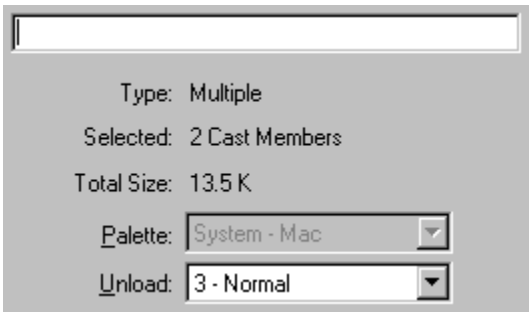
- **3-Normal:** The selected cast member will be removed from memory as necessary.
- **2-Next:** The selected cast member will be among the next to be removed from memory.
- **1-Last:** The selected cast member will be the last to be removed from memory.
- **0-Never:** The selected cast member remains in memory and is never purged.

Options is only available if you have installed Xtra transitions in the Xtras folder (within the Director application folder). The contents of the Options dialog box is determined by the developer of the Xtra. Refer to any documentation supplied with the Xtra.

Cast Member Properties > (Multiple items) (Modify menu)

Use Cast Member Properties to view and change settings for several selected cast members at once.

Click a dialog box option for more information:



Since multiple cast members are selected, no cast member name is shown.

Type displays "Multiple" when several cast members are selected, unless all the selected cast members are the same type.

Selected displays the number of cast members in the selection.

Total Size displays the total size, in kilobytes, of the selected cast members.

Palette lets you choose the palette used by the selected cast members.

Unload controls how Director removes the cast members from memory if memory is low. Choose one of these options from the pop-up:

- **3-Normal:** The selected cast members will be removed from memory after all purge priority 3 cast members have been purged.
- **2-Next:** The selected cast members will be among the next to be removed from memory.
- **1-Last:** The selected cast members will be the last to be removed from memory.
- **0-Never:** The selected cast members remain in memory and is never purged.

Cast Member Properties > Xtras (Modify menu)

Xtras are cast members created as plug-in extensions to Director. They can be new media types or add-on transitions. Xtra cast members may have additional settings accessible through an Options button.

Click a dialog box option for more information:

{button See also,AL(`Xtras_Cast_Info_help')}

Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

Unload controls how Director removes the cast member from memory if memory is low. Choose one of these options from the pop-up:

- **3-Normal:** The selected cast member will be removed from memory as necessary.
- **2-Next:** The selected cast member will be among the next to be removed from memory.
- **1-Last:** The selected cast member will be the last to be removed from memory.
- **0-Never:** The selected cast member remains in memory and is never purged.

Options opens a dialog box containing special controls and options for the Xtra cast member. Not all Xtra cast member have options available. The contents of the Options dialog box is determined by the developer of the Xtra. Refer to any documentation supplied with the Xtra.

Cast Member Script command (Modify menu)

Choose this command to open the script attached to the selected cast member. This is the same as clicking the Script button in the Cast Member Properties dialog box or in the cast window.

Shortcuts: To open the cast member script:

- Press Control-' (Control-hyphen).
- In the Cast window, select a cast member and click the Script button.
- In any media editor window, click the window's Script button.

Sprite > Properties command (Modify menu)

Use the Sprite Properties command to change the size, location, and blend percentage of a selected sprite. If you select more than one sprite, this command lets you edit them as a group.

Click a dialog box option for more information:

{button See also,AL(`Sprite_Info_help')}

Size changes the sprite's size. Enter an exact size or a scaling percentage.

- **Height, Width**: Enter a specific width and height for the sprite, in pixels.
- **Scale**: Enter a percentage in the Scale field. The sprite is scaled relative to its current size, not to the size of its parent cast member.
- **Maintain Proportions**: Turn on this option to maintain the same proportions of width and height when the object is resized.

Location changes the top left corner position of the sprite on the Stage.

- **Left:** Enter the number of pixels you wish to offset the sprite from the left edge of the Stage.
- **Top:** Enter the number of pixels you wish to offset the sprite from the top edge of the Stage.

Blend specifies the blend percentage for selected sprites in the Score. You can apply a blend effect to sprites that use Blend, Background Transparent, Mask, or Matte inks. Each sprite in the same frame can store its own blend value.

Use the Blend option in the In Between Special dialog box to fade sprites in or out of the Stage as they are animating.

Sprite > Script command (Modify menu)

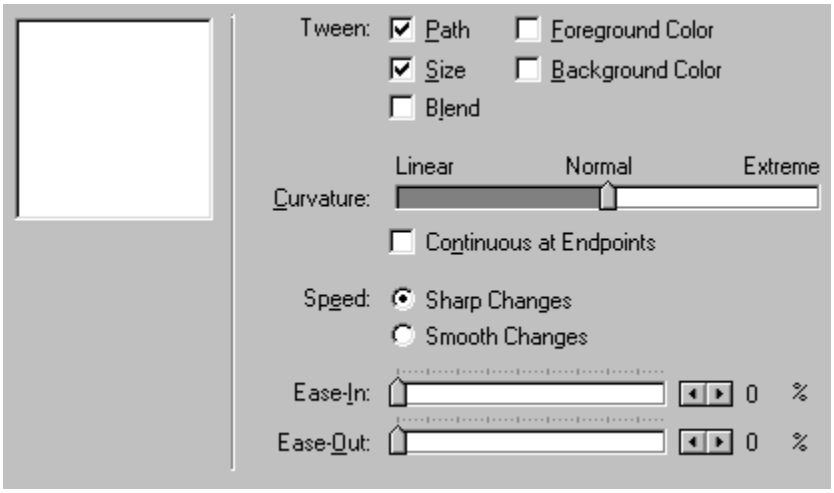
Sprite Script opens a Script Window for the currently selected sprite.

{button See also,AL(`Sprite_Script_help')}

Sprite > Tweening command (Modify menu) Control-Shift-B

This command opens the Sprite Tweening dialog box. Set the tweening properties to change the curve a sprite, to accelerate or decelerate across the Stage, or to change color or blend.

Click a dialog box option for more information:



{button See also,AL(`Sprite_Tween_Properties_help')}

Tween identifies the sprite properties to be tweened: Path, Size, Blend, Foreground Color, and/or Background Color.

Curvature controls the degree to which the sprite's curved path follows the inside or outside boundaries of the path.

- **Linear**: The sprite travels in straight lines between the sprite keyframe positions.
- **Normal**: The sprite follows a curved path that is inside the sprite keyframe positions.
- **Extreme**: The sprite follows a curved path outside keyframe positions.

Continuous at Endpoints makes the sprite move smoothly through start and end frames when moving in a circle.

Path diagram 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 beginning and ending points of the sprite are the same, the diagram is circular, indicating that the sprite travels in a circle when tweened. If the beginning and ending points are not the same, the diagram describes a curved path, indicating that the sprite ends in a position different than the starting point.

Speed defines how the tweened sprite properties change between keyframes.

- **Sharp Changes:** Changes in properties occur abruptly.
- **Smooth Changes:** Changes in properties occur gradually.

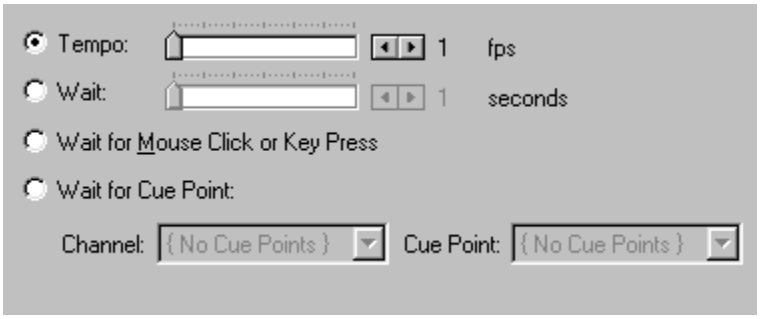
Ease-In and **Ease-Out** defines how tweened sprite properties change over the whole length of the sprite.

- **Ease-In:** Defines the percentage of the sprite span over which you want to accelerate the sprite.
- **Ease-Out:** Defines the percentage of the sprite span over which you want to decelerate the sprite.

Frame > Tempo command (Modify menu)

Use this command to set a tempo or pause in your movie. The tempo setting determines how the playback head moves from frame to frame.

Click a dialog box option for more information:



When you set a tempo, it also applies to all frames to the right of the tempo setting until another tempo is encountered in the tempo channel.

Shortcuts: To open the Frame Properties: Tempo dialog box:

- Double-click a cell in the tempo channel.
- Right-click a cell in the tempo channel and choose Tempo.

Tip: If you place tempo settings in the same frame as a transition, some tempo channel settings such as Wait become disabled. To avoid this, don't place a transition in the same frame as your tempo settings. Instead, place the tempo settings in the frame immediately before or after the transition.

{button See also,AL(`Set_Tempo_help')}

Tempo sets a new tempo for the movie. Use the arrows or slide the box to change the settings. This is the same as changing the tempo in the **Control Panel**.

Wait stops the movie at the current frame for the time specified. Use the arrows or slide the box to change the settings.

Wait for Mouse Click or Key Press pauses the playback head until the user clicks the mouse or presses a key. The cursor changes to a blinking mouse to indicate that the movie is paused.

Wait For Cue Point pauses the playback head at the designated cue point until the sound or digital video specified in the designated channel finishes playing. This option allows waiting for any cue point in digital audio or video media, not just the end point.

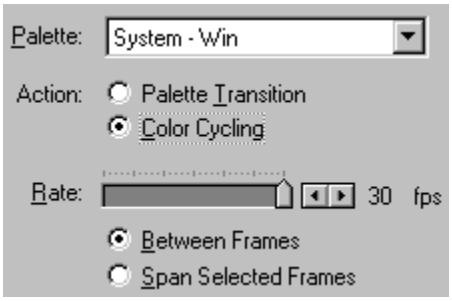
- **Channel** Select Sound 1, Sound 2, or any active sprite channel with media that supports cue points. You can type into the text field or choose from the list.
- **Cue Point** Select Next, End, or a cue point from the list of cue-point names or numbers for the element selected in Channel. You can also type into the field or choose from the list.

Frame > Palette command (Modify menu)

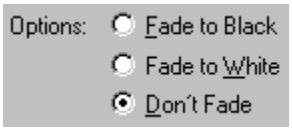
Use the Frame Palette command to change palettes in a selected frame of the palette channel. When you set a palette in the palette channel of the Score, the colors of the cast members in the movie are determined by that palette until another palette is encountered in the palette channel.

When you specify a new palette in the palette channel, cast members change color depending on the position of their colors in the palette. For example, if one cast member is yellow, and yellow occupies the fifth color in the palette, the cast member will change to whatever color is in the fifth position in the new palette.

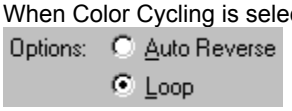
Click a dialog box option for more information:



The options in the dialog box change depending on whether you choose Palette Transition or Color Cycling.
When Palette Transition is selected, these options appear in the dialog box (click for more information):



When Color Cycling is selected, these options appear in the dialog box (click for more information):



Shortcuts: To open the Frame Properties: Palette dialog box:

- Double-click a frame in the palette channel.
- Right-click a frame in the palette channel and choose Palette.

{button See also,AL(`Set_Palette_help')}

Palette lets you choose the palette used for the selected cells in the palette channel.

Palette Transition changes the palette at the current frame, or the beginning of the current selected range. Choose Palette Transition when you want a smooth transition from one palette to another. Instead of your cast member abruptly changing colors when you switch palettes, this option gradually blends from one palette to the next.

Color Cycling changes the palette by rotating the colors in a selected range of the palette. For example, if a cast member's color is the fifth color in the palette, and you select a range of colors from four to six, the cast member changes colors when the movie is played, cycling through colors four, five, and six.

To select a range to cycle, drag across the colors to be cycled in the dialog box. You can also click a color and Shift-click another color to select those colors and all the colors in between.

Rate displays the rate at which the palette changes between frames. Use the arrow keys or slide the box to change the setting. This setting does not apply if you select Span Selected Frames.

- **Between Frames** changes the palette between frames. The Rate setting determines the length of the transition. All animated movement will halt as the transition takes place. This option appears only if you first select a range of frames in the palette channel.
- **Span Selected Frames** changes the palette while the selected frames are playing. The number of frames selected determines the length of the transition. Any movement in those frames will take place as usual. This option appears only if you first select a range of frames in the palette channel.

Palette Transition options

- **Fade to Black** fades the entire screen to black. Like other palette transitions, this can occur over time or between frames. A nice way to end a movie is to make the final frame a black cast member that covers the whole Stage, and then fade to black over several frames before the last frame.
- **Fade to White** fades the entire screen to white. Like other palette transitions, this can occur over time or between frames. A nice way to end a movie is to make the final frame a white cast member that covers the whole Stage, and then fade to white over several frames before the last frame.
- **Don't Fade** changes the palette without fading the screen during the palette transition.

Note: Palette transitions including Fade to Black and Fade to White do not work in 16-, 24-, and 32-bit environments. These features require either that palettes be in use or that a video card set to 256 colors is present.

Color Cycling options

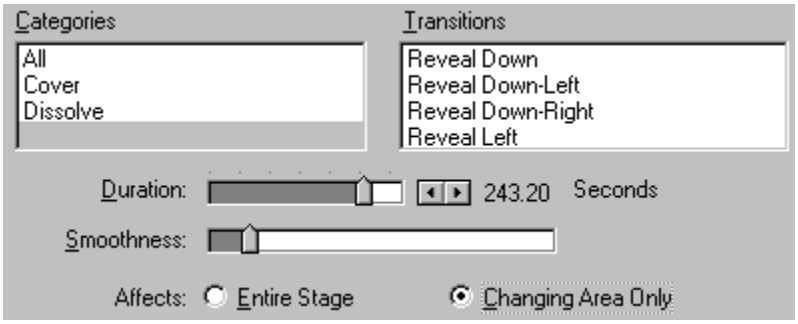
- **Cycles** specifies the number of cycles per frame.
- **Auto Reverse** reverses the direction of color cycling when the cycle completes.
- **Loop** returns the color cycle to the beginning when it reaches the end.

Frame > Transition command (Modify menu)

Use this command to define and select a transition. To set a transition, select a cell in the transition channel of the Score window and choose Frame Transition from the Modify menu. The transition occurs when the playback head reaches that cell.

Different options are available for different transitions.

Click a dialog box option for more information:



Shortcuts: To open the Frame Properties: Transition dialog box:

- Double-click a cell in the transition channel.
- Right-click a cell in the transition channel and choose Transition.

{button See also,AL(`Set_Transition_help')}

Categories lists the categories of transitions. If you select a category, such as Dissolve, then only the Dissolve transitions are listed in the categories list. If you select the All category, then all available transitions are listed.

Transitions lists the available transitions.

Duration indicates the approximate amount of time (in seconds) of the entire transition. Adjust the slider to change the setting.

Smoothness selects the smoothness of the transition. Adjust the slider to change the degree of smoothness.

Affects indicates where the transition takes place on the Stage.

- **Entire Stage** makes the transition take place over the entire Stage area.
- **Changing Area Only** makes the transition takes place over the changing area of the frame.

Frame > Sound command (Modify menu)

Use this command to select and preview sounds. Select one or more cells in one of the sound channels, and then choose this command. If you have not made a selection in one of the sound channels in the Score, this command is not available.

The dialog box lists all sounds in the currently opened casts. Select a sound from the list and click OK to place the sound in the selected frames.

To play a sound, select one from the list and click Play.

Shortcuts: To open the Frame Properties: Sound dialog box:

- Double-click a frame in the sound channel.
- Right-click a frame in the sound channel and choose Sound.

{button See also,AL(`Set_Sound_help')}

Frame > Script command (Modify menu)

Choose this command to open the script for the current frame.

Shortcut: To open the script for the current frame:

- Double-click a frame in the script channel.
- Right-click a frame in the script channel and choose Frame Script from the shortcut menu.

Movie > Properties command (Modify menu) Control-Shift-D

The Movie > Properties command lets you specify options such as Stage size and color for the currently open movie.

{button See also,AL(`Movie_Info_help')}

Dialog box options

Stage Size defines the size of the Stage. Changing the Stage size is useful if you want to display movies on a smaller or larger Stage, or if you want to change the Stage size to match the size of a digital video. Change the size of the Stage by choosing a setting from the menu, or by entering the width and height of the Stage.

If you choose a setting from the pop-up, the values in the Width and Height fields automatically update.

Stage Location changes the location of the Stage.

- **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. You can also use this option if you are creating a movie on a larger screen that will be seen on smaller screens.
- **Upper Left** places the Stage in the top left corner. Alternatively, the values you type in the Left and Top boxes represent the number of pixels the Stage is moved from the top left corner of the screen. These values apply only if the Stage is smaller than the current monitor's screen size.

Default Palette defines the palette Director uses for the movie until it encounters a different palette setting in the palette channel.

Stage Color determines the color of the Stage. Click to select a new background color from the current palette.

Remap Palettes When Needed remaps the current palette when cast members with different palettes appear on the Stage. If this box is checked, Director automatically creates a common palette and remaps all images on the Stage that have a different palette to the common palette. The cast members themselves are not modified. The common palette determines how the cast member is remapped. For example, if a cast member uses a grayscale palette, it will be drawn on the Stage using whatever grays are available in the common palette.

Allow Outdated Lingo lets you include Lingo commands used by Director 4.0 that are no longer acceptable.

Save Font Map saves the current font map settings in a text file named Fontmap.txt.

Load Font Map loads the font mapping assignments specified in the selected font map file.

Movie > Casts command (Modify menu) Control-Shift-C

Use Movie Casts to view the casts in the current movie, create new casts, and link external casts to the movie. The Movie Casts dialog box displays a list of all the casts in the current movie, including internal casts and linked external casts.

Dialog box options

New opens the **New Cast** dialog box so that you can create a new cast.

Link attaches existing external casts to the movie. Use the dialog box that appears to select an external cast file to link to the movie.

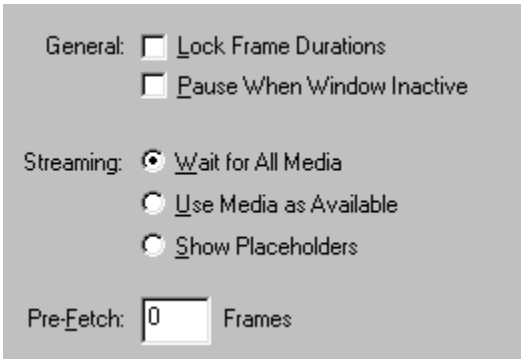
Remove unlinks or deletes a cast from the movie. Select a cast from the list and click Remove. If the cast is internal, it is deleted. If it is external, it is unlinked from the movie. You cannot delete the first internal cast in the movie.

Properties opens the **Cast Properties** dialog box for the cast selected on the list.

{button See also,AL(`Movie_Casts_help')}

Movie > Playback command (Modify menu)

This command opens the Movie Playback Properties dialog box. Use this box to determine how an Internet-based movie is played on your local system.



Dialog box options

General:

- **Lock Frame Durations** locks the current playback rate so that Director plays the movie at the same speed on all types of computers. For frames without recorded durations, Director uses the current tempo.

  Locked movies will not play faster when played on a faster computer, but may play slower on a slower computer.

- **Pause When Window Inactive** specifies when movies in windows play. When this option is set, a movie in a window only plays when the main movie is playing or when it is the frontmost window; otherwise the movie in a window continues to animate.

Streaming:

- **Wait for All Media** delays playback until the movie is completely downloaded from the internet.
- **Use Media as Available** begins playback as soon as the target frame is available, or the designated number of pre-fetch frames are available.
- **Show Placeholders** displays placeholders (rectangular outlines) on the screen until the required images are downloaded from the Internet.
- **Pre-Fetch _ Frames** Designates the number of frames to download from the Internet before playback of the movie can begin. If 0 frames are designated, playback will begin as soon as the target frame is available.

{button See also,AL(`Movie_Playback_help')}

Movie > Xtras command (Modify menu)

This command opens the Movie Xtras dialog box. The dialog box lists Xtras required by the movie. It displays Xtras that control or import cast members that appear in the Score. Xtras used only in Lingo scripts may not appear on the list.

Xtras appearing on the list shown in the dialog box are automatically included when a projector is created, as long as the Check Movie for Xtras option in the Projector Options dialog box is turned on.

Add opens a dialog box for selecting Xtras. Any Xtra you choose is added to the list.

Remove removes the selected Xtra from the list.

{button See also,AL(`Movie_Xtras_help')}

Font command (Modify menu) Ctrl-Shift-T

Use the Font dialog box to specify all the formatting options for characters and lines of text. Not all options are available for fields.

To change character formatting, first select the text you want to change, and then choose Font from the Modify menu. If you select the cast member, all the text changes.

Dialog box options

The scrolling field on the left lists all the fonts installed in your system. Select a new font by clicking one on the list. Director may not be able to anti-alias certain fonts in your system (such as bitmap fonts and some variations of TrueType fonts). When you select a font that can't be anti-aliased, the message "This font cannot be anti-aliased" appears above the font preview pane.

Style: Bold, Italic, and Underline each apply the character attribute to the selected text.

Size specifies the font size, in points.

Spacing sets the total height for all lines in the paragraph in points. This option is not available for fields.

Kerning increases or decreases the amount of space (in points) between selected characters. This option is not available for fields.

Color determines the color of the selected text. Click to choose a new color from the current color palette.

{button See also,AL(`Font_Settings_help')}

Paragraph command (Modify menu) Alt-Control-Shift-T

Use the Paragraph dialog box to view and change paragraph formatting. The Paragraph command is only available for text cast members.

Dialog box options

Alignment determines how the selected paragraph is aligned with the text box. The choices are Left, Right, Center, and Justify. (The Justify setting aligns text to both the left and right margins.)

Margin:

- **Left and Right** defines the left and right margin settings of the text box. Click the arrows or enter a number to change the amount the paragraph is indented.
- **First Indent** controls the indent setting of the first line of text in the paragraph. Changing the setting is the same as moving the margin markers on the text ruler.

Spacing increases or decreases the amount of space before or after the current paragraph. Click the arrows or enter a value in points.

{button See also,AL(`Paragraph_Settings_help')}

Borders submenu (Modify menu)

The commands on the Borders submenu only apply to fields. You cannot use them to change text cast members.

Line adds a box around the field on the Stage. Choose the line thickness from the submenu.

Margin changes the distance between the edges of the field and the characters inside. Choose the margin width from the submenu.

Box Shadow adds a drop shadow to the text box. Choose the drop shadow width from the submenu.

Text Shadow adds a drop shadow to bitmapped text in the Paint window or field text. Choose the shadow width from the submenu. Drop shadow on text is a good way to ensure that your text will remain legible in color if you are planning to overlay text to videotape.

Split Sprite command (Modify menu)

This command breaks a single sprite into two adjacent sprites, with ending and beginning key frames directly abutting. If a single cell or key frame is selected, the split occurs to the immediate right. If two cells are selected, the split occurs between them and turns each into a key frame.

{button See also,AL(`Split_Sprite_help')}

Join Sprites command (Modify menu)

This command turns two or more sprites into a single sprite span by tweening the ending and beginning key frames of the selected sprites.

{button See also,AL(`Join_Sprites_help')}

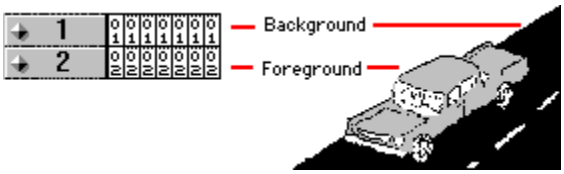
Extend Sprite command (Modify menu)

This command extends the end of each selected sprite to the current frame in a movie or to the position of the playback head.

{button See also,AL(`Extend_Sprite_help')}

Arrange submenu (Modify menu)

The Arrange submenu contains four commands that move sprites up or down in the Score, changing their order on the Stage. Sprites appear on the Stage in order, starting with the first channel. A sprite in channel two appears on top of a sprite in channel one.



Bring to Front (Control-Shift-Up arrow) moves the selected cells to the last channels in the Score. The sprites in those cells move in front of all other sprites.
Move Forward (Control-Up arrow) switches the selected cells with the cells immediately below. This command is the same as clicking the Move Forward button at the bottom of the Score window.

Move Backward (Control-Alt-Down arrow) switches the selected cells with the cells immediately above. This command is the same as clicking the Move backward command at the bottom the Score window.

Send to Back (Control-Alt-Shift-Down arrow) moves the selected cells to the first channels in the Score. The sprites in those cells move behind all other sprites.

{button See also,AL(`Arrange_help')}

Align command (Modify menu)Control-K

Use the Align palette to align sprites on the Stage. You can align sprites in multiple channels and frames.

Dialog box options

Horizontal alignment options include No Change, Align Tops, Align Centers, Align Bottoms, Align Reg. Points.

Vertical alignment options include No Change, Align Lefts, Align Centers, Align Rights, Align Reg. Points.

Tip: You can also click the preview to experiment with sprite alignment.

{button See also,AL(`Align_window_help')}

Tweak (Modify menu) Control-Shift-K

Use the Tweak window to move one or more selected sprites in any direction with precision. Drag the point on the left side of the window, or enter the number of pixels in the fields for horizontal and vertical change and click Tweak.

Continue clicking Tweak to repeatedly move the selected sprites the same distance.

Reverse Sequence command (Modify menu)

Reverses the order of selected cells in the Score.

Sort (Modify menu)

Use Sort to rearrange selected cast members in the cast and eliminate empty cast member positions. To rearrange an entire cast, first choose Select All from the Edit menu before choosing this command.

Director automatically updates the Score with the new number for each repositioned cast member.

Note: Because cast member numbers may change when you use this command, cast member number references in scripts may become invalid. If you use Sort Cast Members, you may have to go through your scripts to update them with the new numbers. Use cast member names instead of numbers to address cast members in a Lingo script so that you don't have to worry if your cast members get re-numbered.

Dialog box options

Usage in Score sorts selected cast members in the order in which they appear in the Score. If a cast member does not appear in the Score, it is placed after all the cast members that are referenced from the Score.

Media Type sorts selected cast members by type (bitmap, palette, button, text, sound, shape, PICT, digital video, film loop, movie, script, field, Xtra, transition, OLE).

Name sorts selected cast members alphabetically by name.

Size sorts selected cast members by file size, in decreasing size order.

Empty at End places empty cast members at the end.

{button See also,AL(`Sort_Cast_Members_help')}

Cast to Time command (Modify menu)

Use Cast to Time to speed the creation of a cast member sequence in your movie. This command places selected cast members sequentially into separate frames in the Score.

If you select a single cell in the Score before choosing this command, the selected cast members are added to the Score beginning at the selected cell. Any existing Score data is replaced by the Cast to Time sequence. If you set an insertion point in the Score before choosing this command, the Cast to Time sequence is inserted in channel 1, beginning at the insertion point. If you select a range of Score cells before choosing this command, the Cast to Time sequence that is inserted will only be as long as the number of selected cells in the Score.

Shortcut: You can also place selected cast members across time in the Score by Alt dragging from the cast.

{button See also,AL(`Cast_to_Time_help')}

Space to Time command (Modify menu)

Space to Time moves selected sprites in one frame to a single channel in the Score so they play in a sequence of frames.

The dialog box lets you specify the number of frames apart to spread sprites. Consecutive cells (1 frame apart) is the default.

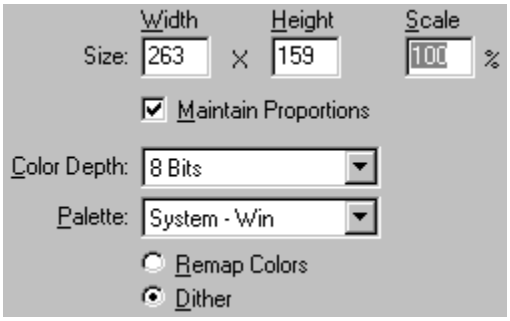
{button See also,AL(`Space_to_Time_help')}

Transform Bitmap command (Modify menu)

Transform Bitmap changes 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. As a result, color depth and palette changes can't be undone. If you want to keep a cast member's original bitmap unchanged but temporarily apply a different palette, use **Cast Member Properties** instead. To change the size of only the sprite on the Stage, use **Sprite Properties**.

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.

Click a dialog box option for more information:



{button See also,AL(`Transform_Bitmap_help')}

Size determines the dimensions of the selected cast member. If multiple cast members are selected, you can resize all the cast members to the dimensions you enter. You can either enter new measurements (in pixels) in the Width and Height fields, or enter a scaling percentage in the Scale box.

Turn on the **Maintain Proportions** check box 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.

Color Depth sets the color depth of the selected cast member. A cast member's color depth is determined at import by the selection set in the Import dialog box. A movie's color depth is determined by the cast member with the highest color depth.

You can change the color depth to save memory and disk space when you are creating a color movie.

Palette selects the palette for the selected cast member. The palettes listed in the pop-up are the default Director palettes, plus any additional ones found in the casts. You can create a common palette that contains most of the colors your cast member needs.

When you use the Transform Bitmap command to remap the color in a cast member, Director matches the colors of the cast member with similar colors in the new palette. For example, if the original artwork is red and the closest red available in the new palette is pink, the red is changed to pink.

If the movie is playing, the active palette is the one that is currently in use at any given time, as specified in the Score. The palette active when you use Transform Bitmap may be different from the palette used by the movie.

Remap Colors replaces the image's colors with the most similar solid colors in the palette you select from the pop-up.

Dither blends the colors in the new palette to approximate the original colors in the graphic.

Convert to Bitmap command (Modify menu)

This command converts fields to bitmapped cast members. The converted graphic can then be edited in the Paint window. Once you convert a cast member to a bitmapped graphic, you cannot undo the change.

You can't convert a shape to a bitmap.

Control menu

Click the name of a menu command for more information:

- Play
- Stop
- Rewind
- Step Forward
- Step Backward
- Real-Time Recording
- Step Recording
- Loop Playback
- Selected Frames Only
- Volume
- Disable Scripts
- Toggle Breakpoint
- Watch Expression
- Remove All Breakpoints
- Ignore Breakpoints
- Step Script
- Step Into Script
- Run Script
- Recompile All Scripts

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Play command (Control menu) Control-Alt-P

The Play command starts the movie. If you press the Shift key while choosing Play, the menu bar is hidden and the Stage is cleared of all open windows as the movie plays.

Shortcut: The keypad + key toggles between Play and Stop.

{button See also,AL(`Play_help')}

Stop command (Control menu) Control-period (.)

The Stop command halts the movie.

Shortcut: The keypad + key toggles between Play and Stop.

{button See also,AL(`Stop_help')}

Rewind command (Control menu) Control-Alt-R

Rewind moves the playback head back to frame 1. If the animation is playing, it also stops.

Shortcut: Keypad 0 rewinds the movie.

{button See also,AL(`Rewind_help')}

Step Forward command (Control menu)

The Step Forward command advances the movie forward one frame. When using the step recording technique, it can be used to advance to the next frame to record sprites.

Shortcut: Keypad 3 or Control-right arrow steps the movie forward.

{button See also,AL(`Step_Forward_help')}

Step Backward command (Control menu)

Step Backward steps the movie backward one frame at a time.

Shortcut: Keypad 1 or Control-left arrow steps the movie backward.

{button See also,AL(`Step_Backward_help')}

Real-Time Recording command (View menu)

Real-time recording is an animation technique. When you select this option you can create animation by recording the movement of a sprite as you drag it across the Stage.

{button See also,AL(`Realtime_Recording')}

Step Recording command (View menu)

Step recording is an animation technique. When you select this option, you can create animation one frame at a time. Position and record the sprite in the starting frame, step forward one frame, position and record the sprite in this frame, and then step forward again. Repeat this process until the animation sequence is complete.

{button See also,AL(`Step_Recording')}

Loop Playback command (Control menu) Control-Alt-L

If selected, the Loop Playback command causes the movie to repeat continuously when played. When the movie reaches the last frame, it automatically starts again from frame 1. By default, this option is on.

Shortcut: Keypad 8 causes the movie to loop.

{button See also,AL(`Loop_help')}

Selected Frames Only command (Control menu)

Selected Frames Only designates a range of frames that can be played. This is convenient if you are working on just one part of a movie.

To play a portion of a movie, open the Score and select the frames to be played. Choose Selected Frames Only, make sure loop is turned on, and play the movie.

When a portion of the Score has been marked as selected frames, a green bar appears at the top of the Score over the selected frames.

Turn off Selected Frames Only when you want to return to normal play mode.

This command is dimmed if no frames are selected in the Score.

{button See also,AL(`Selected_Frames_Only_help')}

Volume submenu (Control menu)

The Volume submenu specifies the sound level of the movie. Choose a level from 0 (mute) to 7 (loud).

Shortcut: Keypad 7 toggles between sound on and sound off.

{button See also,AL(`Volume_help')}

Disable Scripts command (Control menu)

This command is useful when you want to control whether interactivity is on or off during playback, or if you want to preview exporting a range of frames as a digital video.

Toggle Breakpoint command (Control menu) Control-Shift-Alt-K

This command inserts and removes breakpoints for the line of Lingo that the Script Window cursor is in. Director opens the Debugger window whenever it encounters a breakpoint in scripts.

When a line in a script has a breakpoint, the Toggle Breakpoint command removes the breakpoint. When there is no breakpoint, the command inserts one.

{button See also,AL(`Toggle_Breakpoint_help')}

Watch Expression command (Control menu) Control-Shift-Alt-W

The Watch Expression command adds to the **Watcher window** any expressions and variables in the line of Lingo that the Script window's text cursor is currently in. This has the same effect as clicking the Watch Expression button in the **Debugger window**.

Remove All Breakpoints command (Control menu)

This command removes all breakpoints from the movie's scripts.

{button See also,AL(`Remove_All_Breakpoints_help')}

Ignore Breakpoints command (Control menu)Control-Shift-Alt-I

This command has Lingo ignore any breakpoints in the movie's scripts as the movie plays.

{button See also,AL(`Ignore_Breakpoints_help')}

Step Script command (Control menu) Control-Shift-Alt-down arrow

The Step Script command runs the current line of Lingo but doesn't run any nested handlers that the line calls.

{button See also,AL(`Step_Script_help')}

Step Into Script command (Control menu) Control-Shift-Alt-right arrow

The Step Into Script command runs the current line of Lingo and follows Lingo's normal flow through any handlers called by that line.

{button See also,AL(`Step_Into_Script_help')}

Run Script command (Control menu) Control-Shift-Alt-Up arrow

The Run Script command is only available when Lingo has stopped at a breakpoint. Use this command to restart Lingo.

{button See also,AL(`Run_Script_help')}

Recompile All Scripts command (Control menu) Control-Shift-Alt-C

This command recompiles all scripts and checks them for errors. If a script error is found, the appropriate Script Window opens and the location of the error is selected.

If a Script Window is the active window, Director first saves any changes in the current Script window and compiles its script before continuing.

Xtras menu

Click a command or submenu for more information:

Update Movies

Filter Bitmap

Auto Filter

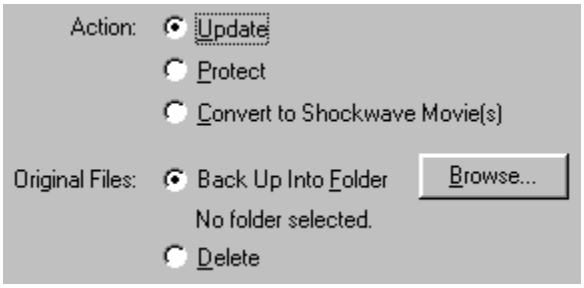
Auto Distort

Update Movies command (Xtras menu)

Use the Update Movies command on the Xtras menu to:

- Update movies from Director 4.x movies to the latest file format
- Creating several Shockwave movies at once
- Remove redundant and fragmented data in movie and cast files
- Prevent users from opening movie and cast files
- Batch-process movie and cast files in large projects

Click a dialog box option for more information:



{button See also,AL(`Update_Movies_help')}

Update converts movies from Director 4 or later to the latest file format. As it updates movies, Director consolidates and removes fragmented data. You can also use this option to rewrite files from the current version of Director. This removes redundant and fragmented data in the same way as Save As does. (To update movies from older versions, you must first convert them to the Director 4 file format.)

Protect makes a movie or cast uneditable. It prevents users from opening the movie or cast and making changes. Protect compacts the movie in the same way as Update and Compact, but it makes the movie even smaller by removing lingo script text and thumbnails. Once a movie is protected, there is no way to "unprotect" it, so be sure to keep an unprotected copy.

Convert to Shockwave Movie(s) rewrites movies in the compressed Shockwave file format and adds the DCR extension. This options also prevents users from opening the movie or cast and making changes. Once a movie is compressed, there is no way to "decompress" it, so be sure to keep the original movie.

Back Up Into Folder specifies that the original files should be placed 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 original files should be overwritten by the newly updated files. Be very careful using this option, especially if you are protecting files. Once a file is protected, you cannot open it in Director.

Filter Bitmap command (Xtras menu)

The Filter Bitmap command displays all the filters you have installed as Xtras. Filters are plug-in image editors that apply effects to bitmapped images. You can also install Adobe Photoshop and Premiere.

Filter Bitmap Dialog options

Categories displays the categories of available filters. These categories are defined by the filters themselves. When you select a category, the filters in that category appear in the Filters list to the right. Choose All to view filters in all categories.

Filters displays all the filters in the current category.

Filter activates the current filter and opens the filter controls.

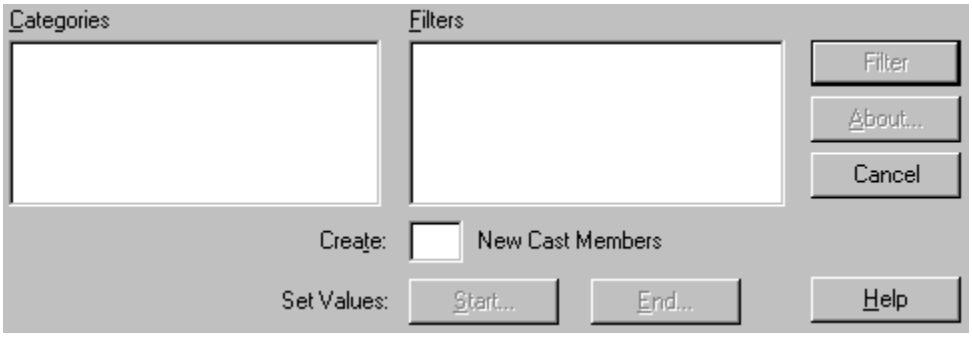
{button See also,AL(`Filter_Bitmap_submenu_help')}

Auto Filter command (Xtras menu)

Use Auto Filter to create dramatic animated effects with filters. Auto Filter applies a filter incrementally to a series of cast members. You can use it to change a range of selected cast members or to generate a series of new filtered cast members based on a single cast member. You define a beginning and ending setting for the filter, and Auto Filter applies an intermediate filter value to each cast member.

For example, if a filter converts an image to look as if it is breaking apart like broken glass, you can apply it to a cast member with Auto Filter and create a series of ten cast members. The first would show the pieces just coming apart and the last would show the pieces completely fragmented. You could then show the image breaking apart in an animation.

Click a dialog box option for more information:



Auto Filter generates new cast members and places them in empty 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.

{button See also,AL(`Xtra_Auto_Filter_help')}

Categories displays the categories of available filters. These categories are defined by the filters themselves. When you select a category, the filters in that category appear in the Filters list to the right. Choose All to view filters in all categories.

Filters displays all the filters in the current category.

Set Values:

- **Start** defines the filters settings of the first cast member to be filtered.
- **End** defines the filter settings for the last cast member to be filtered.

Create _ New Cast Members defines the number of new cast members that will be created. This option is not available if you select a range of cast members.

Filter activates the current filter and opens the filter controls.

Auto Distort command (Xtras menu)

The Auto Distort command automatically generates tween positions for any cast member that is free rotated, made into a perspective, slanted, distorted, or stretched. After artwork has been altered with one of these five effects, and before you deselect the artwork, choose Auto Distort, and enter the number of tween cast members in the Generate New Cast Members field in the Auto Distort dialog box. The new cast members are placed in the next available cast member positions.

{button See also,AL(`Xtra_Auto_Distort_help')}

Help menu

The Help menu contains commands for launching various sections of the Director Help file. It also lets you register your copy of Director.

Help Pointer

Select the Help Pointer command to change the cursor to the Help Pointer ("?"). You can also select the Help Pointer with the Help Pointer tool at the far right of the toolbar.

When you click a Director interface element (such as a menu command or a window) with the Help Pointer, Director Help appears with information about that interface element.

Keyboard shortcuts

Click a category for more information:

Shortcut menus

Menu command shortcuts:

File menu shortcuts

Edit menu shortcuts

View menu shortcuts

Insert menu shortcuts

Modify menu shortcuts

Control menu shortcuts

Window menu shortcuts

Director window shortcuts:

Score shortcuts

Stage shortcuts

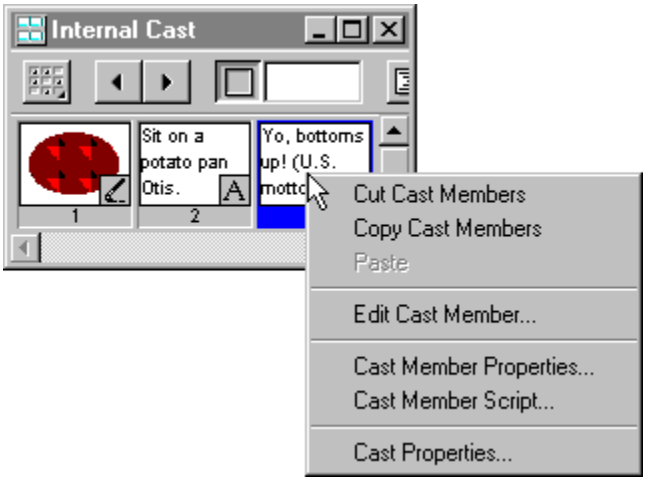
Cast window & cast editor shortcuts

Paint window shortcuts

Shortcut menus

Director supports shortcut menus throughout the user interface.

To display a shortcut menu, right-click a sprite or cast member, or in any window. A menu of commonly used commands is displayed.



File menu shortcuts

Command	Shortcut
New Movie	Control-N
New Cast	Control-Alt-N
Open	Control-O
Close	Control-F4
Save	Control-S
Import	Control-R
Export	Control-Shift-R
Page Setup	Control-Shift-P
Print	Control-P
General Preferences	Control-U
Exit	Alt-F4

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Edit menu shortcuts

Command	Shortcut
Undo	Control-Z
Repeat	Control-Y
Cut	Control-X
Copy	Control-C
Paste	Control-V
Clear	Delete
Duplicate	Control-D
Select All	Control-A
Find Text	Control-F
Find Handler	Control-Shift-;
Find Cast Member	Control-;
Find Selection	Control-H
Find Again	Control-Alt-F
Replace Again	Control-Alt-E
Edit Sprite Frames	Control-Alt-]
Edit Entire Sprite	Control-Alt-[
Exchange Cast Members	Control-E
Launch External Editor	Control-, (comma)

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

View menu shortcuts

Command	Shortcut
Next Marker	Control-right arrow
Previous Marker	Control-left arrow
Zoom In	Control-+
Zoom Out	Control-- (minus)

Show Grid	Command-Shift-Alt-G
Snap to Grid	Control-Alt-G
Rulers	Control-Shift-Alt-R
Show Info	Control-Shift-Alt-O
Show Paths	Control-Shift-Alt-H
Toolbar for current window	Control-Shift-H
Keyframes	Control-Alt-Shift-K

Insert menu shortcuts

Command	Shortcut
Keyframe	Control-Alt-K
Frames	Control-Shift-]
Remove Frame	Control-[

Modify menu shortcuts

Command	Shortcut
Cast Member Properties	Control-I
Cast Member Script	Control-' (apostrophe)
Sprite Properties	Control-Shift-I
Sprite Script	Control-Shift-' (apostrophe)
Sprite Tweening	Control-Shift-B
Movie Properties	Control-Shift-D
Movie Casts	Control-Shift-C
Font	Ctrl-Shift-T
Paragraph	Control-Shift-Alt-T
Join Sprites	Control-J
Split Sprite	Control-Shift-J
Extend Sprite	Control-B
Bring to Front	Control-Shift-Up arrow
Move Forward	Control-Up arrow
Move Backward	Control-Alt-Down arrow
Send to Back	Control-Alt-Shift-Down arrow
Align	Control-K
Tweak	Control-Shift-K

Control menu shortcuts

Command	Shortcut
Play	Control-Alt-P
Stop	Control-period (.)
Rewind	Control-Alt-R
Step Backward	Control-Option-left arrow

Step Forward	Control-Option-right arrow
Loop Playback	Control-Alt-L
Volume: Mute	Control-Alt-M
Toggle Breakpoint	F9
Watch Expression	Shift-F9
Ignore Breakpoints	Alt-F9
Step Script	F10
Step Into Script	F8
Run Script	F5
Recompile All Scripts	Control-Shift-Alt-C

Window menu shortcuts

Command	Shortcut
Toolbar	Control-Shift-Alt-B
Tool Palette	Control-7
Behavior Inspector	Control-Alt-;
Sprite Inspector	Control-Alt-S
Text Inspector	Control-T
Stage	Control-1
Control Panel	Control-2
Markers	Control-Shift-M
Score	Control-4
Cast	Control-3
Paint	Control-5
Text	Control-6
Field	Control-8
Color Palettes	Control-Alt-7
Video	Control-9
Script	Control-0
Message	Control-M
Debugger	Control-`(back single quote)
Watcher	Control-Shift-`(back single quote)

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Stage shortcuts

Action	Shortcut
Select only the current frame of the sprite	Alt-click the sprite
Show/hide sprite paths	Control-Shift-AltH
Create a keyframe within a sprite path	Alt-click a tick mark in the sprite path
Show/hide sprite information panels	Control-Shift-AltO

Change the opacity of sprite information panels	Drag the horizontal line on the right side of a panel.
Open cast member editor	Double-click sprite
Open paint window	Control-5
Inks pop-up	Control-click
Real-time record	Control-Spacebar-drag a sprite on the stage
Display shortcut menu for selection	right-click
Hide selection indicators	Keypad + (plus)
Move sprite by one pixel	Arrow keys
Move sprite by ten pixels	Shift-arrow keys

Score window shortcuts

Selecting sprites and frames

Action	Shortcut
Duplicate selection (sprite or keyframe)	Alt
Select a frame within a sprite	Alt-click a frame within sprite
Turn Edit Sprite Frames on or off	Alt-double-click a frame within sprite
Select empty frames and sprite frames	Alt-drag beginning in an empty frame
Select all the frames in a channel	Double-click channel number, drag to select mulitple
Select all the sprite in a channel	Click the channel number, drag to select multiple

Moving and stretching sprites

Action	Shortcut
Shuffle backward	Control-up arrow
Shuffle forward	Control-down arrow
Overwrite sprite frames while dragging a selection to a new location	Press Control while dragging
Move sprite on the Stage by one pixel	Select in the Score and use arrow keys
Move sprite on the Stage by ten pixels	Select in the Score and use Shift-arrow keys
Move entire sprite (instead of keyframe)	Spacebar-drag
Stretch a sprite without proportionally relocating keyframes	Control-drag the end frame

Moving the playback head

Action	Shortcut
Move playback head to end of movie	Tab
Move playback head to beginning of movie	Shift-Tab
Move playback head to beg/end	Control-Shift-left/right arrow
Go to next marker comment (or jump 10 frames)	Control-right arrow
Previous marker comment (or back 10 frames)	Control-left arrow

Opening editors

Action	Shortcut
Open cast editor for selected sprite	Double-click a sprite frame or the cast thumbnail
Open frame settings dialog box	Double-click tempo, palette, or transition channel

Changing the Score

Action	Shortcut
Open shortcut menu for Score display options and preferences	Right-click in the channel number area of the Score

Cast window & cast editor window shortcuts

Action	Shortcut
Open cast member editor	Double-click a paint, text, palette or script cast member or select the cast member and press Return
Cast member script	Control-' (apostrophe)
Switch selected cast member with score selection	Alt-double-click thumbnail
Display cast member info	Control-click cast thumbnail
Open script in new window	Alt-Script button
Place button	Control-Shift-L (places selected cast member in center of stage)
Cast to Time (Option-Place button)	Control-Shift-Alt-L
Create a new cast member*	Control-Shift-A
Previous cast member*	Control-left arrow
Next cast member*	Control-right arrow
* same function, in a new window	Control-Alt-left/right arrow
Scroll up/down one window	Page up, Page down
Scroll to top left of cast window	Home
Scroll to show last occupied cast member	End
Type-select by cast member	Type number.

Paint window shortcuts

Changing tool settings

Action	Shortcut
Open Gradient Settings dialog box and set ink to gradient	Double-click paintbrush, rectangle, paint bucket, or polygon tool
Open Air Brush Settings dialog box	Double-click airbrush
Open Pattern Settings dialog box	Double-click pattern chip
Open Brush Settings dialog box	Double-click paintbrush
Open Paint Window Preferences	Double-click line width selector
Turn selected tool into foreground eyedropper	D key, while pressed
Turn selected tool into background eyedropper	Shift-D key
Turn selected tool into hand tool	Spacebar, while pressed
Turn selected tool into destination eyedropper	Alt-D key

General

Action	Shortcut
Undo	~ (tilde)
Next/previous cast member	Keypad left/right arrow keys
Open Transform Bitmap dialog box	Double-click color resolution indicator
Toggle Zoom in/Zoom out	Control-click in window or double-click pencil tool

Working with images

Action	Shortcut
Nudge selection rect. or lasso selection	Keypad arrows with selection rectangle or lasso
Change airbrush size (while painting)	Keypad up/down arrows with airbrush selected
Change airbrush flow (while painting)	Keypad left/right arrows with airbrush selected
Change foreground color (not painting)	Keypad up/down arrows, all tools
Change background color (not painting)	Shift-keypad up/down arrows, all tools
Change destination color (not painting)	Alt-keypad up/down arrows, all tools
Draw border w/current pattern	Alt-shape or line tools
Select background color	Shift-eyedropper
Select destination color	Alt-eyedropper
Toggle between custom and grayscale patterns	Alt-click pattern
Polygon lasso	Alt-lasso
Duplicate selection	Alt-drag
Stretch	Control-drag
Draw with background color	Alt-pencil tool
Clear visible part of window	Double-click eraser
Open color palettes window	Double-click foreground, background, or destination color chip

Text, Field, and Script window shortcuts

Action	Shortcut
Bold	Control-Alt-B
Italic	Control-Alt-I
Underline	Control-Alt-U

Sprite shortcuts

Changing sprite duration

Action	Shortcut
Stretch a sprite without proportionally relocating keyframes	Control-drag the end frame
Join Sprites	Control-J
Split Sprite	Control-Shift-J
Extend Sprite command	Control-B

Tweening sprites

Sprite Tweening command	Control-Shift-B

Selecting and moving sprites

Select a single frame within a sprite	Alt-click the sprite
Select a single frame within a sprite	Alt-click the sprite on the Stage, or a frame within the sprite in the Score
Move entire sprite between frames (instead of keyframe)	Spacebar-drag
Select all the sprite in a channel	Click the channel number, drag to select

	multiple
Select all the frames in a channel	Double-click channel number, drag to select mulitple
Turn Edit Sprite Frames on or off	Alt-double-click a frame within sprite
Select empty frames and sprite frames	Alt-drag beginning in an empty frame
Overwrite sprite frames while dragging a selection to a new location	Press Control while dragging

Changing sprites on the Stage

Move sprite on the Stage by one pixel	Select on the Stage or in the Score and use arrow keys
Move sprite on the Stage by ten pixels	Select on the Stage or in the Score and use Shift-arrow keys
Show/hide sprite paths	Control-Shift-Alt-H
Create a keyframe within a sprite path	Alt-click a tick mark in the sprite path on the Stage

Windows

For information on a window, click its name:

- Behavior Inspector
- Cast
- Color Palettes
- Control Panel
- Debugger
- Field
- Markers
- Memory Inspector
- Message
- Paint
- Score
- Sprite Inspector
- Text Inspector
- Text
- Toolbar
- Tool Palette
- Video
- Script
- Watcher

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Window menu

The commands in the Window menu open and close Director's authoring windows. Open windows have checkmarks next to their names.

Click the name of a menu command or submenu for more information:

- New Window
- Toolbar
- Tool Palette
- Inspectors > Behavior
- Inspectors > Sprite
- Inspectors > Text
- Inspectors > Memory
- Stage
- Control Panel
- Markers
- Score
- Cast
- Paint
- Text
- Field
- Color Palettes
- Video
- Script
- Message
- Debugger
- Watcher

Director automatically hides all open windows if you choose Stage from the Window menu.

New Window command (Window menu)

The New Window command duplicates the front-most window and its contents, creating another view. This command works with Score, Cast, Text, Field, Script, and Digital Video windows.

Duplicating a window is useful if the window's contents are large and you want to look at or edit different sections of the window simultaneously. It's especially useful for viewing several casts at once. Changes you make in the window are automatically reflected in all other views of the same window.

Shortcut: Press Alt while choosing a Text, Digital Video, Script, Cast, or Field window from the Window menu.

Toolbar (Window menu) Control-Shift-Alt-B

The buttons on the toolbar provide shortcuts for common commands and functions.

The Toolbar command on the Window menu shows or hides the toolbar underneath the menu bar. Click a tool for more information:

New Movie	Exchange Cast Members
New Cast	Extend Sprite
Open	Align
Save	Rewind
Print	Stop
Import	Play
Undo	Cast window
Cut	Score window
Copy	Paint window
Paste	Text window
Find Cast Member	Behavior Inspector
	Script window

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Tool palette (Window menu) Control-7

- Click a tool shown at the left for more information.
- Text, shapes, and buttons you create with tools appear as cast members in the Cast and Score windows.
- Shapes are QuickDraw graphics, not bitmaps.
- Shapes print better than bitmaps, but they animate more slowly.

{button See also,AL(`Tools')}

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

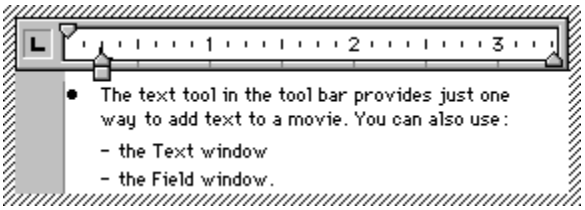
Selection (arrow) tool

The Selection tool is a standard selection arrow.

Text tool

The Text tool creates text cast members directly on the Stage. Click the Text tool and then drag to define the area on the Stage where you want text. When you release the mouse button, a text insertion point appears in the area you just defined and you can begin entering text. The new text cast member is placed in the first available position in the current cast. The sprite is placed in the first open score cell in the current frame.

Click the Arrow to select text that is already on the Stage. You can change the color of selected text using the Foreground and Background color chips in the Tool palette.



To edit the text cast member on Stage, click once to select and move the sprite or double-click to edit the text. Click and drag the handles to change the size of the text box. Work with the text formatting ruler and define tab settings in the text box by choosing **Ruler** from the View menu. When you make a change, Director updates all instances of the text cast member.

The Text tool is an alternative to the Text window for creating text. The **Text window** is faster and more convenient for working on substantial amounts of text.

You can use **Sprite Properties** in the Modify menu to change or determine the size, location, or blend of a selected sprite. Any changes to a sprite's properties only affects the sprite's appearance on the Stage and does not alter the actual properties of the cast member.

{button See also,AL(`Tools_Text')}

Line tool

Click the Line tool and drag it across the Stage to draw. You can choose the color for the line with the foreground and background pop-up that appears when you click the color chips. The width of the Line tool is controlled by the Line Width Selector at the bottom. The Line tool is constrained to horizontal, vertical, or 45-degree lines with the Shift key.

Shape tools

Click the Shape tools to draw an outline of the selected shape or on the left to draw shapes with a solid color or pattern.

Choose the color for the shape with the foreground and background palettes that appear when you press the color chips.

Use the Pattern chip to select the current pattern. Use the Line Width Selector at the bottom of the Tool palette to control the thickness of the borders of the Shape tools.

{button See also,AL(`Tools_Shape')}

Field tool

The Field tool creates field cast members directly on the Stage. Click the Field tool and then drag to define the area on the Stage where you want the field.

Use fields for creating user-editable text in movies or text that you want to format with Lingo. Use the Text tool to create all other types of text.

{button See also,AL(`Tools_Field')}

Button tools

Director provides three tools for creating buttons, checkboxes, and radio buttons. Click the Check box tool, Push button tool, or Radio button tool and drag a rectangle on the Stage to create the button. Then type the text that you want to appear on or next to the button. Set the font, style, and size. The button is placed in the cast as a button cast member. You can edit the button's text on the Stage or in a Text window.

Do not confuse these buttons with Custom buttons created by the Button Editor.

Buttons do not perform any special function until you write a Lingo script for them.

{button See also,AL(`Tools_Button')}

Foreground and Background color chips

The Foreground and Background color chips in the Tool palette set the color of text, shapes, and sprites.

To set the color for a sprite, select the sprite in the Score or on the Stage and choose a New Foreground Color using the Foreground color chip, or a New Background Color using the Background color chip.

To set the color of text, select the text you want to change and then choose a text color using the Foreground color chip. To set the text's background color, choose a color using the Background color chip.

If you apply color to a 1-bit cast member, Director changes the color of the sprite on the Stage but does not change the color of the actual cast member, which remains black and white.

Pattern settings

The Pattern selector lets you select the current pattern for a Shape tool. It also provides access to the **Tile Settings** and Pattern Settings dialog box.

Line width selector

Lets you select the current width of the Line tool, or the border for a shape tool.

Inspectors > Text (Window menu) Control-T

The Text Inspector is a floating window that provides tools to edit text cast members directly on the Stage. The tools are shortcuts for the formatting options in the **Paragraph** and **Font** dialog boxes.

Font pop-up is used to choose any font in your system.
Bold, Italic, and Underline buttons apply character formatting to selected text.
Size specifies the font size. Choose a size from the pop-up, or enter a size in the field. You may need to adjust the line spacing.

Line spacing displays the line spacing in points. Click the up and down arrows to change the setting, or enter a value in points.

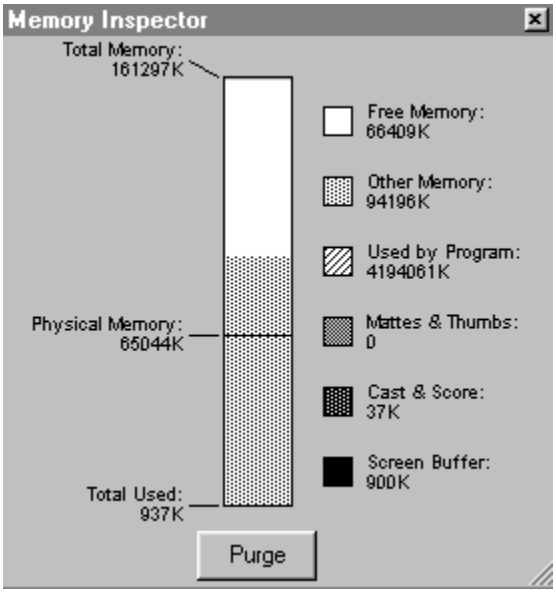
Alignment aligns selected paragraphs. Click Align Left, Align Center, Align Right, or Justify to align paragraphs.

Kerning sets the kerning between selected paragraphs. The value is in points.

{button See also,AL(`Text_Inspector_help')}

Inspectors > Memory (Window menu)

The Memory Inspector displays information about how much memory is available to Director for your movie. It also indicates how much memory different parts of the current movie use and the total disk space the movie occupies.



Total Memory displays the total system memory available. This number depends on the amount of RAM installed on your computer and any virtual memory that's available.
Partition Size indicates the memory limit assigned to Director in the Limit Memory Size to box of the General Preferences dialog box.

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 Windows, by DIRECTOR.EXE, and by other applications.

Used by Program indicates the amount of memory used by Director (excluding the amount of memory taken up by DIRECTOR.EXE).

Mattes & Thumbs shows how much memory is used by cast members that use the Matte ink in the Score and by thumbnail images in the Cast window.

Cast & 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, and any sounds, palettes, buttons, digital video movies, or linked files imported into the cast and currently loaded into memory.

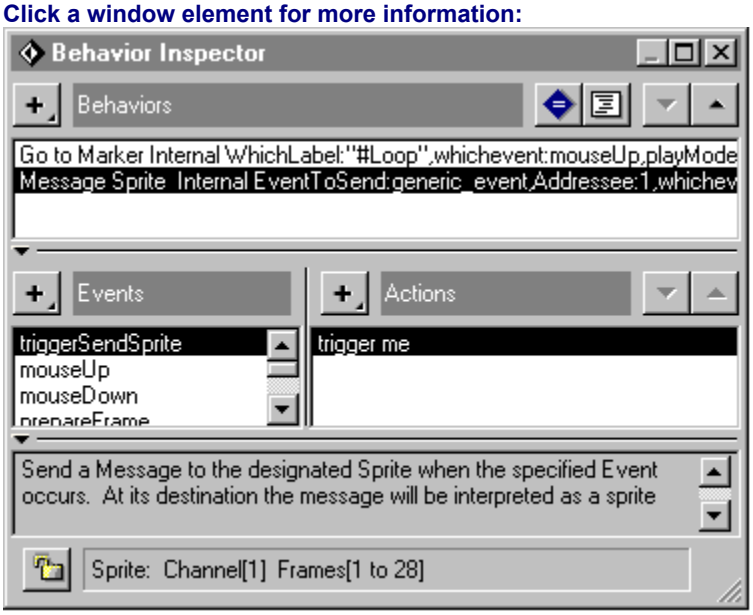
Screen Buffer shows how much memory Director reserves for a "working area" while animating on the Stage.

Purge button removes all purgeable items from RAM, including all thumbnail images in the Cast window. All cast members that have Unload (purge priority) set to a priority other than "0-Never" (as specified in the Cast Member Properties dialog box) are removed from memory. This is useful for gaining as much free memory as possible before importing a large file. Edited cast members don't get purged.

Inspectors > Behavior (Window menu) Control-Alt-;

The Behavior Inspector is a floating window that lists the behaviors attached to the current selection. The Behavior Inspector provides controls for editing existing behaviors or creating new ones. It also displays descriptions of the selected behavior.

Click a window element for more information:



{button See also,AL(`Behavior_Inspector_help')}

Behavior pop-up

Displays the behaviors currently available to the movie. Choose New Behavior to create a new behavior.

Note: You can attach pre-created behaviors to sprites. See the Xtras > Behavior Library menu. Once attached, the behaviors appear in the Behavior pop-up in the Behavior Inspector.

Parameters button

Opens the Parameters dialog box for the selected behavior.

Cast Member Script button

Opens the Script window and displays the script of the current behavior.

Behavior Shuffle Down/Up buttons

These buttons move the selected behavior down or up in the Behavior list. Director executes behaviors in the order they appear on the list.

Behavior list

Lists the behaviors attached to the current sprite. The cast in which the behavior is stored and the parameter settings appear after the name. Director executes behaviors in the order they appear on the list.

Edit pane expander

Shows or hides the Edit pane. Use the Edit pane to modify behaviors.

Event pop-up

Use the Event pop-up to append a new event to the list. Choose New Event from the pop-up to specify a message from a script or behavior as a custom event.

Action pop-up

Use the Action pop-up to append a new action to the Action list. Attach as many actions as you need to a single event. Choose New Action from the pop-up to enter the name of Lingo function or handler.

Action Shuffle Down/Up buttons

Moves the selected action down or up in the Action list. Director executes actions in the order the appear on the list.

Events list

Lists the events that the current behavior responds to.

Actions list

Lists the actions performed when the event occurs.

Description pane expander

Shows or hides the description pane for the selected behavior.

Behavior description

Describes the current behavior. Behaviors included with Director have descriptions. Others may not.

Lock Selection button

Locks the current selection so nothing changes in the Behavior Inspector when new sprites are selected.

Selection

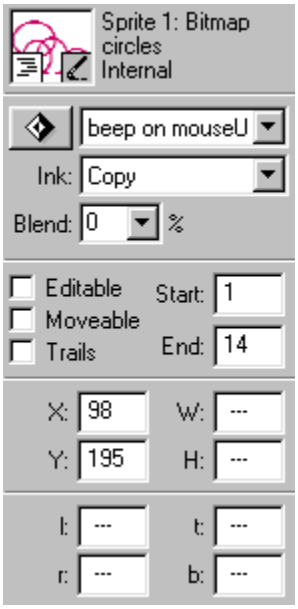
Shows the name, channel number, start frame, and end frame of the selection.

Inspectors > Sprite (Window menu) Control-Alt-S

The Sprite Inspector is a toolbar that allows you to view and modify sprite properties. The Sprite Inspector appears as an attachment at the top of the Score window (see **Sprite Toolbar**) or it can be displayed as a separate window.

Note: When it is displayed as a separate window, you can change the orientation of the Sprite Inspector by clicking inside the lower right corner of the inspector and dragging the inspector. Possible orientations are: vertical, horizontal, and stacked.

Click a window element for more information:



{button See also,AL(`Sprite_Inspector')}

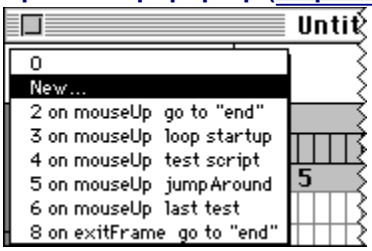
Cast member preview(Inspectors > Sprites)

Displays a small preview or image of the cast member for the selected sprite in the Score. Double-click the sprite's image to open a window in which you can edit the cast member.

Behavior Inspector (Inspectors > Sprites)

This button displays the Behavior Inspector.

Sprite Script pop-up (Inspectors > Sprites)



The Sprite Script pop-up lists all the frame and sprite scripts used in the current movie. The Sprite Script pop-up displays the script number associated with a selected frame in the Script channel. If the selected frame has no script associated with it, the pop-up is blank. If more than one script has been attached to a selected sprite, only the first script appears on the pop-up. Use the Behavior Inspector to see all attached scripts (or behaviors). You can use this pop-up to assign existing scripts to areas of the Score. Select the cells or sprite you want to apply the script to, then choose the script you want from the Script pop-up.

Clear Script removes the script from the selected frame or sprite.

New Script creates a new score script.

{button See also,AL(`Sprite_Insp_Script')}

Ink pop-up (Inspectors > Sprites)

You apply ink to sprites to change the way they appear on the Stage. The Ink pop-up also indicates the current ink applied to selected sprites in the Score.

For a demonstration of how each of the following ink effects work, see the **Ink Effects** movie.

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 non-white background. Sprites with the Copy ink animate faster than sprites with any other ink.

Matte removes the bounding box (rectangular area) around a sprite. Artwork within the boundaries is opaque. Matte functions much like the Lasso in the Paint window, in that the artwork is outlined rather than enclosed in a rectangle. Matte, like Mask, uses more memory than the other inks, and sprites with this ink animate more slowly than other sprites.

Background Transparent makes the pixels in the background color of the selected sprite appear transparent and permits the background to be seen. This effect uses more memory and may make your sprite animate more slowly.

Transparent makes all colors transparent so you can see the artwork through it.

Reverse reverses overlapping colors. A pixel that was originally white becomes transparent and lets the background show through unchanged. Reverse is good for making custom masks.

Ghost is useful for reversing black and white. When it is applied to the foreground sprite, any black pixel turns the pixel beneath it white. Anything white becomes transparent.

Not Copy, **Not Transparent**, **Not Reverse**, and **Not Ghost** are variations of the above four effects. The foreground image is first reversed, then the Copy, Transparent, Reverse, or Ghost inks are applied. These are good for odd effects. Like Transparent, the Not Transparent ink is good for reversing black and white. Just choose Not Transparent, select a white fill, then draw a rectangle on Stage on top of the artwork you want to reverse.

Mask ink allows you to define exactly what parts of a sprite are transparent and opaque. For mask ink to work, you must place a 1-bit 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 the white areas make the sprite transparent. This ink is especially useful for sprites in which you want some white areas to be transparent, and some opaque.

For example, to show a white car you would want the white body of the car to be opaque and the windows to be transparent. To create a mask, make a copy of the car in the next cast position, convert it to 1-bit color depth with the Transform Bitmap command, and then fill in the body of the car with black. In the Score, apply Mask ink to the sprite of the car. The body of the car becomes opaque and the windows transparent.

Blend ensures that the sprite uses the blend percentage specified in the Sprite Properties dialog box.

Darkest compares pixel colors in the foreground and background, and uses whichever pixel color is darkest.

Lightest compares 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 color value of the foreground sprite with the color value of the background sprite. If the value of the two colors exceeds the maximum color value, the numbering begins again at 1.

Add Pin is similar to Add. The foreground sprite's color is added to the background sprite's color, but the value of the new color cannot exceed the maximum color value.

Subtract subtracts the value of the foreground sprite's color from the value of the background sprite's color to arrive at the new color. If the color value of the new color is less than the minimum color in the color scale, the new color is determined by wrapping around and starting at the top of the color scale.

Subtract Pin subtracts the color value of the foreground sprite from the value of the background sprite. The value of the new color does not wrap around the color scale.

Tip: Mask and Matte use twice the memory of any other ink because Director has to internally create a duplicate

of the artwork.

{button See also,AL(`Sprite_Insp_Ink')}

Blend pop-up(Inspectors > Sprites)

This pop-up displays the blend percentages that can be applied to a sprite. You can vary the blend value between 0 and 100 percent.

{button See also,AL(`Sprite_Insp_Blend')}

Trails checkbox (Inspectors > Sprites)

If Trails is checked, the selected sprite remains on the Stage, leaving a trail of images along its path as the movie plays. If Trails is unchecked, the selected sprite is erased from previous frames as the movie plays. The checkbox also reflects the current selection.

Moveable checkbox (Inspectors > Sprites)

If Moveable is checked, the selected sprite(s) can be moved around on the Stage during playback and in projectors. If checked, the moveable setting is in effect only when the Playback Head is executing those frames that contain the moveable sprites. The checkbox also reflects the current selection.

The Moveable checkbox displays a dash (-) if the current selection includes sprites that don't all have the same setting.

Editable checkbox (Inspectors > Sprites)

If Editable is checked, the selected field sprites can be edited on the Stage during playback. This option is convenient for making a field sprite editable in some frames, and noneditable in others. You can turn this setting off when it is no longer required. If checked, the editable setting is in effect only when the Playback Head is executing those frames that contain the editable sprites. The checkbox also reflects the current selection. If the current selection includes sprites that don't all have the same setting, the Editable checkbox displays a dash (-).

You can set a field cast member to always be editable in the Score using the **Field Cast Member Properties** dialog box. Director ignores the Score's Editable checkbox setting for the cast member.

Using the Lingo statement `set the` **editable of sprite** `to TRUE` is the same as checking the Editable checkbox in the Score.

Start/End Frames (Inspectors > Sprites)

These fields display the start and end frames of the sprite.

{button See also,AL(`Sprite_Insp_Frames')}

Sprite Coordinates (Inspectors > Sprites)

These fields display the sprite coordinates. The top left corner of the stage is 0, 0.

- **X** and **Y** are the horizontal and vertical coordinates of the registration point.
- **W** and **H** are the width and height of the sprite.
- **l, r, t** and **b** are the left, right, top and bottom edges of the sprite's bounding rectangle.

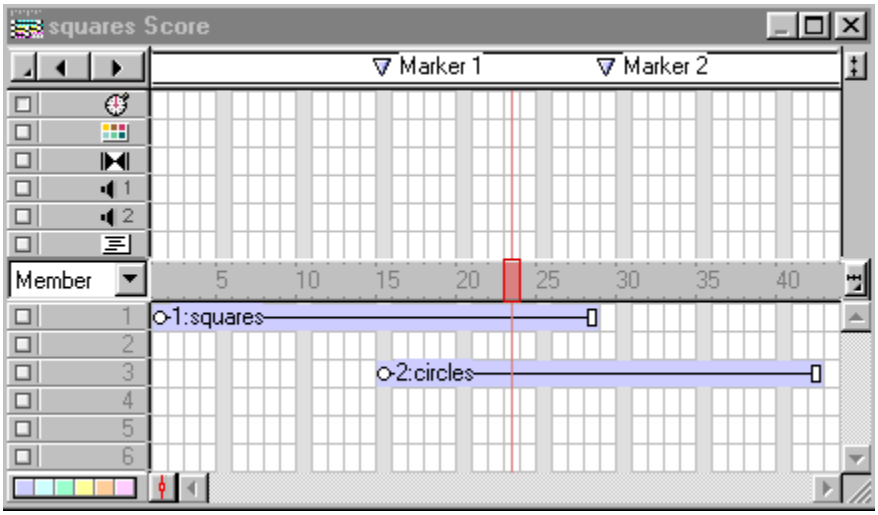
{button See also,AL(`Sprite_Insp_Coord_help')}

OVERVIEW | SCRIPTING | HOW TO | REFERENCE | TROUBLE | SHOW ME | WEB LINKS

Score window (Window menu) Control-4

The Score window contains the notation that describes your movie and is the primary tool for creating and editing animation. The Score window contains a record of everything that happens on the Stage.

Click a window element for more information:

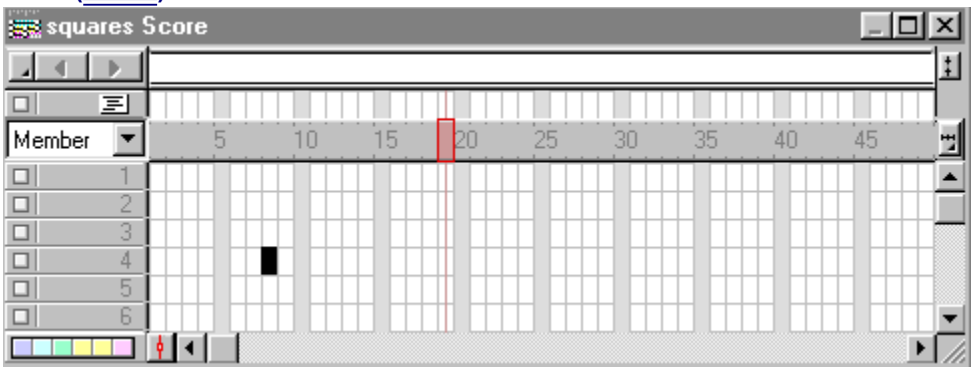


Click a topic for more information:

- Cells
- Frames
- Sprites
- Markers channel
- Markers pop-up
- Special effects channels
- Hide/Show Effects Channels
- Sprite channels
- Playback head
- Channel playback toggle
- Display pop-up
- Zoom pop-up
- Center Current Frame

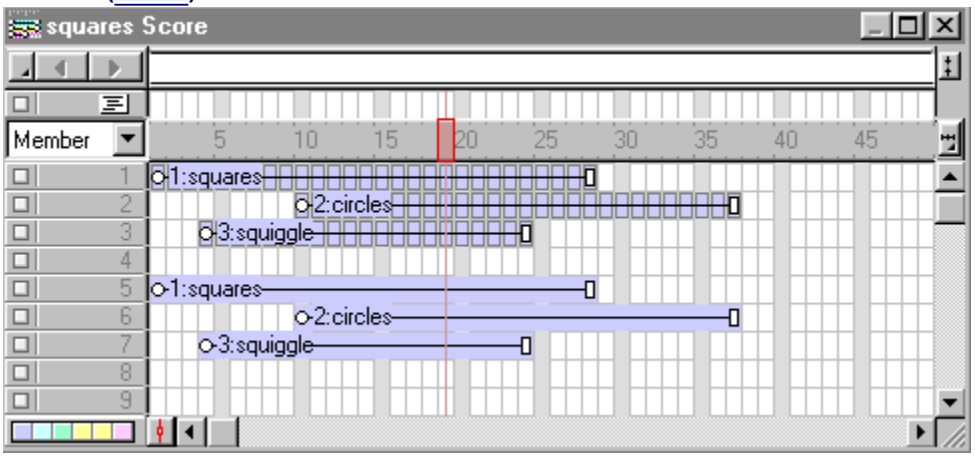
{button See also,AL(`Score_help')}

Cells (Score)



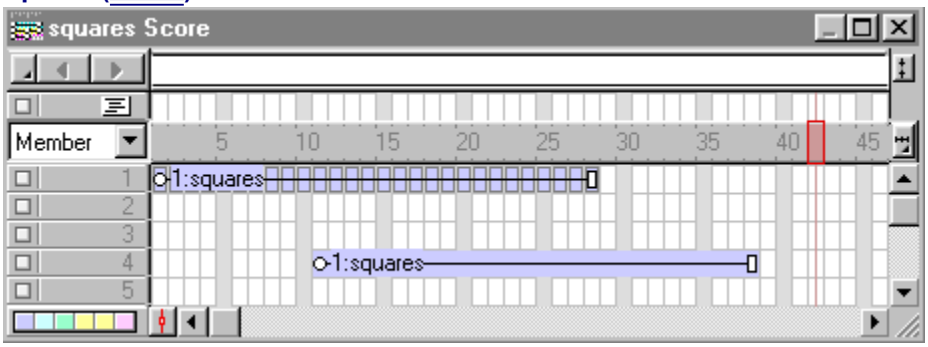
A cell is the smallest unit in the Score.
A horizontal row of cells can be viewed as a channel. A vertical column of cells can be viewed as a frame.
Cells are "empty" and contain no information until they are occupied by sprites or set in the special effects channels.

Frames (Score)



Frames are numbered, vertical columns of cells. Like a frame in a movie, each Score frame is a snapshot of your Director movie at a particular point in time.
An individual frame contains information about everything you see on the Stage when you stop a movie at that frame. A frame shows what each sprite in each channel is doing on the Stage at that moment.

Sprites (Score)



A sprite is an object that determines when, where, and how a cast member appears in a Director movie. Each sprite object consists of a cast member and a set of behaviors or characteristics.
For a demonstration of using sprites, see the **Cast Members and Sprites** movie.
A sprite is represented in the Score by a horizontal bar which can extend across any number of cells or frames in the Score. The first frame in which the sprite appears (the head of the sprite) is a keyframe indicated by a circle. The last frame in which the sprite appears (the tail of the sprite) is marked by a square.

- Extend the number of frames in which the sprite appears in the Score by dragging either the head or tail of the horizontal bar to the appropriate frame.
- The sprite can be moved to a position earlier or later in the movie by clicking and dragging the horizontal bar.
- Clicking anywhere on the body of the horizontal bar, selects the entire sprite, indicating that menu or keyboard commands will affect the entire sprite object.

{button See also,AL(`Score_Sprites_help')}

Zoom pop-up (Score)

The Zoom pop-up adjusts the display size of the frames in the Score.

{button See also,AL(`Score_Zoom')}

Playback Head (Score)

The Playback Head shows which frame is on the Stage. You can display an animation sequence by moving the Playback Head across the frames in the Score.

There are several ways to move the Playback Head:

- Position the cursor on the Playback Head, click the left mouse button, and drag the Playback Head to any frame in the Score.
- Click any frame in the Score and the Playback Head moves to this frame.
- If markers are in use, click the Next Marker or Previous Marker arrows to move the Playback Head to the designated marker.

{button See also,AL(`Score_Playback_help')}

Center Current Frame (Score)

The Center Current Frame button moves and centers the Score to the position of the Playback Head.

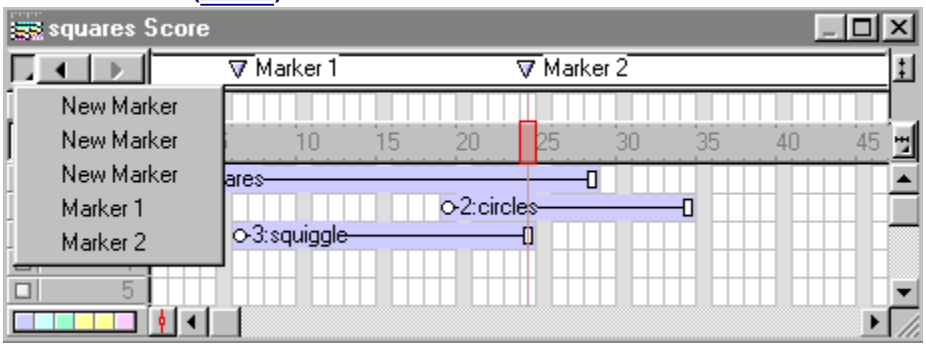
Markers channel (Score)

Use the Markers channel to place markers in the Score. Markers identify frames and can be used as navigation points within a movie. You can also use markers to add comments, speaker notes, or storyboard notes to specific frames of animation.

- Insert a marker by clicking any frame in the Markers channel.
- Reposition a marker by dragging it to the left or right within the Markers channel.
- Remove a marker by dragging the marker off the Score.

{button See also,AL(`Score_Markers_Chnl')}

Markers menu (Score)



Use the Markers menu or the Previous Marker and Next Marker arrows to position the playback head at a particular marker.

{button See also,AL(`Score_Markers_Menu')}

Special effects channels (Score)

A channel is a row of frames. The first six channels (located above the sprite channels) keep track of special effects. You can hide or display these channels using the Hide/Show Effects Channels toggle:

- Tempo
- Palette
- Transition
- Sound 1
- Sound 2
- Script

{button See also,AL(`Score_Effects_Chnl_help')}

Sprite channels (Score)

A channel is a row of frames. Sprite channels (located below the six special effects channels) describe the state of one or more sprites in a movie.

There are 120 sprite channels. Sprites that occupy higher-numbered channels appear in the foreground; sprites that occupy lower-numbered channels appear in the background.

{button See also,AL(`Score_Sprite_Chnl_help')}

Hide/Show Effect Channels toggle (Score)

The toggle shows or hides the special effects channels (tempo, palette, transition, sound, and script) of the Score.

Channel playback toggle (Score)

The square-shaped toggle at the left of the sprite channel turns the channel on or off. Turning a channel off tells Director to ignore the channel during playback. By default, all channels are on.

If you turn off a sprite channel, sprites in the selected channel do not appear when the movie is played.

If you turn off the script channel, Director ignores all scripts during playback. (This is the same as checking the Disable Scripts command in the Control menu.)

Color Selector (Score)

Use the Color Selector to apply color to sprites. The color only affects the display of sprite spans in the Score; it does not affect how sprites appear on Stage.

To apply a color, select the sprite and then click a color from the Color Selector.

Display menu (Score)

The Display pop-up lets you change the type of information displayed in each cell of the Score. It is located between the special effects and sprite channels at the left of the Score window. Use it to view different types of notation in the Score. By default, the Score displays cast member notation.

{button See also,AL(`Score_Display')}

Stage (Window menu) Control-1

The Stage is the backdrop for all Director movies. Choose Modify > Movie Properties to set the size of the Stage. In most cases, the edges of the Stage window extend to the edges of your screen, so you can use all of the monitor for your movie. Movies continue to play on the Stage when other windows are open and active. This permits you to study the movie in the Score, for example, while keeping an eye on the Stage.

Choose Window > Stage to bring the Stage to the front of the screen. It temporarily hides all open windows. To open a menu just click and hold.

{button See also,AL(`Stage')}

Control Panel (Window menu) Control-2

The Control Panel has controls similar to a VCR. Use it to play, stop, step forward or backward, or rewind your movie. The Control Panel is also used to loop animation, set tempo, and turn sound on and off. The Control Panel indicates the current frame number, the current tempo, and the actual duration of the current frame.

Choose Window > Control Panel to show or hide the Control Panel. The Control Panel buttons have corresponding commands on the Control menu.

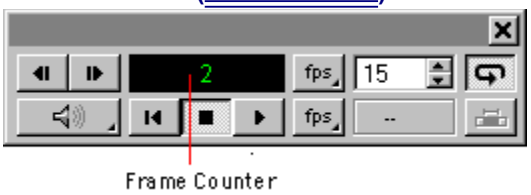
Use shortcuts on the number pad instead of clicking Control Panel buttons. See the diagram on the back cover of the Learning Lingo book.

Click part of the Control Panel for more information:

Click the name of a Control Panel indicator for more information:

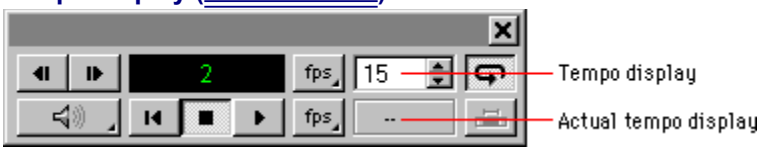
Frame counter
Tempo display
Actual tempo display

Frame counter (Control Panel)



The frame counter displays the number of the frames currently on the Stage. To go to a specific frame number, click the field, type a frame number, and press Enter.

Tempo display (Control Panel)



The Tempo display shows the tempo of the current frame. You can view the tempo in either FPS (frames per second), or SPF (seconds per frame). Click the Tempo Mode button and choose the mode you want from the pop-up. Seconds per frame measures the duration of a frame in milliseconds.
To change the tempo, click or press the up or down arrow next to the Tempo display. To use a specific tempo, click the Tempo display, type a tempo, and press Enter. If you are entering a tempo in frames per second (FPS), you must enter a whole number. If you are entering a tempo in seconds per frame (SPF), you must type a decimal (.) before entering a value.

If there are no tempo settings in the tempo channel, the Control Panel displays the default tempo. (If there is a tempo setting in the tempo channel, the Control Panel displays the tempo of the current frame.)

Tip: You should always enter a tempo setting in the first frame in the tempo channel.

{button See also,AL(`Control_Panel_Tempo_display')}

Actual Tempo display (Control Panel)

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

While a movie is playing, the Actual Tempo display shows how fast the movie is playing. Click the Actual Tempo Mode button and choose a mode for viewing the tempo from the pop-up. There are four different modes:

FPS (frames per second) shows the actual duration of the previous frame in frames per second (FPS), including the time necessary to execute any Lingo scripts except Exit Frame scripts. If the movie is stopped, the display shows the duration of the current frame. Frames that don't have a recorded tempo value display "--" instead of a value for the Actual Tempo.

SPF (seconds per frame) shows the duration of the current frame in milliseconds.

Running Total provides a quick summary of elapsed seconds from the beginning of the movie to the current frame.

Estimated Total provides a more accurate but slower calculation of elapsed time. It is useful if you want to include transitions and palette changes in determining frame durations.

Computing estimated frame durations can reduce playback speed, so don't leave the Actual display in Estimated Total mode.

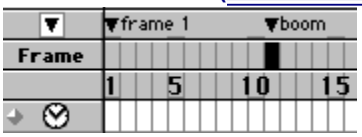
If the movie is locked (using the Lock Frame Durations option in the Movie Properties dialog box), the Actual Tempo display shows the previously recorded frame durations.

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

- Clear all recorded frame durations from the movie if you want to record frame durations for a section of the movie and lock them. To clear all recorded frame durations, press Alt while clicking the Lock option in the Movie Properties dialog box. During playback, frames that don't have a recorded duration instead display "--" .
- Since not all computers are equally fast, you can step through the movie frame-by-frame and compare the actual frame durations to the tempos you've set for the movie. To find frames that have longer durations than the current tempo, set both the Tempo and Actual Tempo displays to show seconds per frame (SPF); then step through the range of frames, and look for any frame whose actual duration is longer than the current tempo.

{button See also,AL(`Control_Panel_Actual_display')}

Markers window (Window menu) Control-Shift-M



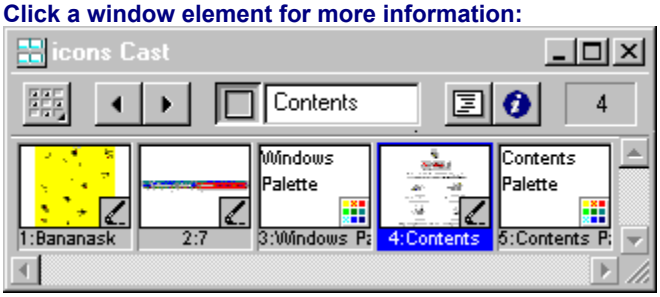
The Markers window lets you write comments associated with markers you set in the Score. For example, a Director animation can have staging or acting directions, storyboard scripts, or speaker's notes written in the Markers window and tied to specific frames in the Score. Storyboards, transparencies, or handouts can be printed that include pictures of selected frames of your movie along with the comments written in the markers window. Double-click a marker in the Score to open the Markers window to the comment associated with that frame.

{button See also,AL(`Markers')}

Cast window (Window menu) Control-3

The Cast window displays the cast members in the current cast. A cast is a database of graphics, sounds, Color Palettes, scripts, buttons, transitions, digital video movies, and text used in a Director movie.

Click a window element for more information:



Click a topic for more information:

Cast window positions and thumbnails

Moving cast members

Cast Preferences

Cast Properties

Viewing multiple casts

Open multiple Cast windows to display the different casts in your movie, or select a different cast to display in the current window.

- To open a new Cast window for a particular cast in the current movie, choose Window > Cast and then select the name of the cast from the submenu.
- To open a new Cast window for the current cast, make a Cast window active and then choose Window > New Window.
- 

To change the cast displayed in a Cast window, click the cast pop-up and choose the cast you want to display from the pop-up.

The Cast window title bar indicates whether a cast is internal, external linked, or external unlinked.

Cast window size

A movie's cast can contain up to 32,000 cast members. You control the row width and the number of visible rows using **Cast Preferences** in the File menu.

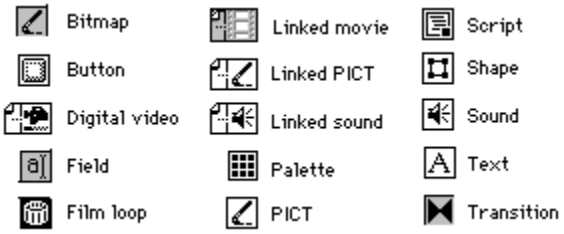
Selecting cast members

Clicking any cast member selects it. Select a range of cast members by Shift-clicking. Control-click selects multiple non-adjacent cast members. Individual cast members can be cut, copied, pasted, or cleared from the Cast window.

{button See also,AL(`Cast_help')}

Cast window positions and thumbnails

Each cast member position is identified by a number and, optionally, a name. For every occupied position in the Cast window, a thumbnail image is displayed that represents the cast member's type.



Note: Thumbnail images for Xtra cast members vary.

Cast members with scripts display a Script icon in the lower left corner. To control whether or not Director displays a script icon in the cast, use the Show Cast Member Script Icons checkbox in the **Cast Preferences** dialog box.

Tip: Double-clicking a cast member is a shortcut for opening the Paint window for a graphic cast member, the Text window for a text cast member, the Color Palettes window for a palette cast member, the Script window for a script cast member, and the Digital Video window for a digital video cast member. Alt-double-clicking is a shortcut for opening the cast member in a new window.

Choose Cast pop-up

Use the cast pop-up to select which cast is displayed in the current Cast window. You can choose between any internal cast or external cast linked to the current movie. You can also choose New Cast to create a new cast.

To open a new Cast window, make sure a Cast window is active and then choose New Window from the Window menu.

Drag Cast Member button

The Drag Cast Member button moves the selected cast members to the Stage or Score, or moves them within the cast. When cast members are moved within the Cast window, the Score window is updated with their new position. When you press and hold down this button, the cursor changes to a closed hand to let you drag one or more selected cast members. Using this button is the same as clicking and dragging a selected cast member in the Cast window. Use this button to move selected cast members that may not currently be visible in the Cast window, if you've scrolled to a new location.

To use the Drag Cast Member button to move a cast member to a new location within the Cast window:

1. **Select the cast members you want to move.**
2. **Scroll to the new location in the Cast window where you want to insert the selected cast member.**
3. **Drag from the Drag Cast Member button to the new location in the Cast window. As you drag within the Cast window, a blinking insertion bar indicates the location where the cast member will be inserted.**
4. **Release the mouse button to insert the selected cast member at the new location.**

Previous, Next Cast Member arrows

The Previous and Next Cast Member arrows let you navigate to the previous or next cast member, skipping over empty cast members.

Cast Member Properties button

The Cast Member Properties button displays the **Cast Member Properties** dialog box for the selected cast member. If the selection consists of more than one cast member, the dialog box displays the number of cast members selected, their total size, and purge priority.

Shortcut: Control-I also opens the Cast Member Properties dialog box.

Cast Member Script button

The Cast Member Script button opens a new Script window or makes an existing Script window active for the selected cast member, or for the first selected cast member if more than one are selected. If the selected cast member has no script associated with it, clicking this button opens a new Script window, creating a script for the cast member. This is the same as clicking the Script button in the Cast Member Properties dialog box.

Shortcut: Pressing Control-' (apostrophe) is the same as clicking this button.

Cast Member Number button

The Cast Member Number button displays the position of the selected cast member in the Cast window or the position of the first selected cast member in a multiple selection.

Tip: When the Cast window is front-most, typing the number of an existing cast member automatically scrolls the Cast window to the cast member's location and selects it. After you've stopped typing, the Cast window will scroll to show the new selection.

Cast Member Name field

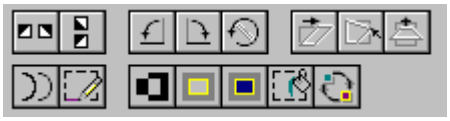
The Cast Member Name field displays the name of the selected cast member or the first selected cast member in a multiple selection. Click and type into the name area to enter or edit the name of a cast member. Press Enter to confirm your changes.

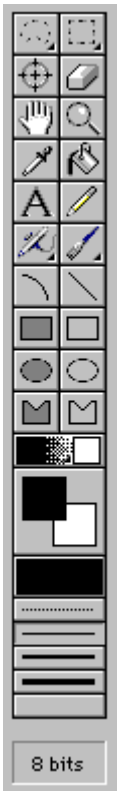
OVERVIEW | SCRIPTING | HOW TO | REFERENCE | TROUBLE | SHOW ME | WEB LINKS

Paint window (Window menu) Control-5

{button See also,AL(`Paint')}

The Paint window has a complete set of paint tools and inks you can use to create and edit bitmapped cast members for your movies.





Click part of the Paint window illustrated or on a topic below for more information:

Creating cast members within Director
Using rulers in the Paint window
Zooming in and out in the Paint window
Using the Effects toolbar
Using shapes

Shortcut: A shortcut for opening the Paint window is to double-click a bitmapped cast member on the Stage or in the Score or Cast windows. The Paint window opens with that cast member showing.

The Paint window tools

The list below shows whether clicking and/or dragging makes the tool work. Click the name of a Paint window tool for more information:

Lasso-(Drag) Selects irregular shapes
Marquee-(Drag) Selects rectangular areas
Registration-(Click) Sets registration point
Eraser-(Drag) Erases artwork
Hand-(Drag) Moves artwork within window
Magnifying Glass-(Click) Scales the view
Eyedropper-(Click or drag) Picks foreground color
Paint Bucket-(Click) Fills with foreground color or current pattern
Text-(Click) Starts text entry
Pencil-(Click or drag) Toggles pixels between foreground and background color
Air Brush-(Click or drag) Sprays foreground color or current pattern
Brush-(Click or drag) Paints foreground color or current pattern
Arc-(Drag) Draws arcs (one quarter of an ellipse or circle)
Line -(Drag) Draws straight lines
Rectangle-(Drag) Draws hollow or filled rectangles and squares
Ellipse-(Drag) Draws hollow or filled ellipses and circles
Polygon-(Click) Draws hollow or filled polygons

Tip: If you Control-click the image in the Paint window, your view of the artwork will zoom in to a magnified view. In most cases, pressing the Shift key while dragging a tool constrains it to horizontal or vertical. The Ellipse and Rectangle tools are constrained to a perfect circle or square when Shift-dragging.

OVERVIEW | SCRIPTING | HOW TO | REFERENCE | TROUBLE | SHOW ME | WEB LINKS

Lasso tool (Paint window)

Use the Lasso tool to select an area. Once selected, drag, cut, copy, or clear the artwork. You can also use the following buttons on the effects toolbar: Invert, Trace Edges, Fill, Darken, Lighten, Smooth, and Switch Colors.

Tips:

- When artwork is selected with the Lasso, hold down the Alt key while dragging the artwork to make a copy of it.
- If you press the Alt key while dragging the Lasso, the Lasso draws straight lines to select a polygon shape. Click the Lasso to anchor a point and draw another straight line. Double-click when you reach the end of your selection.
- Pressing the Shift key while dragging the object constrains its movement to a horizontal or vertical line. To move the cast member in one-pixel increments, select it on the Stage and use the arrow keys on your keyboard.

Note: Use the Lasso to select everything but the pixels of a certain color. The color of the pixels not selected is determined by where you begin to drag the Lasso. For example, if you're selecting an object that is red, white, and blue, and you only want to select the red and white pixels in the object, begin your drag on a blue pixel. Then, only the red and white pixels will be selected. If you want to avoid this effect, use the No Shrink option in the **Lasso pop-up**.

Lasso pop-up (Paint window)

Press and hold the mouse button while the pointer is on the Lasso tool to display the pop-up and modify how the Lasso works.

- **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.

Selection Marquee (Paint window)

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.

Select the contents of the visible part of the current cast member by double-clicking the Marquee.

Tips:

- Stretch and compress art that is selected with the Marquee using Control-drag.
- Make a copy of artwork that is selected with the Selection Marquee using Alt-drag.

{button See also,AL(`Paint_Selection_rectangle')}

Marquee tool (Paint window)

If you press and hold the mouse button when the pointer is positioned on the Marquee tool, the Marquee pop-up appears with commands to modify the action of the tool.

- **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 like the Lasso tool and selects the pixels according to the color of the pixel beneath the crosshair when you started your drag.
- **See Thru** tightens the Marquee around the object and applies the Transparent ink.

Any artwork selected with the Lasso or Marquee can be further modified with ink effects or commands on the Paint toolbar.

Double-clicking the Marquee tool selects the entire cast member.

As with the Lasso, you can reposition the selected area of the cast member. Move the crosshair into the selected area so that the crosshair turns into a pointer and drag the selected area to reposition it. There are several key combinations that affect the selected area when you drag. They include:

- **Copy-**Alt-drag
- **Stretch-**Control-drag
- **Stretch proportionally-**Control-Shift-drag
- **Constrain to horizontal or vertical-**Shift-drag
- **Clear-**Backspace or Delete
- **Scale-**Control-Alt-drag

Use the keyboard arrow keys to horizontally or vertically nudge a selection.

Hand tool (Paint window)

The Hand tool moves the view within the Paint window, changing your position in the window relative to the artwork. Click the Hand tool to select it, then drag the artwork to pan the view.

Tip: Press the Spacebar and click the mouse as a shortcut to turn any tool into the Hand tool.

Magnifying Glass tool (Paint window)

Click to zoom in on an area. Shift-click to zoom out.

Text tool (Paint window)

Use the Text tool to type text in any font, size, or style in the Paint window.

The text you create in the Paint window is bitmapped. Text can be dragged around the Paint window before you deselect it. However, once you click outside the field after creating the text, you cannot edit its font, size, or style. To change the font, size, or style after clicking, you must erase the text and replace it with new bitmapped text.

Modify selected text with ink effects from the Ink pop-up, patterns from the Patterns pop-up, or foreground and background colors with the Foreground and Background color chips in the Paint window.

{button See also,AL(`Paint_Text')}

Paint Bucket tool (Paint window)

The Paint Bucket fills any enclosed area with the currently selected color and pattern. Modify the fill with the ink effects in the Ink pop-up in the Paint window. If there is a break in the outlined area you are filling, the paint will leak out and fill the surrounding area. If this happens, choose Edit > Undo Bitmap and then View > Zoom In to get a magnified view and inspect the outline for breaks.

Tip: Double-click the Paint Bucket tool to open the **Gradient Settings** dialog box.

Air Brush tool (Paint window)

The Air Brush sprays the currently selected color and pattern. To modify the spray, choose the ink effects from the Ink pop-up in the Paint window. The longer you hold the Air Brush in one spot, the darker it fills in the area.

Air Brush pop-up

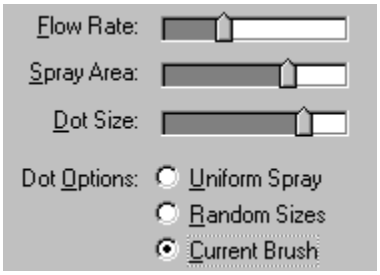
If you press and hold the mouse button when the pointer is positioned on the Air Brush, the Air Brush pop-up appears. Each of the five settings in the pop-up can be defined so you can have several types of spray available without opening the Air Brush Settings dialog box.

To define a setting:

1. **Choose the menu item you want to define from the Air Brush pop-up.**
2. **Click the Air Brush again and choose Settings from the Air Brush pop-up.**
3. **Select the type of spray you want in the Air Brush Settings dialog box.**
4. **Click Set.**

The choices you make in the Air Brush Settings dialog box are assigned to the menu item and remain until you change them.

Air Brush Settings dialog box



The Air Brush Settings dialog box defines the size of the area the Air Brush covers, the size of the dots in the Air Brush's spray, and the flow speed of the Air Brush's paint.

Tip: Double-click the Air Brush in the Tool palette to open the Air Brush Settings dialog box. Use this dialog box to set the size of the Air Brush's spray, the size of the dots of paint it sprays, and how fast it sprays paint.

Uniform Spray causes drops sprayed by the Air Brush to be uniformly sized.

Random Sizes sprays with randomly sized drops.

Current Brush sprays with drops shaped like the current Paintbrush.

Spray Area sets the size of the Air Brush's spray area. To change the spray area, drag the Size scrollbar.

Dot Size sets the size of the dots sprayed by the Air Brush. To change the dot size, drag the Dot Size scrollbar.

Flow Rate controls how fast the Air Brush covers an area with paint. To change the flow, drag the Flow Speed slider.

Brush tool (Paint window)

The Brush draws with the currently selected colors, ink effect, or fill pattern. Double-click the Paintbrush to change the size and shape of the brush. When the Brush Settings dialog box appears, click the Brush shape you need, and then click Set.

Paintbrush pop-up

The Paintbrush pop-up is similar to the Air brush pop-up. Press and hold the mouse button while the pointer is positioned on the Paintbrush tool to open the pop-up.

Each of the five settings in the pop-up can be defined so you can have several brush shapes available without opening the Brush Settings dialog box.

To define a setting:

1. **Choose the menu item you want to define from the Brush pop-up.**
2. **Click the Brush tool again and choose Settings from the Paintbrush pop-up.**
3. **Select the brush shape you want in the Brush Settings dialog box.**
4. **Click Set.**

The choices you make in the Brush Settings dialog box are assigned to the menu item in the pop-up and remain until you change them.

Brush Settings dialog box

The Brush Settings dialog box lets you change the shape of the Brush. You can double-click the Paintbrush in the Paint window Tool palette.

Click a dialog box option for more information:

Custom/Standard selects standard default brush shapes or lets you create your own brush shapes from the set of custom brush shapes. Only the custom Brush shapes are editable.

Change a custom Brush shape by clicking one of the Brush shapes on the left. You can 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 picks up the shape on the screen at the point you click.

Right/Left arrows move the Brush shape one pixel to the right or to the left.

Up/Down arrows move the Brush shape up or down one pixel.

Black/White square reverses the colors of the Brush shape (for example, black becomes white and white becomes black).

Copy copies the Brush shapes to the Clipboard.

Paste pastes the Brushes into the custom set of Brush shapes.

Pencil tool (Paint window)

The Pencil tool creates a one-pixel-wide line. On a black and white cast member, the Pencil draws black pixels on a white background and white pixels on a black background. On a color cast member, the Pencil draws with the currently selected foreground color unless you are drawing on pixels that are the foreground color. In that case, the Pencil draws in the background color.

Double-click the Pencil tool to magnify the object view in the Paint window. You can also zoom in while using the Pencil or any other tool by Control-clicking.

Use the magnified view to edit the cast member pixel by pixel. Use any of the paint tools while in a magnified view. Click the reduced view in the upper right corner of the Paint window to returns to a 100% view. Double-click the Pencil in the Tool palette, while in magnified view, to also return to a 100% view.

Filled Rectangle/Rectangle tool (Paint window)

The Rectangle tool draws rectangles of any size. Click the Rectangle tool to draw an outline in the current foreground color as you drag the crosshair. Click the Filled Rectangle tool to fill the rectangle with the current foreground and background colors selected. The thickness of the rectangle's border is controlled with the Line Width Selector at the bottom of the Tool palette.

Tip: Double-click the Filled Rectangle tool to open the **Gradient Settings** dialog box.

To constrain the rectangle to a square, press the Shift key as you drag. If you press the Alt key while drawing a rectangle, the border is drawn with the current pattern.

Eraser tool (Paint window)

The Eraser clears the portion of the cast member you drag across. The Eraser always clears to white. Double-click the Eraser tool to erase everything in the Paint window's visible area.

Filled Ellipse/Ellipse tool (Paint window)

The Filled Ellipse/Ellipse tool creates circles and ovals. Like the Rectangle tool, the Ellipse is an outline if you click the Ellipse tool. The ellipse is filled with the selected foreground and background colors, ink, and pattern when you click the Filled Ellipse tool. The thickness of the border is controlled by clicking the Line Width Selector at the bottom of the Tool palette.

Tip: Double-click the Filled Ellipse tool to open the **Gradient Settings** dialog box.

When you hold down the Shift key as you draw, the Ellipse tool draws perfect circles. If you use the Ellipse tool while pressing the Alt key, the border of the circle is drawn with the currently selected pattern.

Filled Polygon/Polygon tool (Paint window)

The Filled Polygon/Polygon tool draws polygons with as many sides as you want. The Polygon tool creates an outline and the Filled Polygon tool draws an area filled with the current foreground and background colors, inks, and patterns. When you click the tool, the pointer becomes a crosshair. Click in the Paint window to start drawing the side of the polygon. Each time you click, a line is drawn from the spot you clicked previously. Double-click to connect the end-point of the shape to the beginning of it.

Line thickness is controlled by the Line Width Selector at the bottom of the Tool Palette. Press the Alt key while drawing a polygon, to draw the border with the currently selected pattern.

Tip: Double-click the Filled Polygon tool to open the **Gradient Settings** dialog box.

Line tool (Paint window)

The Line tool draws straight lines at any angle. When you hold down the Shift key, the Line tool draws vertical, horizontal, or 45-degree lines, depending upon the direction you begin to drag. Change the line width by clicking the Line Width Selector in the Tool palette.

The line is drawn with the currently selected foreground color and ink effect.

Tip: Press the Alt key while drawing a line to cause the line to be drawn in the currently selected pattern.

Arc tool (Paint window)

The Arc tool draws one quarter of an ellipse or circle. When the tool is active, the pointer becomes a crosshair. Drag the crosshair from the starting point of the line and move the pointer to see the curve. Experiment with dragging the tool until it produces the line you need. The thickness of the arc is controlled with the Line Width Selector.

The line is drawn with the currently selected foreground color and ink unless you press the Alt key while dragging, in which case the arc is drawn with the current pattern.

Registration tool (Paint window)

The default registration point for a bitmap image is the center of the cast member in the Paint window. However, the registration point for shapes, buttons, and text cast members is always the upper left corner of the image. Using the Registration Point tool to click a point in the Paint window sets the registration point at that location.

Tip: Double-click the Registration tool to reset the default registration point at the center of the cast member.

{button See also,AL(`Paint_Registration')}

Eyedropper tool (Paint window)

The Eyedropper tool is used to match colors. When you select the Eyedropper tool, any color you click in the Paint window becomes the foreground color. Use it to match colors without opening the Color Palette.

Ink pop-up (Paint window)

Choose an ink effect from the Ink pop-up 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. To learn more about using ink effects, see the **Ink Effects** movie.

Ink (click for info)	B&W	Color	Works best with
Normal	Ã	Ã	Solids and patterns
Transparent	Ã	Ã	Patterns
Reverse	Ã	Ã	Solids and patterns
Ghost	Ã	Ã	Solids (b&w) and patterns (color)
Gradient	Ã	Ã	Paintbrush, Paint Bucket, Shape tools
Reveal	Ã	Ã	Paintbrush, Shape tools
Cycle		Ã	Solids and patterns
Switch		Ã	Paintbrush
Blend		Ã	Solids and patterns
Darkest		Ã	Patterns
Lightest		Ã	Patterns
Darken		Ã	Paintbrush
Lighten		Ã	Paintbrush
Smooth		Ã	Paintbrush
Smear		Ã	Paintbrush
Smudge		Ã	Paintbrush
Spread	Ã	Ã	Paintbrush
Clipboard	Ã	Ã	Paintbrush

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Normal (Ink pop-up of the Paint window)

Normal is the default ink. It is opaque and maintains the color of the current foreground color and pattern.

Transparent (Ink pop-up of the Paint window)

Transparent ink makes the background color of patterns transparent so you can see artwork drawn previously in the current cast member through the pattern.

Reverse (Ink pop-up of the Paint window)

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 pop-up of the Paint window)

Ghost in black and white creates an image than can only be seen when drawn over a black background. In color, Ghost draws with the current background color.

Gradient (Ink pop-up of the Paint window)

Gradient lets you paint with the gradient fill selected in the **Gradient Settings** dialog box. A gradient fill is one that progresses from one color, the foreground, to another color called the destination color. You can paint with Gradient ink with the Paintbrush, Paint Bucket, or Shape tools.

Reveal (Ink pop-up of the Paint window)

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 Air Brush. Since it is impossible to mask certain shapes for the Air Brush, spray an area with the Air Brush 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 Air Brush pattern in the previous window.

Cycle (Ink pop-up of the Paint window)

Cycle is a color ink. As you draw with a cycling 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.

Switch (Ink pop-up of the Paint window)

Switch changes any pixel that is the current foreground color to the current gradient destination color as you paint over that color.

This ink only works when your computer is set to 256 colors.

Blend (Ink pop-up of the Paint window)

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 (Ink pop-up of the Paint window)

Darkest is a useful ink for colorizing 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 (Ink pop-up of the Paint window)

Lightest is a useful ink for colorizing 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 (Ink pop-up of the Paint window)

Darken makes colors darker. The more the Paintbrush passes over an area, the darker it becomes. The color of the foreground, background, or 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. Vary the rate of this ink effect in the Paint Preferences dialog box.

Lighten (Ink pop-up of the Paint window)

Lighten makes existing artwork lighter. The more times you pass over the artwork with the Paintbrush, the lighter it becomes. The color of the foreground, background, or 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. Vary the lightness of this ink effect in the Paint Preferences dialog box.

Smooth (Ink pop-up of the Paint window)

Smooth blurs existing artwork when painted with the Paintbrush. It is not directional like Smear and Smudge. The color of the foreground, background, or destination inks have no effect on Smooth. Smooth only works with art already in the Paint window. Use it to smooth out jagged edges.

Smear (Ink pop-up of the Paint window)

Smear works with the Paintbrush and is similar to mixing paint. Any area you drag across with a Smear ink is spread in the direction of the brush and fades as it gets farther from its source. The color of the foreground, background, or destination inks have no effect on Smear. Smear only works with art already in the Paint window.

Smudge (Ink pop-up of the Paint window)

Smudge is a color ink for the Paintbrush that is similar to Smear. It is also like mixing paint. The colors fade faster as they are spread. The color of the foreground, background, or destination inks have no effect on Smudge. Smudge only works with art already in the Paint window.

Spread (Ink pop-up of the Paint window)

Spread works with the Paintbrush in color. Whatever is under the Paintbrush 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 (Ink pop-up of the Paint window)

Clipboard uses the current contents of the Clipboard as a pattern to paint with.

Gradient Colors selection bar (Paint window)

A gradient is a blend of a range of colors that can be used for shading, highlights, backgrounds, and special effects. On a black-and-white monitor, gradients are created with a pattern of black and white pixels that fade from black to white or vice versa. With a color monitor, the two colors that form the beginning and end of a gradient are the foreground color and the foreground and background color. The range of colors between the foreground and background colors is used with the Gradient, Cycle, and Switch inks. Select the background color using the right Gradient Colors selection bar.

To set the current foreground color, click the left side of the selector and choose a color from the pop-up Color Palette. To set the Gradient background color, click the right side of the selector and choose a color from the pop-up color palette.

Tip: Hold down the Alt key while pressing the up or down arrow key to cycle through the colors in the Gradient color chip.

Gradient Colors (Paint window)

The Gradient Colors selection bar creates a blend of colors that you can use for backgrounds, highlights, shading, and special effects. Limited gradient effects can be created with a black-and-white cast member.

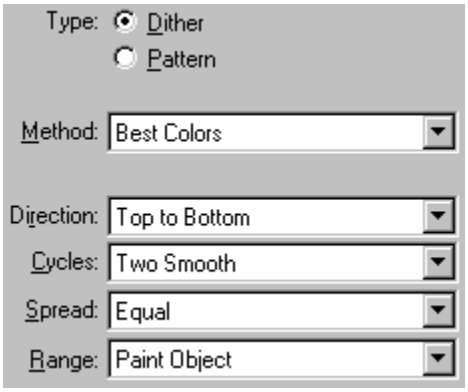
The gradient's foreground color and background color can be selected in the Paint window with the color chips in the Gradient Colors selector or with the controls in the Gradient Settings dialog box. When in the Paint window, use the popup palette on the Foreground color chip to select the foreground color. Pick the destination color with the pop-up at the right side of the Gradient Colors selection bar above the foreground and background color chips. The current foreground color is displayed to the left of the selection bar and the current destination color is displayed on the right side of the selection bar.

To change gradient settings, click the area between the foreground and destination color chips and choose Gradient Settings from the pop-up.

Gradient Settings dialog box

In the Gradients Settings dialog box, set the foreground and background colors as well as the pattern to use with your gradient. There are several pop-ups that control the style of your gradient fill. Each choice you make is immediately previewed on the left.

Click a dialog box option for more information:



Type (Gradient Settings)

Type determines whether the gradient is made with the pattern you select with the Pattern chip pop-up in the Paint window, or with a dithered pattern. If you choose Dither, only dithering options appear on the Method pop-up below. If you choose Pattern, only pattern options appear.

Method (Gradient Settings)

Method determines the way the gradient fills an area in the Paint window. The options on the menu list determine where the dark and light colors of your blend are located.

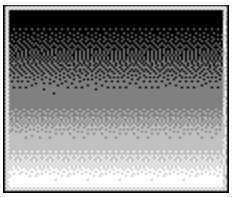
- **Dither Best Colors** ignores the order of the colors in the palette and only uses colors that create a continuous blend from foreground to background colors and blends them with a dithered pattern. Dithering is a technique of creating color with two or more colors of pixels interspersed together.
- **Dither Adjacent Colors** uses all colors between the foreground and background colors and blends them with a dithered pattern.
- **Dither Two Colors** uses only the foreground and the background colors and blends them with a dithered pattern.
- **Dither One Color** uses only the foreground color and fades it with a dithered pattern.
- **Standard Colors** ignores all colors between foreground and background and adds several blended colors with a dithered pattern to create the gradient.
- **Multi Colors** ignores all the colors between foreground and background and adds several blended colors with a randomized dithered pattern to create a smooth gradient. You can interrupt the drawing of this kind of dither by clicking anywhere in the dialog box.
- **Pattern Best Colors** ignores the order of the colors in the palette and only uses colors that create a continuous blend of the foreground and background colors.
- **Pattern Best Transparent** ignores the order of the colors in the palette and only uses those colors that create a continuous blend of the foreground and background colors. White pixels in patterns created with this method are transparent.
- **Pattern Adjacent Colors** uses all the colors in the palette between the foreground and background for the gradient.
- **Pattern Adjacent Colors Transparent** uses all the colors in the palette between the foreground and background for the gradient. White pixels in patterns created with this method are transparent.

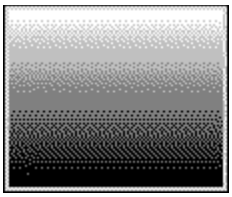
Direction (Gradient Settings)

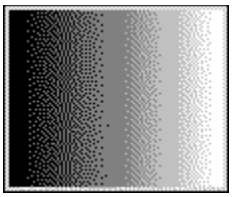
The Direction popup determines the way the gradient fills an area in the Paint window. The options on the menu list determine where the dark and light colors of your blend are located.

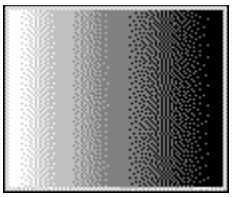
Click the name of a gradient direction to see an illustration:

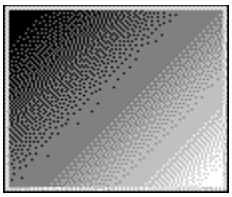
- **Top to Bottom** puts the foreground color at the top and the destination color at the bottom.
- **Bottom to Top** puts the destination color at the top and the foreground color at the bottom.
- **Left to Right** puts the foreground color on the left and the destination color on the right.
- **Right to Left** puts the foreground color on the right and the destination color on the left.
- **Directional** 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.
- **Sun Burst** starts filling at the edge of the artwork and moves in concentric circles to the center.

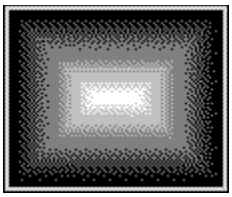


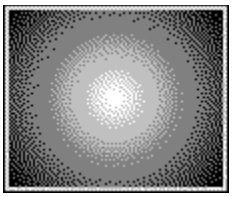










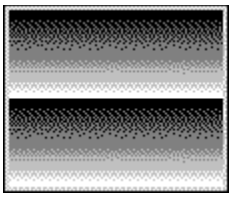


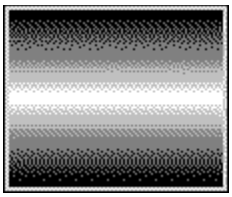
Cycles (Gradient Settings)

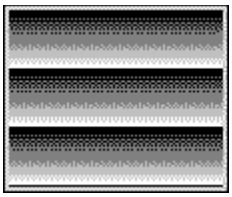
Cycles control the number of times the gradient is created within one filled area and whether or not the colors cycle through the palette in one direction only, or auto reverse at the end of one pass through the palette. Sharp cycles have a banded appearance, while smooth cycles go from foreground to destination, then back to foreground.

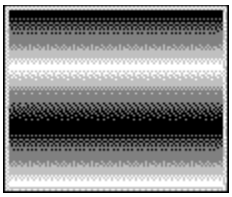
Click the name of a gradient cycle to see an illustration:

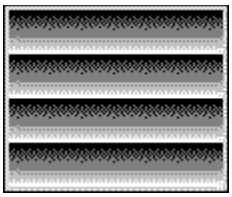
- **One** takes the gradient once through the range of colors you define.
- **Two Sharp** takes the gradient through the range of colors twice, from foreground to destination and from foreground to destination.
- **Two Smooth** takes the gradient from foreground to destination, then from destination to foreground.
- **Three Sharp** takes the gradient from foreground to destination three times.
- **Three Smooth** takes the gradient from foreground to destination, destination to foreground, foreground to destination.
- **Four Sharp** takes the gradient from foreground to destination four times.
- **Four Smooth** takes the gradient from foreground to destination, destination to foreground, foreground to destination, and destination to foreground.

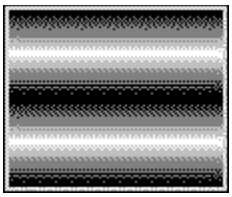












Spread (Gradient Settings)

Spread options let you choose how to distribute colors between the foreground and the destination colors of the gradient.

- **Equal** provides an even spacing of colors between the foreground and the 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.

Range (Gradient Settings)

Range options determine whether the full range of the gradient is created over the paint object, cast member, or the entire Paint window. The options provide greater control over how the gradient is created relative to the cast member's position on the Stage or in the Paint window.

- **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 with respect to 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.

Foreground color chip (Paint window)

Foreground color is the color displayed in the Foreground color chip in the Foreground and Background colors bar. It's also displayed in the color chip on the left side of the Gradient Colors selector bar. The foreground color is the color you work with when you're using the solid pattern and the Normal ink effect.

Press the up or down arrow key to cycle through the colors in the color palette associated with the Foreground color chip.

Double-click the Foreground, Background, or Destination color chip to open the **Color Palettes window**.

Background color chip (Paint window)

Background color is the color displayed in the lower right color chip in the Foreground and Background colors bar. The background color is the secondary color that appears in a pattern. When used with the Transparent ink, the background color in a pattern is transparent.

Tip: Hold down the Shift key while pressing the up or down arrow keys to cycle through the colors in the color palette associated with the Background color chip.

Pattern chip (Paint window)

The current pattern is displayed in the Pattern chip.

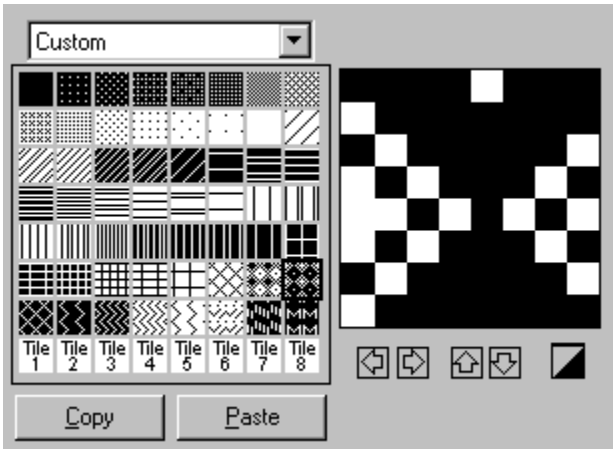
Click the Pattern chip to select a new pattern from the pop-up palette.

Tips:

- Pressing the Alt key before displaying the pattern palette permanently changes the patterns to shades ranging from the foreground color to the background color rather than the set of patterns you see without pressing the Alt key. Press the Alt key again to return to the default set of patterns.
- Double-click the Pattern chip to open the **Pattern Settings** dialog box. Use the dialog box to edit or select new sets of patterns.

Pattern Settings

Click part of the illustration for more information:



Custom/Standard pop-up selects from the standard default patterns to create patterns from the set of custom patterns. Only the custom patterns are editable.

Click one of the patterns on the left to change a custom pattern. Edit the current pattern by clicking the magnified image of the pattern. Click a blank pixel to fill it and click a filled pixel to make it blank.

Right/Left arrows move the pattern one pixel to the right or to the left.

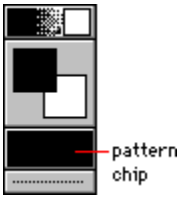
Up/Down arrows move the pattern up or down one pixel.

Black/White square reverses the colors of the pattern (for example, black becomes white and white becomes black).

Copy/Paste copies or pastes the pattern you want to use.

Tile Settings (Paint window)

Tiles are a useful way to create patterns with more than two colors. Cast members in the Paint window form the basis for creating a tile. When you choose a portion of a cast member to be made into a tile, the cast member becomes a building block for a pattern created with a field of tiles. You can use tiles in the Paint window or with the shapes in the Tool palette.

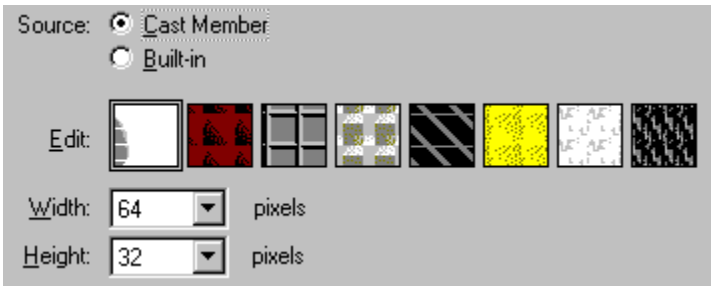


To open the Tile Settings dialog box, click the Pattern chip and choose Tile Settings from the pop-up.

Dialog box options

The Tile Settings dialog box lets you select the tile position, the cast member to make into a tile, what portion of the cast member to use for the tile, and the tile size.

Click a dialog box option for more information:



{button See also,AL(`Tile_settings_help')}

Source controls which cast member is the basis for your tile. Click Built In to restore the original tile to a tile position you've changed. Click Cast Member to make a tile from a cast member in your movie. Use the left and right arrows to step through all the graphic cast members in the movie.

The selection rectangle determines which part of the cast member is used for the tile. Drag the rectangle to reposition it on a different part of the cast member or click a new spot in the cast member view to reposition the rectangle.

Edit lets you select the tile you want to edit. An enlarged version of the tile is displayed in the dialog box.

Width Height lists the available tile sizes in pixels. You can make a tile as small as 16 by 16 pixels or as large as 128 by 128 pixels. As the size of the tile increases, more of the cast member is used for the tile.

Line width selector (Paint window)

The line width selector controls the thickness of the line drawn by the Line or Arc tool and the thickness of the borders drawn by the Shape tools.

The width of the line drawn by the Line, Arc, Rectangle, Ellipse, and Polygon tools can be changed with the line width selector. The line width palette has several settings for line width ranging from No Line (the dotted line in the line width selector) to Other. Use the dotted line setting when you want to draw filled shapes without borders. If you choose Other, the line width is determined by the Other Line Width setting in the **Paint Preferences** dialog box.

Tip: Double-click Other Line Width at the bottom of the line width selector to open the Paint Preferences dialog box. Use it to set the other line width.

Color Depth indicator (Paint window)

The Color Depth indicator displays the color depth of the current cast member in the Paint window.

Double-click the Color Depth indicator to open the **Transform Bitmap** dialog box. Use the Transform Bitmap dialog box to change the color depth of the current cast member in the Paint window. Changing the color depth from color to black and white saves disk space. You can still make a selected 1-bit sprite a color other than black by selecting colors with the foreground and Background color chips in the Tool Palette after it has been reduced to 1-bit. (This colorizes the sprite on the Stage, but does not affect the original cast member, which remains black and white.)

If you import black and white cast members, changing their color depth to multiple colors permits you to colorize them with any color in the current palette.

Flip Horizontal

Flip Horizontal mirrors the selected area in the Paint window horizontally from right to left.

Flip Vertical

Flip Vertical mirrors the selected area in the Paint window vertically from top to bottom.

Rotate Left

Rotate Left rotates the selected area in the Paint window 90 degrees counterclockwise.

Rotate Right

Rotate Right rotates the selected area in the Paint window 90 degrees clockwise.

Free Rotate

When you choose Free Rotate, handles appear at the corners of the Marquee. Drag any handle in the desired direction to rotate the object in the Paint window any number of degrees clockwise or counterclockwise.

Skew

The Skew command skews the selected artwork. When you choose Skew, handles appear at the corners of the Marquee. Dragging a handle in the desired direction moves the opposing corner an equal amount in the same direction, maintaining a parallelogram.

Warp

When you choose Warp, handles appear at the corners of the selection rectangle. Each corner can be dragged in any direction independent of the other corners. When you release the mouse button, the selected artwork assumes the shape that you have created.

Perspective

Perspective stretches the selected artwork to give it a perspective effect. When you choose Perspective, handles appear at the corners of the selection rectangle. Drag one or more handles to create the effect you want. For example, you can bring the two top handles closer together to create the illusion of linear perspective.

Tip: To make artwork appear to be vanishing into the distance, choose Perspective and move the handles on one side of your selection together and the handles on the opposite side of your selection apart.

Smooth

Smooth softens the edges of the selected artwork by adding pixels of blended color to the artwork's edges.

Trace Edges

Trace Edges creates an outline around the edges of the selected artwork. The outline is the same color as the selected line, if the line is a solid color. If the original line is multicolored, an outline is created for each section of the line. Click Trace Edges repeatedly to add multiple outlines.

Invert

The Invert command reverses the colors of the selected area in the Paint window. For 2-, 4-, and 8-bit cast members, to see what the reverse colors are, open the Color Palettes window, select all the colors in the palette, and click Invert Selection in the Color Palettes window; you'll see that the effect is an upside-down mirror image of the palette. For 16- and 32-bit cast members, a true RGB-complement of each color is shown. If you are working in black and white, Invert changes black to white and vice versa.

Lighten

Lighten increases the brightness of anything in the selection rectangle

Darken

Darken reduces the brightness of the selected artwork.

Fill

The Fill command fills a selected area with the current foreground color and pattern.

Switch Colors

Switch Colors changes each pixel that is the currently selected foreground color to the currently selected background color.

Note: This command only works for images whose color depth is 8 bits or less.

Text window (Window menu) Control-6

Use the Text window to create and edit text cast members. Enter and edit text in the Text window using standard word processing procedures. You select text in the Text window or on the Stage by dragging across the text, or by double-clicking to select a whole word. Triple-clicking selects all text in the cast member, which is the same as choosing Select All from the Edit menu.

The ruler and toolbar across the top of the Text window provide several formatting shortcuts.

To open the Text window:

- Choose Text from the Window menu.
- Click the Text window tool on the toolbar.
- Double-click a text cast member in the cast or on the Stage.

Click a topic for more information:

Creating text cast members

Working with text

Importing text

Text Ruler

Text Inspector

Formatting text

Formatting paragraphs

Paragraph command (Modify menu)

Font command (Modify menu)

Editable text (Field window)

Tip: To create another view of the same Text window, choose Window > New Window. This is useful if you are editing a large text cast member, since you can display different sections of the text in each view and cut and paste between them.

Text Ruler

Use the text ruler to set tabs and indents for paragraphs. To show or hide the text ruler, choose View > Ruler. To set units of measure for the ruler, choose File > **General Preferences**.

Setting tabs

- To set tabs, click the Tab well until the symbol for the type of tab you want and then click the Ruler where you want to place the tab.

Left
Right
Center
Decimal

- To remove a tab, drag the tab off the Ruler.
- To move a tab, drag the tab to the new location on the Ruler.
- If you don't define your own tab stops, pressing the Tab key advances the cursor to the next preset tab.

Setting indents

To set an indent for selected paragraphs, drag the left and right indent markers on the ruler. To change the indent for only the first line of text, drag the marker. Setting a special first line indent is useful for creating bulleted paragraphs and hanging indents.

Field window (Window menu) Control-8

Use the Field window to enter and edit text for field cast members. It is identical to the Text window except that the Ruler is not available for setting tabs and indents. When you use the alignment and spacing tools on the toolbar, the changes affect all the paragraphs in the cast member.

To open the Field window:

- Choose Field from the Window menu.
- Or double-click a field cast member.

{button See also,AL(`Text_Field')}

Color Palettes window (Window menu) Control-Alt-7

The Color Palettes window provides several ways of changing Color Palettes.

Click a topic name for more information:

Reserve selected colors

Select reserved colors

Select used colors

Invert selection

Blend Colors

Reverse sequence

Cycle colors

Sort colors

Color picker

All the functions in the Color Palettes window involve changing the currently active color palette. You choose the active palette by selecting a palette from the pop-up.

Director has ten built-in palettes:

- System-Mac (the standard 256-color Macintosh system palette)
- System-Win (the standard 256-color Windows system palette)
- Rainbow palette
- Grayscale palette
- Pastels palette
- Vivid palette
- NTSC palette
- Metallic palette
- VGA palette, a special palette for VGA 4-bit displays. It provides consistent results when playing Director movies under Windows in 4-bit mode.
- System-Win (Dir4), a palette that is included for compatibility with movies created in Director 4.

If you add new palettes to your movie from other graphics applications, those palettes also appear in the pop-up and in the cast.

In Windows, if you use any palette other than the two System-Win palettes (or a palette that does not include the Windows colors in its top and bottom rows), Director switches to Classic user interface look, and switches Windows itself to black and white. This is to maintain legibility while editing your movies.

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 **Movie Properties** on the Modify menu to choose the movie's default color palette, or **Frame Palette** on the Modify menu to change the color palette at a particular frame.

 Use the Hand tool to drag colors in the palette to reposition them.

 Use the Eyedropper tool to match the color of any pixel on the Stage with the same color in the palette. Click the Eyedropper tool and select any color in the Color Palettes window. Without releasing the mouse button, drag to any point on the Stage. The selection in the Color Palettes window and the foreground color in the Tool Palette change to the color at the pointer location.

 Clicking the arrow changes your pointer back to the arrow pointer.

{button See also,AL(`Color_Palettes')}

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Reserve Selected Colors button

Reserve Selected Colors button isolates specific colors used in palette effects like color cycling. For example, if you are cycling colors and don't want to inadvertently use the cycling colors on a noncycling cast member, reserve the cycling colors to prevent them from being used.

To reserve colors, select them in the Color Palettes window and then click the Reserve Selected Colors button.

When the Reserved Colors dialog box appears, choose Reserve Selected Colors and then click Reserved.

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Reserved colors appear striped in the Color Palettes window. The Select Reserved Colors button (shown at the left) also appears in the window to help you see which colors are reserved.

Click the Select Reserved Colors button to select all reserved colors.

To make all the reserved colors available again, click Reserve Selected Colors and choose All Colors Available in the dialog box.

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Select Reserved Colors

Select Reserved Colors in the Color Palettes window highlights the colors in the current palette that have been reserved.

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Select Used Colors button

The Select Used Colors button in the Color Palettes window highlights the colors in the current palette that are used in selected cast members.

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Invert selection button

The Invert Selection button in the Color Palettes window replaces the color or range of colors you selected with a new selection. The new selection consists of all the colors that were not part of your original selection.

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Sort button

The Sort button in the Color Palettes window reorders the selected colors in the palette by hue, saturation, or brightness.

To use the Sort button, select a range of colors in the Color Palettes window and then click the button. When the Sort Colors dialog box appears, choose Hue, Saturation, or Brightness. When you click Sort, the Create Palette dialog box appears. Enter a name for the new palette in the Name field. After you click OK, the colors appear in the palette according to the sorting order selected.

If you sort colors after drawing cast members, the cast members that use the selected colors will also change color as the colors are sorted.

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Reverse Sequence button

The Reverse Sequence button in the Color Palettes window reverses the order of the selected colors: the first color of the palette becomes the last. The colors themselves remain unchanged.

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Cycle button

The Cycle button displaces all the selected colors in the Color Palettes window one square to the left. The leftmost color wraps around and appears at the last right square. Each time you click the Cycle button, the selected colors shift by one more square. As the colors reach the left edge of the selection, they wrap around to the right edge and continue their journey. It is similar to the movement of color within a palette that you can see while colors cycle.

This is precisely what goes on when you use color cycling. If you are using a paint tool in the Paint window with a cycling ink effect, the colors rotate through the palette as you draw. Another example of color cycling is in the **Frame Palette** dialog box in the Modify menu. You can select a range of colors to cycle as your movie plays. Any cast member that is the same color as one of the cycled colors will change as the colors rotate through the palette.

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Blend button Control-B

The Blend button changes the current palette to create a blend of the first and last colors of a selected range in the Color Palettes window. Use it to create blends of color for color cycling or smooth gradients.

To use the button, select a range of colors in the Color Palettes window and then click the button.

If you choose one of the nine built-in palettes, Director creates a new palette. You are prompted to name the new palette.

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Color Picker button

You can define a new color in the current color palette either with the controls at the bottom of the Color Palettes window, or with the Windows Color dialog box.

To use the Windows Color dialog box, select the color you want to change and then click the color picker button. For information about using the Color dialog box, see the user's guide that came with Windows.

To edit selected colors in the Color Palettes window using the HSB (Hue, Saturation, Brightness) system, click the arrows at the bottom of the window to increase or decrease the hue, saturation, or brightness.

Hue is the primary or secondary color created by mixing two primaries. **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 even 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 the brightness is reduced, the color gets darker as if more black were added. If brightness is reduced to 0, then no matter what the values for hue or saturation, the color will be black.

Video window (Window menu) Control-9

The Video window lets you play digital video movies. Use the controls at the bottom of the window to play, stop, advance, or rewind the movie. Stop the movie to cut, copy, and paste frames from the movie into another Video window.

Choose Window > Video, or double-click a digital video cast member in the Cast window or on the Stage to open the Video window.

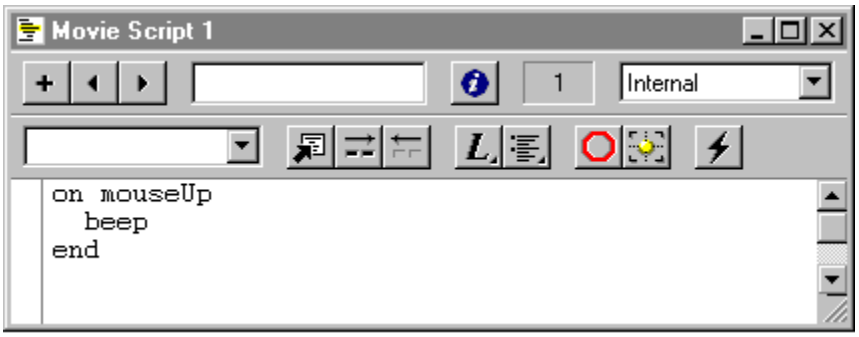
Note: On a Macintosh, the term "digital video" refers only to QuickTime movies. However, Director for Windows supports Microsoft's Video for Windows (.AVI) and QuickTime for Windows.

The **Previous Cast Member**, **Next Cast Member, Drag Cast Member, Cast Member Script, and Cast Member Properties** buttons work the same as they do in the **Cast window**.

{button See also,AL(`Video')}

Script window (Window menu) Control-0

Use the Script window to enter and edit scripts. A script can contain up to 32K of text.



For a description of the buttons at the top of the Script window, see the **Cast window** topic.
For descriptions of the tools on the Script toolbar, see the **Script toolbar** topic.

Tip: The help topics for **Lingo** commands include examples that you can paste into your scripts and use.

Director saves changes you make in the Script window when you click anywhere outside of the window, close it, click the Previous or Next buttons to go to a different script, or if you choose Control > Recompile All Scripts.

Tip: Double-click a cell in the script channel to open the Script window.

{button See also,AL(`Script')}

Script window toolbar

The Script window toolbar has the following tools (click one for more information):

Go to Handler pop-up

Lists the name of each handler that is used in the current movie. Go to a handler by selecting its name from the pop-up.

Go to Handler button

Goes to the handler that is in the current line of Lingo and inserts the cursor there.

Comment

If the current line of Lingo is not commented out, this command comments out the line by putting two dashes at the beginning of the line.

Uncomment

If the current line of Lingo is commented out, this command uncomments the line by removing the two dashes at the beginning of the line.

Alphabetical Lingo pop-up

Displays an alphabetical menu of all Lingo elements. Choosing one of the elements from the menu inserts it in the script at the cursor.

Categorized Lingo pop-up

Displays a pop-up of Lingo elements grouped according to the features that they can implement. Choosing one of the elements from the menu inserts it in the script at the cursor.

Toggle Breakpoint

Inserts and removes breakpoints in the current line of Lingo.

- When the current line of Lingo has a breakpoint, clicking Toggle Breakpoint removes it.
- When the current line of Lingo has no breakpoint, clicking Toggle Breakpoint inserts one.

Watch Expression

Adds the selected expression or variable to the list in the Watcher window.

Recompile All Scripts

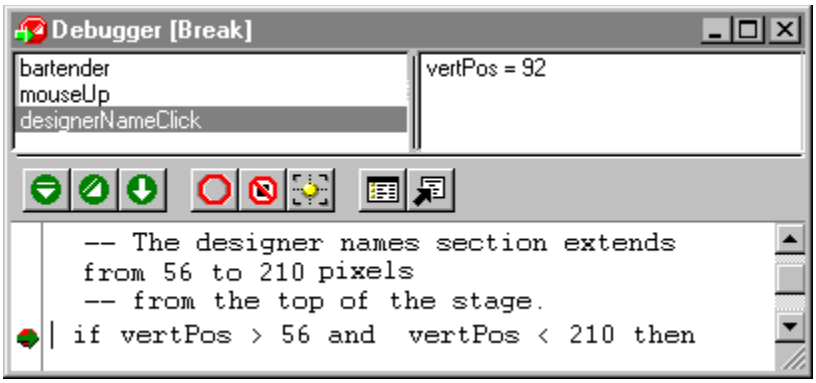
Recompiles the movie's scripts without closing the Script window.

Debugger window (Window menu) Control-`

The Debugger helps with troubleshooting. The window helps to locate and correct bugs in scripts. It includes several tools that let you:

- See the current line of Lingo
- Run the current handler line by line
- Track the sequence of handlers that were called as part of getting to the current handler
- Display the value of any local variable, global variable, or property related to the Lingo that you're investigating
- Open related windows such as the Watcher window and Script window.

Click part of the illustration for more information:



Any of the following actions opens the Debugger window:

- Choosing Window > Debugger
- Encountering a breakpoint in a script
- Clicking Debugger in an error message that appears when Director encounters a syntax error in a script.

You can't edit the script directly in the Debugger; you must return to the Script window.

{button See also,AL(`Debugger')}

Handlers History pane displays the order of handlers that ran to get to the current script. Clicking a handler name displays the script. Using the Step Into or Step Over button always focuses the Debugger back to the bottom-most active handler in the pane.

Variable pane displays local variables, global variables, and property settings that Lingo encountered as it ran up to the current line. Only values are displayed; you can't change them directly in the Debugger. To edit Lingo, return to the Script window. To change any of these values while working with the Debugger, use a `set` statement in the Message window.

Script pane displays the current script and highlights the current line of Lingo. Breakpoints are indicated by a red dot to the left of the line of Lingo. The current line is indicated by a green arrow to the left of the line. The Lingo that appears in the script pane is only a display. You must return to the Script window to edit the script.

Step Script runs the current line of Lingo, runs any handlers that the line calls, and then stops at the next line in the current handler. This is useful when you are confident that handlers called from the current line are performing as expected and want to concentrate on Lingo in the current handler.

Step Into Script follows Lingo's normal flow, line by line from the current line through any nested handlers. Lingo advances one line each time you click Step Into Script. This is useful when you want to examine the handlers called from the current line of Lingo.

Run Script exits debugging and returns to the current handlers.

Toggle Breakpoint turns on and off the breakpoint in the line of Lingo that is highlighted in the script display pane. When the highlighted line has no breakpoint, clicking Toggle Breakpoint inserts one at that line. When the highlighted line has a breakpoint, clicking Toggle Breakpoint removes the breakpoint.

Ignore Breakpoints has Lingo pass over any breakpoints in the movie's scripts.

Watch Expression adds the selected variables or expressions to the Watcher window.

Watcher window opens the Watcher window, which displays the current value of variables that you select. For more information, see **Watcher window**.

Go to Handler goes to the handler in the Script window whose name is selected in the Debugger. After you are in the Script window, you can edit the script normally.

Watcher window (Window menu) Control-Shift-`

The Watcher window shows the values of simple expressions and variables in the movie's scripts.

The variable or expression appears at the left of the window followed by an equal sign (=) and the expression or variable's current value. If Director can't obtain the value of an expression or variable in the current context, the term "<void>" appears to the right of the equal sign. Director updates values in the Watcher window when the user steps through lines of a script while in debug mode or continuously while the movie plays. Some variables, such as `the time` or `the mouseH` are updated even while the movie is not playing.

Note: Watching too many variables can decrease Director's response noticeably. For example, the cursor may blink slowly or windows may resize sluggishly. Reducing the amount of data the Watcher window must continuously update will immediately improve Director's performance.

To add variables or expressions to the list in the Watcher window by selecting them in the Script window:

1. **Open a Script window in which the variable or expression appears.**
2. **Select the variable or expression.**
3. **Click the Watch Expression button**  **at the top of the Script window.**

   Director adds the selected variables or expressions to the list in the Watcher window and also displays those changes in the variable pane.

To change the value of a variable or expression:

1. **Select the value or expression in the Watcher window.**
2. **Type the new value in the field next to the Set button.**
3. **Click Set.**

To add variables or expressions to the list directly from the Watcher window:

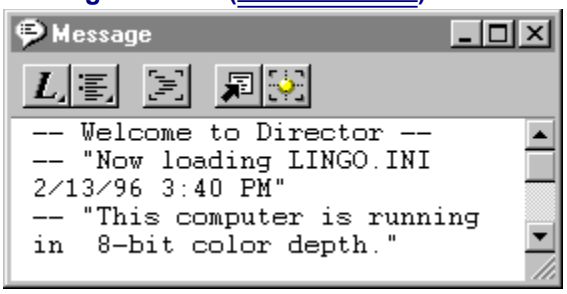
1. **Type the variable or expression in the field to the left of the Add button.**
2. **Click Add.**
   The variable or expression appears in the list.

To remove a variable or expression:

1. **Select the variable or expression in the Watcher window.**
2. **Click Remove.**

{button See also,AL(`Watcher')}

Message window (Window menu) Control-M



The Message window is a convenient place to experiment with and test Lingo scripts. Actions occur immediately when you press the Enter key, so you can see the results before you insert your scripts into a movie. This allows you to see the results of any script, including whether it is a valid script.

For descriptions of the tools on the Message window toolbar, see the **Message toolbar** topic.

To move around the Message window, use the arrow keys or scrollbar. Press Control-up arrow to move the insertion point to the top of the window. Press Control-down arrow to move the insertion point to the bottom of the window.

{button See also,AL(`Message')}

Message window toolbar

The Message window toolbar has the following tools (click one for more information):

Alphabetical Lingo pop-up

Displays an alphabetical pop-up of all Lingo elements. Choosing one of the elements from the menu inserts it in the script at the cursor.

Categorized Lingo pop-up

Displays a pop-up of Lingo elements grouped according to the features that they can implement. Choosing one of the elements from the menu inserts it in the script at the cursor.

Trace

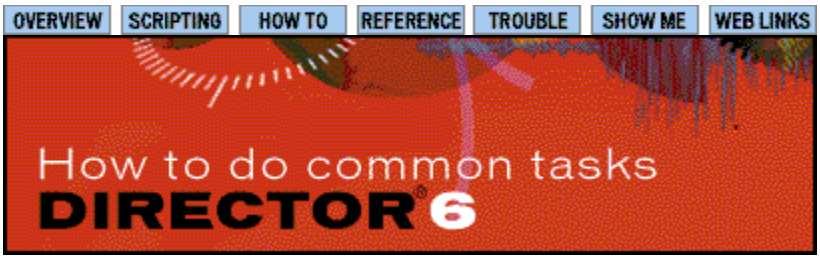
Click the Trace button to have the Message window display all the Lingo that runs as the movie plays. Using Trace slows down animation, so click Trace to turn it off.

Go to Handler

Goes to the handler that is in the current line of Lingo and inserts the cursor there.

Watch Expression

Adds the selected expression or variable to the list in the Watcher window.



Working with Cast Members-Common Tasks

Cast basics

{button ,JI(`',`UsersGuide_006_help')} Creating casts
{button ,AL(`howLink_2_3_help')} Working with the Cast window

Creating cast members

{button ,JI(`',`UsersGuide_012_help')} Importing cast members
{button ,JI(`',`UsersGuide_017_help')} Creating cast members within Director

Viewing cast member properties

{button ,AL(`howLink_2_8_help')} Basic task

Finding cast members

{button ,JI(`',`UsersGuide_022_help')} Basic task
{button ,JI(`',`UsersGuide_023_help')} Searching for cast members in the Score

Sorting cast members

{button ,JI(`',`UsersGuide_024_help')} Basic task

Working with Sprites-Common Tasks

Creating sprites

{button ,JI(`',`UsersGuide_032_help')} Basic task

Moving and resizing sprites

{button ,JI(`',`UsersGuide_034_help')} Basic task
{button ,AL(`howLink_3_2b_help')} Changing the stacking order of sprites

Selecting sprites

{button ,JI(`',`UsersGuide_036_help')} Basic task

Changing when sprites appear

{button ,JI(`',`UsersGuide_037_help')} Basic task
{button ,JI(`',`UsersGuide_038_help')} Moving sprites between frames
{button ,JI(`',`UsersGuide_039_help')} Changing sprite duration
{button ,JI(`',`UsersGuide_040_help')} Extending sprites

Viewing and changing sprite properties

{button ,JI(`',`UsersGuide_045_help')} Blending sprites
{button ,JI(`',`UsersGuide_046_help')} Scaling sprites

Tweening sprites

{button ,JI(`',`UsersGuide_048_help')} Basic task
{button ,AL(`howLink_3_12b_help')} Tweening options
{button ,JI(`',`UsersGuide_051_help')} Showing and editing sprite paths

Exchanging sprite cast members

{button ,JI(`',`UsersGuide_053_help')} Basic task

Editing sprite frames

{button ,JI(`',`UsersGuide_054_help')} Basic task

Animating with a series of cast members

{button ,JI(`',`UsersGuide_056_help')} Basic task
{button ,JI(`',`UsersGuide_058_help')} Creating a sprite from a sequence of cast members
{button ,JI(`',`UsersGuide_060_help')} Using Space to Time
{button ,JI(`',`UsersGuide_062_help')} Using film loops

Splitting and joining sprites

{button ,AL(`howLink_3_19_help')} Basic task

Step recording

{button ,JI(`',`UsersGuide_068_help')} Basic task

Real-time recording

{button ,JI(`',`UsersGuide_070_help')} Basic task

Linking a sequence with Paste Relative

{button ,JI(`',`UsersGuide_072_help')} Basic task

Aligning sprites

{button ,AL(`howLink_3_23_help')} Basic task

Adding and removing frames

{button ,AL(`howLink_3_24_help')} Basic task

Score viewing options

{button ,JI(`',`UsersGuide_079_help')} Zooming the Score

{button ,JI(`',`UsersGuide_083_help')} Using multiple Score windows

{button ,JI(`',`UsersGuide_084_help')} Turning a channel on and off

{button ,JI(`',`UsersGuide_085_help')} Showing and hiding the effects channels

{button ,JI(`',`UsersGuide_087_help')} Using Director 5 Score display

Creating Interactivity-Common Tasks

Attaching behaviors

{button ,AL(`howLink_4_2a_help')} Basic Tasks

{button ,JI(`',`UsersGuide_092_help')} Getting information about behaviors

{button ,JI(`',`UsersGuide_093_help')} Changing the order of attached behaviors

Creating and modifying behaviors

{button ,JI(`',`UsersGuide_101_help')} Basic Task

Using the button editor

{button ,AL(`howLink_4_10_help')} Creating and editing buttons

Editing Media-Common Tasks

Working with color depth and palettes

{button ,JI(`',`UsersGuide_118_help')} Changing the color depth and palette of bitmap cast members

Working with bitmaps

{button ,JI(`',`UsersGuide_124_help')} Resizing a bitmapped cast member
{button ,AL(`howLink_5_17_help')} Using the Paint window
{button ,AL(`howLink_5_18_help')} Using onion skinning
{button ,JI(`',`UsersGuide_139_help')} Using Auto Distort
{button ,AL(`howLink_5_19_help')} Using bitmap filters

Working with text

{button ,AL(`howLink_5_20_help')} Creating text cast members
{button ,AL(`howLink_5_23_help')} Formatting text

Using fields

{button ,JI(`',`UsersGuide_156_help')} Creating fields
{button ,JI(`',`UsersGuide_157_help')} Editing and formatting fields
{button ,JI(`',`Editing_Fontmap')} Editing a movie's Fontmap.txt

Working with digital video

{button ,JI(`',`UsersGuide_160_help')} Importing digital videos
{button ,JI(`',`UsersGuide_161')} Opening the Video window
{button ,JI(`',`UsersGuide_163')} Cropping a video

Using OLE cast members

{button ,JI(`',`UsersGuide_166')} Working with OLE cast members

Launching external editors

{button ,JI(`',`UsersGuide_170')} Basic task

Working Behind the Scenes-Common Tasks

Working with tempo settings

{button ,JI(`',`UsersGuide_180_help')} Using the tempo channel
{button ,JI(`',`UsersGuide_182_help')} Using sound and video cue points
{button ,JI(`',`UsersGuide_184_help')} Comparing actual speed with tempos you've set

Compressing and streaming sounds with Shockwave Audio

{button ,JI(`',`UsersGuide_195_help')} Compressing internal sounds
{button ,JI(`',`UsersGuide_196_help')} Streaming Shockwave Audio files

Working with transitions

{button ,JI(`',`UsersGuide_199_help')} Creating transitions

Changing color palettes

{button ,JI(`',`UsersGuide_203_help')} Changing palettes in a movie
{button ,JI(`',`UsersGuide_205_help')} Using color cycling
{button ,AL(`howLink_6_19_help')} Using the Color Palettes window

Completing Movies-Common Tasks

Creating projectors

{button ,JI(`',`UsersGuide_219a_help')} Basic task

Creating Shockwave movies

{button ,JI(`',`UsersGuide_220_help')} Basic task

Managing Xtras for distributed movies

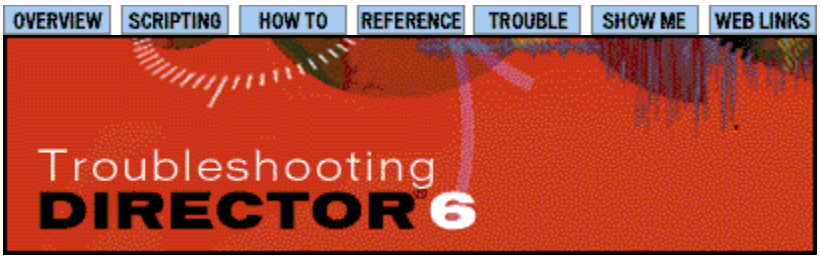
{button ,JI(`',`UsersGuide_223_help')} Making an Xtra package file

Processing movies with Update Movies

{button ,JI(`',`UsersGuide_226_help')} Converting existing movies

Exporting digital video

{button ,AL(`howLink_7_11a_help')} Basic tasks



Common problems

Tech support

Common problems

Click a category to see a list of frequently asked questions:

Technical support

Multiple platforms

General

Macintosh

Microsoft Windows

Digital video

Lingo

Made with Macromedia

Common problems - multiple platforms

{button ,JI(`',`FAQ_1')} How do I convert Director movies to work on a different type of computer? Do I need to purchase a new version of Director?

{button ,JI(`',`FAQ_2')} Will Director run on OS/2? How about Shockwave for Unix?

Common problems - general

{button ,JI(`',`FAQ_4')} Why won't my projector center on the stage when I choose this option in the Movie Properties dialog box?

{button ,JI(`',`FAQ_5')} When I import graphics into Director and apply Background Transparent ink effect to them, why aren't the edges clean?

{button ,JI(`',`FAQ_6')} Why are behaviors attached to frames ignored when I use wait settings in the tempo channel?

{button ,JI(`',`FAQ_7')} How can I expand Director's printing capabilities?

{button ,JI(`',`FAQ_8')} Why do I get an out of memory error when importing a FLC or FLI on Windows, or a PICS file on Macintosh?

{button ,JI(`',`FAQ_9')} How does Director work with a database?

{button ,JI(`',`FAQ_31')} How can I play a Director movie on a hard disk and keep its content on a CD-ROM?

Common problems - Macintosh

{button ,JI(`',`FAQ_11')} Why does Director quit with a Type 1 Error?

{button ,JI(`',`FAQ_12')} How do I make the area behind the stage black?

{button ,JI(`',`FAQ_13')} Why does my fileIO Lingo fail to work after upgrading a title from Director 4 or Director 5?

Common problems - Microsoft Windows

{button ,JI(`',`FAQ_14')} Why do 24-bit images look great in Director but look dithered when viewed in a Window 3.1 projector?

{button ,JI(`',`FAQ_15')} How do I play MPEG movies in Director for Windows?

{button ,JI(`',`FAQ_16')} How can I send MCI commands from Lingo? Where is a list of MCI commands?

{button ,JI(`',`FAQ_17')} When running a Director for Windows projector, why does the error message "Lingo.ini not found" occur?

{button ,JI(`',`FAQ_19')} When playing a QuickTime movie or an AVI file with sound, why am I unable to play another sound file simultaneously?

{button ,JI(`',`FAQ_20')} Is there a way to embed a custom icon for a Director for Windows projector?

{button ,JI(`',`FAQ_21')} Which installer is recommended for Director for Windows projectors? Which compiler is recommended for DLLs?

{button ,JI(`',`FAQ_30')} How do I find the letter of a CD drive in Director for Windows?

Common problems - digital video

{button ,JI(`',`FAQ_31a')} Why don't QuickTime movies play when I create a Windows 3.1 projector and test it Windows 95?

{button ,JI(`',`FAQ_22')} What happens to transitions and sounds when a Director movie is exported to digital video?

{button ,JI(`',`FAQ_23')} In Director for Windows, why does the controller of my QuickTime for Windows movie stay on the stage when I jump to a new frame? Or, on the Macintosh, why does the last frame of the digital video file stay on the screen when I jump to a new frame?

{button ,JI(`',`FAQ_24')} How can I make digital video files play as well as they do outside of Director?

Common problems - Lingo

{button ,JI(`',`FAQ_25')} How do I find the movie script?

{button ,JI(`',`FAQ_26')} How do I use a custom cursor in Director?

{button ,JI(`',`FAQ_27')} What is a mask cast member, and how do I make one?

{button ,JI(`',`FAQ_28')} I have a `rollOver` test in a frame that works properly, but when I jump to a new frame, that `rollOver` area is still being evaluated even though the sprite is no longer there. This also happens with the cursor of sprite property.

{button ,JI(`',`FAQ_29')} Where is a list of keyCodes?

Common problems - Made with Macromedia

{button ,JI(`',`FAQ_40')} Basic information regarding distribution of "Made with Macromedia" titles.

{button ,JI(`',`FAQ_41')} Who needs to comply with the Made with Macromedia logo requirements?

{button ,JI(`',`FAQ_42')} What if the Macromedia runtime is an insubstantial part of a commercially distributed software product that was not Made with Macromedia?

{button ,JI(`',`FAQ_43')} I am using Shockwave to add multimedia to my web site. Am I required to use the Made with Macromedia logo on my "Shocked" Web Site?

{button ,JI(`',`FAQ_44')} I qualify for either full or partial marking of the Made with Macromedia logo. What are the steps I need to take to comply with this agreement?

Technical support

Review the problem:

If you think you need technical support, first review the following steps. Most of the problems you encounter can be solved by following these steps:

1. **Read everything relevant to the problem in the manuals and the online help.**
2. **Check the index for more references to the topic.**
   More information on a procedure or feature may be found in a separate section.
3. **If something used to work, think about what may have changed.**
   Perhaps you installed new software or changed some settings.
4. **Create a new file and reproduce the problem there.**
   If the problem goes away in the new file, compare the new file with your old file to find and eliminate the differences.

Before you call:

If you still need help, consult the following checklist before contacting technical support. This advanced preparation will help the support representative pinpoint and solve your problem more quickly.

1. **Define the problem as a series of steps that can be repeated by the support representative.**
   The support representative needs to know exactly what the problem is.
2. **Provide the following information:**
   - Product name, version number, and product registration number
   - Type of computer, such as 386, 486 or Pentium, local-bus or non-local bus, Quadra or PowerMac
   - Amount of memory installed
   - Amount of free hard disk space
   - Screen resolution (screen size in pixels, for example, 1024 by 768)
   - Screen color depth (number of colors or bits, for example, 256 colors or 8-bit color)
   - Graphics card manufacturer, model name, and driver version number.
   - Sound card manufacturer and model name
   - DOS and Windows or Macintosh System version numbers
   - A list of external devices connected to the computer
   - Brief description of the problem or error, and the specific text of any error messages

Technical support:

Contacting Macromedia

Inside the U.S. and Canada

In Japan

In Europe

Contacting Macromedia

Technical Support

Sales: Call 800-288-4797

Source & Center

Call 800-396-0129 or 415-252-7999.

Contact Source & Center for training, consulting services, purchasing Priority Access technical support, referrals for multimedia development, referrals to Macromedia Authorized Graphics/Imaging Centers (MAGIC) and to user groups, and authorization programs for trainers, developers and service bureaus.

Macromedia International User Conference

Call 415-252-7999

Success Stories

pr@macromedia.com

fax 415-626-1502

Product Suggestions and Feedback

director@macromedia.com

fax 415-626-0554.

Contact the Director Product Team with product suggestions and feedback about Director.

World Wide Web

http://www.macromedia.com/

Made with Macromedia program

Call 415-252-2000.

Macromedia offers Director 5 developers the ability to distribute applications created in Director without paying royalties. Our new Macromedia licensing policy allows you to distribute your Director projects royalty-free, provided you include the Made with Macromedia logo as described in our guidelines.

Technical support inside the United States and Canada

Debugging, designing, creating

Technical support can answer installation, configuration, and general usage questions about the product. For help in debugging, designing, or creating your application, Macromedia can help you find a consultant. You can find an international list of Authorized Macromedia Developers at http://www.macromedia.com/support/developers/.

Internet

Information about technical support and Director can be found on Macromedia's website. Visit http://support.macromedia.com for more details.

Online services

Information about technical support and Director is available in Macromedia's forums on Compuserve and America Online. For an up-to-date list of all of the resources available online, call MacroFacts, Macromedia's 24-hour fax information line and request document 3198. In the United States and Canada, call 800-449-3329. From elsewhere, call 415-863-4409.

- **CompuServe:** To reach the Macromedia forum on CompuServe, use the command Go Macromedia. On CompuServe, we provide message areas for discussion of multimedia development and support of our products, as well as libraries that contain useful utilities and examples-including drivers, sample movies, Xtras, DLLs, and XCMDs.
- **America Online:** In the United States, to reach the Macromedia forum on America Online, use the keyword Macromedia. On America Online, we provide message areas for discussion of multimedia development and support of our products, as well as libraries that contain useful utilities and examples- including drivers, sample movies, Xtras, DLLs, and XCMDs.

MacroFacts

Macromedia's 24-hour fax information line, providing instant access to Macromedia's products and services. In the United States and Canada, call 800-449-3329. From elsewhere, call 415-863-4409.

Contacting Technical Support

Technical Support fax: 415-703-0924

Technical Support phone: 415-252-9080

Macromedia, Inc.

600 Townsend Street

San Francisco, CA 94103

Technical Support in Japan

Internet

Information about technical support and Director can be found on Macromedia's website. Visit http://support.macromedia.com for more details.

Online services

Information about technical support and Director is available in Macromedia's forums on Compuserve. For an up-to-date list of all of the resources available online, call MacroFacts, Macromedia's 24-hour fax information line and request document 3198. In the United States and Canada, call 800-449-3329. From elsewhere, call 415-863-4409.

CompuServe: To reach the Macromedia forum on CompuServe, use the command Go Macromedia. On CompuServe, we provide message areas for discussion of multimedia development and support of our products, as well as libraries that contain useful utilities and examples-including drivers, sample movies, Xtras, DLLs, and XCMDs.

MacroFacts

Macromedia's 24-hour fax information line, providing instant access to Macromedia's products and services. You can reach MacroFacts by calling 415-863-4409.

If you're looking for assistance by fax or phone internationally, please contact the vendor or the distributor from which you acquired Director.

For additional help, contact:

- **Macromedia Japan**
  Serom Building 3F
  Shinsen-Cho 11-7
  Shibuya-Ku
  Tokyo
  Japan 150

  81.3.3462.5790
  81.3.3462.5794 (fax)

Technical Support in Europe

Internet

Information about technical support and Director can be found on Macromedia's website. Visit http://support.macromedia.com for more details.

Online services

Information about technical support and Director is available in Macromedia's forums on Compuserve and America Online. For an up-to-date list of all of the resources available online, call MacroFacts, Macromedia's 24-hour fax information line and request document 3198. In the United States and Canada, call 800-449-3329. From elsewhere, call 415-863-4409.

CompuServe: To reach the Macromedia forum on CompuServe, use the command Go Macromedia. On CompuServe, we provide message areas for discussion of multimedia development and support of our products, as well as libraries that contain useful utilities and examples-including drivers, sample movies, Xtras, DLLs, and XCMDs.

For additional help, contact:

- **Macromedia Europe (including Europe, the Middle East, and Africa)**
  4 Wellington Business Park
  Dukes Ride, Crowthorne Berkshire
  England
  United Kingdom RG45 6LS

  44.1344.76.1111
  44.1344.76.1149 (fax)
  44.1344.750.517 (fax)

How do I convert Director movies to work on a different type of computer? Do I need to purchase a new version of Director?

Projectors are platform-specific. You need the Windows version of Director to create Windows projectors, and the Macintosh version to create Macintosh projectors.

In general, you do not need to duplicate your work. Director movies can be opened in the authoring environment on either platform. The Lingo code is usually the same. Most of the other issues you will encounter are content related: making sure that your cast members work on both platforms as well.

For the best results, work with both platforms from the beginning of your project. Test your project early, often, and on all target machines.

Will Director run on OS/2? How about Shockwave for Unix?

OS/2 is not a supported platform for Director 6 authoring or delivery. OS/2 is able to run many Windows applications, however Macromedia does not support this operating environment.

Macromedia offers the Director 4.04 Player for OS/2 which allows existing Director 4 developers to develop 32-bit OS/2 native applications. For information on this player contact Macromedia Sales.

There is no Shockwave plug-in for Unix systems, nor are there any announcements of one. For the up-to-date information, please visit our website at http://www.macromedia.com

Why won't my projector center on the stage when I choose this option in the Movie Properties dialog box?

To avoid this, use the following Lingo command in your `on startMovie` handler: "`set the centerStage to TRUE`".

When I import graphics into Director and apply the Background Transparent ink effect to them, why aren't the edges clean?

Imported images may have very light shades around the edges that look white to the eye, but are darker than true white. Background Transparent ink effect subtracts one color only. The default for this background color is white, the first indexed position of a palette. You can change the background color of a sprite by selecting the sprite on the stage and choosing a different color in the background chip of the Tools window.

Some techniques used to work with these light colors are:

- Use the Eyedropper tool in the Paint window to determine the position in the palette the light color. Select the image in the Paint window with the selection rectangle tool in Shrink or No Shrink modes. Select the Paintbrush tool and the switch ink effect to change that color to a true white.
- Images that are anti-aliased against a white background may get a halo around them. This is because the image has feathered edges with intermediate colors between the object color and the white background. To avoid the halo recreate the image without anti-aliasing, or edit the edges by hand.

Why are behaviors attached to frames ignored when I use wait settings in the tempo channel?

Wait settings in the tempo channel stop the Playback Head. Any script or behavior that waits for an `exitFrame` event will not work as long as the playback head is stopped.

For example, you have:

- a sprite script for a button to "`go to the frame + 1`"
- a Wait for Digital Video Movie to Finish in Channel: X set in the tempo channel

Director will not know which message to execute. The Tempo channel and Lingo compete for control of score navigation.

To avoid this problem and maintain interactivity, script everything in the Tempo channel with Lingo.

How can I expand Director's printing capabilities?

From the Director application, you can print for authoring purposes of previewing the stage, your scripts, cast members and more. For many reasons, one being to keep the size of the projector small, Director projectors have simple printing engines.

Use the Lingo `printFrom` command to print the stage in a projector at 25, 50 or 100%. You can also use third-party solutions to enhance the printing capabilities of a projector. Director 6 installs Printomatic Lite, an extra that will enhance printing cast members. For more information, see the **PrintOMatic Xtra** help topic, and consult the Printomatic documentation which is installed with Director 6.

PrintOMatic Xtra

PrintOMatic "Heavy" - The Full Version

The PrintOMatic Xtra is the full featured version of PrintOMatic Lite. It is a printing tool for Director. PrintOMatic adds a full set of page-layout, text and graphics printing features to Director 4, 5, and 6 projects on Macintosh and Windows.

PrintOMatic includes commands for specifying the position of any text or graphic element on the page. PrintOMatic documents can contain graphic files from disk, styled text, graphic primitives, cast member bitmaps, and snapshots of the Director stage.

Product Features

- Generates multi-page layouts with full control over the placement of text and graphic elements on the page.
- Print styled text: any combination of available fonts, sizes, or styles
- Print PICT, BMP or EPS files from disk, any portion of the Director stage, or Director cast members.
- Print object-oriented graphic primitives: lines, boxes, ovals and rounded rectangles.
- "Master Page" can contain any combination of text or graphic elements
- Automatic page numbering
- Customizable and hideable Print Progress dialog
- Paper-saving Print Preview feature
- Supports all Macintosh and Windows compatible printers
- Supports color and landscape-mode printing
- Fully compatible with MacOS, Windows 3.1, Windows 95 and Windows NT

Where to order PrintOMatic

The PrintOMatic Xtra is published and distributed worldwide by g/matter, inc. The registered edition of PrintOMatic is shipped with demonstrations, sample code, and other tidbits.

For more information, contact:

g/matter, inc.
300 Brannan Street, Suite 210
San Francisco, CA 94107

Tel: (800) 933-6223 or (415) 243-0394
FAX: (415) 243-0396

Email: sales@gmatter.com
http://www.gmatter.com/

Why do I get an out of memory error when importing a FLC or FLI on Windows, or a PICS file on Macintosh?

When you import a FLC/FLI or PICS file into Director, for example, a 27-frame file, it is similar to importing 27 full screen graphics. You may run out of memory if you try to import all of these graphics at once.

To avoid problems with memory, do the following:

1. **Import one small section of the file at a time.**
   PICS (Mac) or FLC/FLI (Windows).

2. **Save the file by choosing Save and Compact**
   This re-writes the file and your changes.

3. **Open About Director, and purge the memory.**

4. **Repeat this sequence as necessary.**
   How often you need to repeat the sequence depends on the size of the file.

Once the entire animation is imported, put it back together on the stage with "Cast To Time." Use the paint window to make sure the registration points of all of the images are the same. Double-click the registration tool to make sure it is in the center of each image.

For additional information about 3D animation see the technical support KnowledgeBase file, available on our CompuServe and America Online forums, and our web page.

How does Director work with a database?

Two database options are available:

- Create your own database structure using Lingo's lists. This is an easy, fast, cross-platform solution.
- Communicate with an external database using one of the Xtras available from third parties, such as FileFlex or V12.

  Issues you need to consider when using this approach include: cross-platform compatibility issues for the database, how to distribute the database, and the overhead of running multiple applications on the target machine.

  Note: Information on third party Xtras can be found on Macromedia's website.

Why does Director quit with a Type 1 Error?

A Type 1 Error is a bus error. This is one of the most common errors on the Macintosh. It could mean a number of things. If you get this error, review the following list:

1. **Allocate more memory to Director:**
   - Quit Director.
   - Select the Director application icon in the Director folder (do not launch it).
   - Select Get Info from the File Menu.
   - Allocate more memory to the Preferred size for Director.
   - Try the task you were doing before.

2. **If that does not work, disable all of your extensions:**
   - Restart your computer and hold down the Shift key.
   - Keep the Shift key held down until a message comes up saying "Extensions Disabled."

   If the error goes away, work through your extensions to see which ones might be conflicting. Sometimes two extensions that cause no problems by themselves will cause an error if they are both active at the same time. Similarly, sometimes extension conflicts are the result of load order

   Director does not require extensions, except QuickTime if you are using QuickTime movies. If you find an extension conflict on your machine, use the Extension Manager (System 7.5 and up) to disable all of your extensions except QuickTime while working in Director.

3. **If that does not work, reinstall Director.**

4. **If that does not work, reinstall the System Software.**
   A full list of all the Macintosh System errors is available from Apple.

How do I make the area behind the stage black?

To do this in a projector, choose Full Screen in the Projector Options dialog box, and change the color of the stage to black.

Why does my fileIO Lingo fail to work after upgrading a title from Director 4 or Director 5?

On Macintoshes the functionality of fileIO Xobject is no longer built into the Director or projector codebase. Director 5 introduced "fileIO Xtra", which is a newer MOA-compliant piece of code. This Xtra gets installed into the Director 6 Xtras folder. If your title relies on the older FileIO Xobject, you will need to distribute the Xobject with your projector. You can find a copy of the Xobject on your Director 6 CD.

Why do 24-bit images look great in Director but look dithered when viewed in a Window 3.1 projector?

Windows 3.1 projectors display bitmap sprites on stage in 8-bit color depth (256 colors) or less. This abides by a Microsoft standard set for Windows 3.1 to ensure a high level of compatibility with a broad range of video display drivers. Director 6 projectors running on Windows 95 and NT support 24-bit color.

This limitation does not apply to digital video, like QuickTime for Windows and Video for Windows. Digital video goes through a different display buffer than the other sprites on the Director stage. You have two options:

- Convert your bitmaps to one-frame digital video files, as they are optimized to color depths greater than 8-bit. However, because digital video files play directly to the screen buffer you will not be able to overlay sprites on top of them during playback.
- Dither your graphics to a custom 8-bit palette. This option maintains the score's 120 channel layering effects.

How do I play MPEG movies in Director for Windows?

There are third party Xtras which facilitate integrating MPEG with Director. A demo version of the MPEG Xtra is contained in the third party Xtras folder on the Director 6 CD.

Also, because Director 6 supports MCI, more experienced developers have the option to send MCI strings from Lingo. For more information on MCI, please see TechNote 3521.

How can I send MCI commands from Lingo? Where is a list of MCI commands?

Send MCI commands using the Lingo `mci "string"` command. The command you insert in the `"string"` area contains the device you are trying to control. See the **mci** help topic for information on using the Lingo `mci` command. Microsoft has a series of help files and technical documents on CompuServe and in their multimedia developer's kit. An MCI help file, called MCISTRWH.HLP, is located in our libraries on our CompuServe and America Online forums, as well as our web page: http:///www.macromedia.com.

When running a Director for Windows projector, why does the error message "Lingo.ini not found" occur?

The two most frequent causes of this problem include:

- A particular Cirrus Logic video driver that overwrites other areas of memory. Solution: Get newer Cirrus Logic video drivers from Cirrus Logic or use the generic Microsoft SVGA (640x480x256) driver.
- A multimedia shell is in use on a low-end machine, thus, reducing system resources and available RAM below a workable level. The solution is to run straight Program Manager instead.

When playing a QuickTime movie or an AVI file with sound, why am I unable to play another sound file simultaneously?

Windows supports only one sound channel. Director for Windows can play more than one sound at a time (.WAV or .AIF) because it mixes them together. Macromedia created a technology called MACROMIX.DLL that allows Director to do this.

What Director cannot do is mix sounds for QuickTime for Windows or AVI movies with audio files. Whichever sound gets to the sound port first, wins. The second sound cannot start playing until the first sound is completely finished with the sound channel.

The Director for Windows "README.WRI" file recommends keeping at least one frame between video and audio sources. Other techniques used to ensure the first sound is completely finished include:

- For a puppetSound: `puppetSound 0`
- For a digital video: `set the movieRate of sprite X to 0`

Is there a way to embed a custom icon for a Director for Windows projector?

To embed a custom icon in Windows, edit the icon inside the PROJECTR.SKL file.

1. **Make a backup copy of PROJECTOR.SKL.**
   It should be installed into the same directory as DIRECTOR.EXE.

2. **Open PROJECTOR.SKL as an .EXE using a resource editor**
   One recommended resource editor is AppStudio. AppStudio comes with Microsoft's Visual C++.

3. **Edit the icon.**
   The icon is located in the icon resource called APPICON.

4. **Save the file.**
   Make sure the file is still called PROJECTOR.SKL and that it is saved to the same directory.

5. **Create a projector with your own icon.**

6. **Optional: restore the original version of PROJECTOR.SKL.**

Notes:

- Borland Resource Workshop versions 4.0 and 4.0.2 will not work for this procedure. (It comes with Borland C++.) The Director projector skeleton includes half a megabyte of code, but BRW will not recognize those code elements - it only understands resources. Thus, you will run into "unexpected file format" errors trying to save your updated PROJECTR.SKL file as an EXE.
- You will need to use a 16-bit resource editor in order modify the resources of a 16-bit projector.

Which installer is recommended for Director for Windows projectors? Which compiler is recommended for DLLs?

There are many installer and compiler products on the market. We cannot, as a policy, officially recommend third-party software or hardware. It seems, from the feedback of many developers, that this decision is based on personal preference. For a list of installers, please see Tech Note #3528 at Macromedia's, or our forums on Compuserve and AOL.

What happens to transitions and sounds when a Director movie is exported to digital video?

You cannot export transitions in Director 6.

On Windows, the export to AVI option does not support sound.

On the Macintosh, exporting sound with the Director animation to QuickTime can vary in reliability. Success in doing this is affected by the sound format, available resources on the machine, where the sound is placed in the score and many other factors, none of which are constant.

The best technique is to export only the animations from Director. Add the transitions and sounds using a digital video editing tool. SoundEdit 16 works well when editing the sound of a QuickTime movie on a Macintosh. You can choose keyframes in the QuickTime movie, synchronize parts of the wave form to those frames, and save the QuickTime movie once again.

In Director for Windows, why does the controller of my QuickTime for Windows file stay on the stage when I jump to a new frame? Or, on the Macintosh, why does the last frame of the digital video file stay on the screen when I jump to a new frame?

In Windows, digital video plays directly to the screen buffer, on top of Director's compositing engine. You can turn this property off on the Macintosh, but not in Windows. It is a good technique on the Macintosh, however, to select Direct to Stage in the Cast Member Properties dialog box for the QuickTime movie. This way, the video can play on the top layer, above the other sprites.

When the video plays direct to stage, Director does not know that the movie is there, and therefore does not redraw the stage when jumping to a new frame. You will see an artifact of the video when you jump to a new frame if you do not remove it yourself.

There are many techniques you can use to redraw the stage. Here is one example:

```
on exitFrame
   set the visible of sprite X to FALSE
      --X is the channel where the video is placed
   updatestage
end
```

How can I make digital video files play as well as they do outside of Director?

If you are having trouble with the performance of a digital video file in Director, you should:

1. **Play the digital video file in Apple's MoviePlayer or the Windows Media Player.**
   Make sure it runs to your satisfaction outside of Director.

2. **After importing the file into Director, play it in the video window.**
   You can access this window by double-clicking on the digital video file in the Cast window. If the file plays poorly from there, it will not play well in the score. If you are having troubles with the file, you should return to the MoviePlayer or Media Player to see if the file is intact.

3. **On the Macintosh, select Direct to Stage in the Digital Video Cast Member Info dialog box for the QuickTime movie.**
   Note: Digital video is always Direct to Stage on Windows.

4. **Place the video in the score.**
   - In the script channel for that frame, type the following commands. It is important to use this "`go to the frame`" loop to keep Director's playback head moving in that frame:

     ```
     on exitFrame
        go to the frame
     end
     ```

   - If you would like to loop in that frame until the digital video is done playing, you can use this script:

     ```
     on exitFrame
        if the movieRate of sprite channelNumber
        then go to the frame
     end
     ```

For a list of all of the digital video Lingo terms, see the **Lingo digital video** topic.

How do I find the movie script?

Unlike previous versions, Director 5 and Director 6 support multiple movie scripts. Once created, you can find a movie script in the cast window. Additional ways to locate a movie script include:

- Press Ctrl-Shift-U (Windows) or Cmd-Shift-U (Macintosh)
- In the Window menu, select the script window.
- If you're in a score script, click the Add (+) button.

How do I use a custom cursor in Director?

To use a custom cursor, follow these steps:

1. **The cast member you use for your custom cursor must be 1-bit (black and white.**
   You can verify this in the bottom left hand corner of the paint window.

2. **Your Lingo syntax needs to be correct. You have two options of how to do this. Here are some sample scripts to illustrate this:**

   You can use the "`cursor`" command:

```
on startMovie
  cursor [5]
  --your cursor is in cast #5 and would be
  --active for the entire movie
end
```

   You can use the "`cursor of sprite`" command:

```
on enterFrame
  if rollover (2) then
    --the sprite you rollover is in channel 2
    set the cursor of sprite 2 to [5]
    --your cursor is in cast #5
  end if
    --only when sprite 2 is rolled over
end
on exitFrame
  go to the frame
end
```

Note: For additional information refer to Director's *Learning Lingo* manual.

What is a mask cast member, and how do I make one?

When you use a custom cursor, the areas that are white in the black-and-white area will be transparent when rolling over other sprites. To make the white areas opaque, you need to make a mask cast member. The Macintosh system watch cursor is a good example of this.

Here is one technique to do so:

1. **Duplicate your custom cursor cast member in the cast window.**
2. **Double-click that cast member and bring it up in the paint window.**
3. **After zooming in, select the pencil tool and draw a one-pixel circumference around the bitmap.**
   Note: Make the circumference one pixel thicker than the other bitmap.
4. **Take the paint bucket tool and fill the white areas with black.**
   The paint bucket tool may do the trick. In order to have the white area be opaque over other sprites, there needs to be opposition of black and white pixels when the two cursors are used together.
5. **Make sure this cursor is set to 1-bit after editing.**
6. **Make sure this cast member is in the cast position following the custom cursor.**
7. **Add it to your syntax. For example:**

```
set the cursor of sprite 2 to [5,6]

--where the cursor cast member is in cast #5

--and the mask cast member is in cast #6
```

I have a `rollOver` test in a frame that works properly, but when I jump to a new frame, that `rollOver` area is still being evaluated even though the sprite is no longer there. This also happens with the `cursor of sprite` property.

These tests for the mouse position, or "hot" areas, actually refer to sprite channel characteristics, and not to individual sprites. If the channel used to hold the sprite being rolled over alternates between empty and full, unexpected results may occur.

Here are three solutions:

1. **Keep channels full in all frames.**
   You can move all the frames next to one another and delineate them with markers.

2. **Create a one-bit "dot" cast member using the tools window.**
   Put an instance of the "dot" in the empty frames of the sprite channel being used to test for the rollOver.

3. **Scoot the sprite off-stage,**
   Move the sprite above the menu bar to give it an off-screen position before removing it from the stage.

This issue is similar to setting the `cursor of sprite`, as mentioned in the Lingo Dictionary. The cursor property will stay in effect until you turn it off by setting the cursor to zero.

Where is a list of keyCodes?

The **keyCode** function returns the numerical code for the last key pressed. This keyboard code is the key's numerical value, not the ASCII value.

You can generate a list of keyCodes by creating a one-frame test movie with these scripts:

- In the movie script, type:

```
on startMovie
set the keyDownScript to "put the keyCode"
end
```

- In the frame script of frame 1, type:

```
on exitFrame
go to the frame
end
```

When you play the movie, leave the message window open and the keyCode for any key you type will appear in the message window. Make sure that the message window is not the active window, or the keys you press will not be evaluated correctly.

To test for keys in the numeric keypad, you need to test the keyCodes from a projector and `put the keyCode` into a text field. It is necessary to test for certain modifier keys specifically with Lingo (i.e. the `controlDown`, `the shiftDown`, etc.)

These keyCodes are standard on Macintosh keyboards, but might not be standard across IBM-compatible keyboards. We have not, however, come across a nonstandard keyboard in technical support.

How do I find the letter of a CD drive in Director for Windows?

Crossing drive volumes is more difficult in Windows than on the Mac. On the Mac, you can use the following script and any mounted volume with that name and path will work automatically:

```
play movie "MyCD:Data:MovieA"
```

On Windows, however, each user's machine could have the hard drive on a different drive letter. The path could be "`G:\MyCD\Data\MovieA`" or "`D:\MyCD\Data\MovieA`" or something else. For Windows, create a handler that locates the drive letter for the CD drive and call that handler from a script.

Script:

put CheckDrive("weirdfil.txt") into myCD

Note: The script passes the file named "weirdfil.txt" to the handler. If the file is located, the handler returns the letter name of the CD volume, followed by a colon.

Handler:

The handler assumes that you have a uniquely named file that you pass to the handler, here named "`weirdfil.txt`", at the root level of your CD drive. The handler successively searches for the file on the root of each drive letter in the alphabet (A to Z).

Note: This handler begins looking at drive C, since the chances of both A and B drive being floppy drives is very high:

```
on CheckDrive weirdfile
   repeat with i = 66 to 90
     set drive = numTochar( i )
     set thisPath = string(drive & ":\"& weirdfile)
     set myFile = new(xtra "fileio")
     openFile(myFile, thisPath, 1)    -- attempt to open the file
     if status(myfile) = 0 then       -- status returns 0 for success
        set myFile = 0                -- Dispose of the instance
        return drive&":"
        exit
     end if
  end repeat
  set myFile = 0                      -- Dispose of the instance
  alert "Please check that"&"E&weirdfile"E&&"is on your CD drive."
end
```

Why don't QuickTime movies play when I create a Windows 3.1 projector and test it on Windows 95?

You may have the 32-bit version of QuickTime installed. Director projectors for Windows 3.1 are 16-bit applications. 16-bit Director projectors will only be able to use 16-bit QuickTime. To obtain the most recent version of QuickTime visit Apple's website at http://quicktime.apple.com.

How can I play a Director movie on a hard disk and keep its content on a CD-ROM?

Here are some tips on how to play your Director movie from a hard disk while keeping its content on the CD-ROM:

1. **Make sure linked cast members are all on one volume, and save the movie onto the same volume.**
   This allows a relative pathname to be constructed at playback.

2. **In the movie script, determine the drive letter of the CD-ROM.**
   You may need to use a custom Xtra.

3. **Construct a string starting with the drive letter of the CD-ROM, and the rest of the path duplicating the directory hierarchy where the movie was saved.**
   This allows the relative paths constructed in the first step to remain usable.

4. **setAt the searchPath, 1, <<the constructed string>>**
   This tells Director to search for files on the CD-ROM, beginning at the named directory.

   If your content or subsidiary movies are on the hard disk and CD-ROM, search the hard disk first. These searches will be relative to the directory that the movie was launched from.

   If you know that your content and subsidiary movies are on the CD-ROM, you can dispense with searching the hard disk.

5. **set the searchCurrentFolder to FALSE**
   This bypasses searches relative to the current directory before scanning through the searchPath.

Basic information regarding distribution of "Made with Macromedia" titles.

Macromedia's royalty-free licensing policy means that you can distribute applications created with Director or Authorware to millions of end-users on multiple platforms-free. Simply include the Made with Macromedia logo on your product's packaging and credit screen, complete the Run time distribution agreement, and register your product with Macromedia to qualify.

This is meant to give a quick overview of the Made with Macromedia Run-time distribution agreement and answer frequently asked questions. It is not, however, a replacement for the actual agreement. Please refer to the Made with Macromedia (MwM) folder on your product CD for the Run-Time distribution agreement, logos, and logo usage guidelines.

Who needs to comply with the Made with Macromedia logo requirements?

Any user of Authorware or Director who creates an End-user Product and distributes it outside of his own organization or anyone who causes an End-user Product to be created and distributed outside of his own organization. You must fill out Exhibit A of the Run time distribution agreement and place the Made with Macromedia logo on the outer most front, side, or back of the packaging and within the software on either a splash or credits screen.

What if the Macromedia runtime is an insubstantial part of a commercially distributed software product that was not Made with Macromedia?

You do not need to place the Made with Macromedia logo on the outside of the packaging, but you will need to put the logo onscreen within the software that makes use of a Macromedia run-time. You must sign and return the Run-time distribution agreement, fill out Exhibit C, "Product Qualifying for Limited Markings," and place the Made with Macromedia logo on the splash or credits screen only. You do not have to place it on the packaging.

I am using Shockwave to add multimedia to my web site. Am I required to use the Made with Macromedia logo on my "Shocked" Web Site?

No. This is a benefit to help inform your viewers that your site contains cutting edge multimedia content. Viewers will be directed to the Shockwave Plug-In so they can view the Macromedia-created movies within your web site.

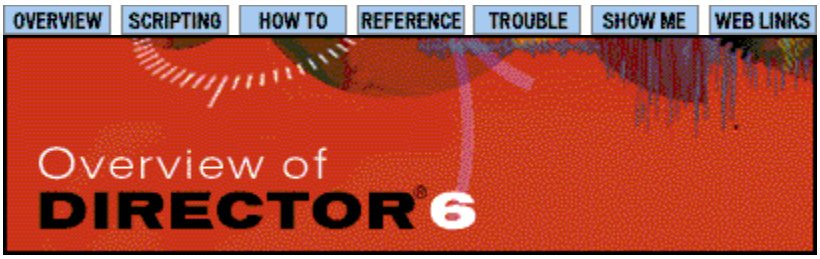
I qualify for either full or partial marking of the Made with Macromedia logo. What are the steps I need to take to comply with this agreement?

1. Complete, sign, and return one copy of the Run-time Distribution Agreement and either Exhibit A or C that are located in the Made with Macromedia folder of the Director or Authorware product CD. Exhibit A is for products that qualify for full marking and Exhibit C is for products that qualify for limited, software only, markings. The agreement becomes effective upon receipt by Macromedia. For multiple products or future products, you need only fill out and return an additional copy of Exhibit A or C. Each additional copy of Exhibit A or C will become effective upon receipt by Macromedia.
2. If you qualify for full markings, you need to place the Made with Macromedia logo on the outside of the packaging and on screen within the software. Logos are located in the Made with Macromedia (MwM) folder of your Authorware or Director product CD. See Exhibit B of the Run Time Distribution Agreement for detailed size and location guidelines. If you qualify for limited markings, you must place the Made with Macromedia logo on screen only within the software guidelines outlined in Exhibit B.
3. Incorporate the following copyright statement into the copyright screen of the end-user product.
   - If Authorware was used to create the Publisher Product:

     AUTHORWARE ¨ COPYRIGHT © 1993 Macromedia, Inc.
   - If Director was used to create the Publisher Product:

     DIRECTOR ¨ COPYRIGHT © 1994 Macromedia, Inc.
4. Send Macromedia two (2) copies of the final, packaged end-user software within 30 days of ship to:

   Macromedia Developer Relations
   600 Townsend Street
   San Francisco, CA 94103



What's new

Basic concepts

Ease-of-use improvements

Sprites

Sprites are now unified objects that span frames in the Score. To make a sprite span more frames, just drag the end frame. Tweening is automatic. Moving a sprite on the Stage moves it in every frame in which it appears.

For more information, see **Cast members and sprites**.

New Score

The Score has been redesigned so it's more powerful and easier to use. A new playback indicator displays in every channel. Useful information about selected sprites appears at the top of the Score. New views provide zoomed and expanded views of the movie. Open multiple Score windows to view the data in different ways at the same time.

For more information, see **Score viewing options**.

Key frames and automatic tweening

Key frames are selectable objects in the Score and on the Stage. Director automatically changes your animation when you move sprites in key frames.

Sprite paths

Display and adjust the path of a sprite moving over time right on the stage.

For more information, see **Showing and editing sprite paths**.

Sprite overlay

Display the most important sprite information directly on the Stage and easily open key editing windows with the sprite overlay.

For more information, see **Viewing and changing sprite properties**.

Sprite inspector

Display and edit the most important sprite properties in a floating palette with the Sprite Inspector.

For more information, see **Using the Sprite Inspector**.

Movies in Director Help

Director Help now includes a complete interactive tutorial and several movies demonstrating key concepts such as cast members and sprites, tweening, film loops, and drag-and-drop behaviors. Choose Help > Learning Director to run the interactive tutorial. See the **Show Me** help topic to see a list of all included movies.

Scriptless authoring

Drag-and-drop behaviors

Drag behaviors onto sprites and frames to create interactivity without scripting. Drag several behaviors onto a single sprite to build more complex actions. Choose from a large library of included behaviors, or create your own with Lingo or the Behavior Inspector.

For more information, see **Using included behaviors**.

Behavior Inspector

Create, combine, and edit flow-control and interactive behaviors with the Behavior Inspector. Enter parameters for existing behaviors, or make your own from scratch.

For more information see **Understanding behaviors**.

Button editor

Make buttons that show normal, pressed, rollover, and disabled states in a simple dialog box. No Lingo required.

For more information, see **Using the Button Editor**.

Internet improvements

Streaming playback

Movies begin playing as soon as the data for the first scene reaches the user's system. The beginning of the movie plays while the rest of the movie continues to download in the background.

For more information, see **Setting streaming playback options**.

Shockwave integration

Shockwave is now fully integrated with the Director authoring environment. Create movies ready for use with Shockwave without the AfterBurner Xtra. All internet Lingo functions are now recognized by the compiler and are fully documented in online help. Test Shockwave movies, including all net Lingo and internet functions, right in the Director authoring environment.

For more information, see **Creating Shockwave movies**.

Linked media on the internet

Link to media anywhere on the internet, just as you would link to external media on disk.

For more information, see **Importing from the internet**.

More net Lingo

Director's scripting language, Lingo, includes more commands for internet functions. Use new Lingo commands like frameReady, mediaReady, and getNetText to expand the power of Shockwave.

Browser integration

Use new scripting commands from the host environment to control Shockwave movies and more tightly integrate them with Internet Explorer, Netscape, and Visual Basic.

For more information, see **Browser scripts**.

General improvements

Media cue points

Synchronize the playback head to predefined cue points in sound or digital video.

For more information, see **Using sound and video cue points**.

Launch and edit

Launch any media editor directly from Director. Specify different editors for different graphic formats. Director automatically re-imports the data after editing in an external editor.

For more information, see **Launching external editors**.

New import formats

Both Windows and Macintosh versions of Director now support BMP, GIF, JPEG, LRG (xRes), Photoshop 3.0, MacPaint, PNG, TIF, and PICT graphic formats.

Director for Windows also supports Photo CD, PCX, WMF, PostScript, and the FLC and FLI multiple image formats.

Director for Macintosh also supports PICS, Scrapbook multiple image formats.

For more information, see **Importing cast members**.

Shockwave Audio

Shockwave Audio is now fully integrated with Director for use via the internet and off disk. Compress any internal sound by up to 176 to 1. Stream sounds directly from the internet or CD-ROMs.

For more information, see **Compressing and streaming sounds with Shockwave Audio**.

Shockwave compression

Shockwave compression is now usable off disk or over the internet. Compress projectors to make them use less disk space.

For more information see **Preparing a movie for distribution**.

Basic Concepts

Getting Started

Cast members and sprites

{button ,JI(`',`UsersGuide_001_help')} Basics

Using the Stage and Score

{button ,JI(`',`UsersGuide_002_help')} Basics

Controlling movie playback

{button ,JI(`',`UsersGuide_003a_help')} Basics

Working with Cast Members-Basic Concepts

Cast basics

{button ,JI(`',`UsersGuide_004_help')} Basics
{button ,JI(`',`UsersGuide_005_help')} Understanding internal and external casts
{button ,JI(`',`UsersGuide_007_help')} Working with the Cast window

Creating cast members

{button ,JI(`',`UsersGuide_017_help')} Basics
{button ,JI(`',`UsersGuide_012_help')} Importing cast members
{button ,JI(`',`UsersGuide_013_help')} Linking to a file
{button ,JI(`',`UsersGuide_014_help')} Importing from the internet
{button ,JI(`',`UsersGuide_015_help')} Supported file types
{button ,JI(`',`UsersGuide_018_help')} Using Xtra cast member types

Viewing cast member properties

{button ,JI(`',`UsersGuide_019_help')} Basics

Finding cast members

{button ,JI(`',`UsersGuide_022_help')} Basics

Sorting cast members

{button ,JI(`',`UsersGuide_024_help')} Basics

Using external casts

{button ,JI(`',`UsersGuide_025_help')} Basics

Working with Sprites-Basic Concepts

Creating sprites

{button ,JI(`',`UsersGuide_032_help')} Basics
{button ,JI(`',`UsersGuide_033_help')} Default sprite duration

Moving and resizing sprites

{button ,JI(`',`UsersGuide_034_help')} Basics
{button ,JI(`',`UsersGuide_035_help')} Changing the stacking order of sprites

Selecting sprites

{button ,JI(`',`UsersGuide_036_help')} Basics

Changing when sprites appear

{button ,JI(`',`UsersGuide_037_help')} Basics
{button ,JI(`',`UsersGuide_038_help')} Moving sprites between frames
{button ,JI(`',`UsersGuide_039_help')} Changing sprite duration
{button ,JI(`',`UsersGuide_040_help')} Extending sprites

Viewing and changing sprite properties

{button ,JI(`',`UsersGuide_041_help')} Basics
{button ,JI(`',`UsersGuide_042_help')} Using the Sprite Inspector
{button ,JI(`',`UsersGuide_043_help')} Sprite coordinates
{button ,JI(`',`UsersGuide_044_help')} Applying ink effects to sprites

Tweening sprites

{button ,JI(`',`UsersGuide_047_help')} Basics
{button ,JI(`',`UsersGuide_049a_help')} Tweening options
{button ,JI(`',`UsersGuide_050_help')} Tweening speed
{button ,JI(`',`UsersGuide_052_help')} Tweening suggestions and shortcuts

Exchanging sprite cast members

{button ,JI(`',`UsersGuide_053_help')} Basics

Editing sprite frames

{button ,JI(`',`UsersGuide_054_help')} Basics

Animating with a series of cast members

{button ,JI(`',`UsersGuide_055_help')} Basics
{button ,JI(`',`UsersGuide_057_help')} Animating with Cast to Time
{button ,JI(`',`UsersGuide_059_help')} Using Space to Time
{button ,JI(`',`UsersGuide_061_help')} Using film loops

Splitting and joining sprites

{button ,JI(`',`UsersGuide_065_help')} Basics

Step recording

{button ,JI(`',`UsersGuide_067_help')} Basics

Real-time recording

{button ,JI(`',`UsersGuide_069_help')} Basics

Aligning sprites

{button ,JI(`',`UsersGuide_073_help')} Basics

Selecting, moving and deleting frames in the Score

{button ,JI(`',`UsersGuide_075b_help')} Basics

Using markers

{button ,JI(`',`UsersGuide_076_help')} Basics

Score viewing options

{button ,JI(`',`UsersGuide_078_help')} Basics
{button ,JI(`',`UsersGuide_079_help')} Zooming the Score
{button ,JI(`',`UsersGuide_080_help')} Using sprite labels
{button ,JI(`',`UsersGuide_086_help')} Using Director 5 Score display

Creating Interactivity-Basic Concepts

Understanding behaviors

{button ,JI(`',`UsersGuide_089_help')} Basics

Attaching behaviors

{button ,JI(`',`UsersGuide_090a_help')} Basics
{button ,JI(`',`UsersGuide_092_help')} Getting information about behaviors
{button ,JI(`',`UsersGuide_091_help')} Entering behavior parameters

Using included behaviors

{button ,AL(`conceptLink_4_5a_help')} Basics

Creating and changing behaviors

{button ,JI(`',`UsersGuide_100_help')} Basics
{button ,AL(`conceptLink_4_7b_help')} Actions and events

Using the button editor

{button ,JI(`',`UsersGuide_106_help')} Basics

OVERVIEW | SCRIPTING | HOW TO | REFERENCE | TROUBLE | SHOW ME | WEB LINKS

Editing Media-Basic Concepts

Working with color

{button ,AL(`conceptLink_5_12_help')} Basics
{button ,JI(`',`UsersGuide_116_help')} Palettes in web browsers
{button ,AL(`conceptLink_5_14_help')} Solving color palette problems
{button ,JI(`',`UsersGuide_122_help')} Image resolution

Working with bitmaps

{button ,JI(`',`Paint_help')} Basics
{button ,JI(`',`UsersGuide_124_help')} Resizing bitmaps
{button ,JI(`',`UsersGuide_128_help')} Using registration points
{button ,AL(`conceptLink_5_18_help')} Using onion skinning
{button ,AL(`conceptLink_5_19_help')} Using bitmap filters

Working with text

{button ,JI(`',`UsersGuide_144_help')} Basics
{button ,JI(`',`UsersGuide_145_help')} Creating text cast members
{button ,JI(`',`UsersGuide_147_help')} Importing text
{button ,JI(`',`UsersGuide_150_help')} Using anti-aliased text
{button ,AL(`conceptLink_5_25_help')} Using different types of text in Director
{button ,AL(`conceptLink_5_26_help')} Creating fields
{button ,JI(`',`UsersGuide_158_help')} Mapping fonts between platforms

Working with digital video

{button ,JI(`',`UsersGuide_159a_help')} Basics
{button ,JI(`',`UsersGuide_159b_help')} Using Direct to Stage

Using shapes

{button ,JI(`',`UsersGuide_164_help')} Basics

Using OLE cast members

{button ,JI(`',`UsersGuide_165_help')} Basics
{button ,JI(`',`UsersGuide_167_help')} Working with OLE cast members
{button ,JI(`',`UsersGuide_168_help')} Using Paste Special with OLE objects

Launching external editors

{button ,JI(`',`UsersGuide_169_help')} Basics

Using movies within Director movies

{button ,JI(`',`UsersGuide_171_help')} Basics
{button ,JI(`',`UsersGuide_172_help')} Importing movies
{button ,JI(`',`UsersGuide_173_help')} Playing a movie in a window
{button ,JI(`',`UsersGuide_174_help')} Comparing film loops, digital videos, linked movies, and MIAWs

Working Behind the Scenes-Basic Concepts

Working with tempo settings

{button ,AL(`conceptLink_6_1_help')} Basics
{button ,JI(`',`UsersGuide_179_help')} Using the tempo channel
{button ,JI(`',`UsersGuide_181_help')} Using sound and video cue points

Using sounds

{button ,JI(`',`UsersGuide_185_help')} Basics
{button ,JI(`',`UsersGuide_186_help')} Importing sounds
{button ,JI(`',`UsersGuide_187_help')} Using internal and external sounds
{button ,JI(`',`UsersGuide_188_help')} Placing sounds in the Score
{button ,JI(`',`UsersGuide_189_help')} Repeating a sound
{button ,JI(`',`UsersGuide_190_help')} Setting sound volume
{button ,JI(`',`UsersGuide_191_help')} Synchronizing sound

Compressing and streaming sounds with Shockwave Audio

{button ,JI(`',`UsersGuide_192_help')} Basics
{button ,JI(`',`UsersGuide_193_help')} Understanding compression quality
{button ,JI(`',`UsersGuide_194_help')} Compressing internal sounds
{button ,JI(`',`UsersGuide_196_help')} Streaming external Shockwave Audio

Working with transitions

{button ,JI(`',`UsersGuide_197_help')} Basics
{button ,JI(`',`UsersGuide_198_help')} Creating transitions
{button ,JI(`',`UsersGuide_200_help')} Tips for using transitions

Changing color palettes

{button ,JI(`',`UsersGuide_201_help')} Basics
{button ,JI(`',`UsersGuide_202_help')} Changing palettes in a movie
{button ,AL(`conceptLink_6_19_help')} Using the Color Palettes window

Printing movies

{button ,JI(`',`UsersGuide_211_help')} Basics

Completing Movies-Basic Concepts

Preparing a movie for distribution

{button ,JI(`',`UsersGuide_213_help')} Basics
{button ,JI(`',`UsersGuide_214_help')} Distributing movies on disk
{button ,JI(`',`UsersGuide_215_help')} Distributing movies on the internet
{button ,JI(`',`UsersGuide_216_help')} Distributing movies on a local network
{button ,JI(`',`UsersGuide_217_help')} Organizing movie files

Creating projectors

{button ,JI(`',`UsersGuide_218_help')} Basics

Creating Shockwave movies

{button ,JI(`',`UsersGuide_219b_help')} Basics

Managing Xtras for distributed movies

{button ,JI(`',`UsersGuide_221_help')} Basics

Processing movies with Update Movies

{button ,JI(`',`UsersGuide_225_help')} Basics
{button ,JI(`',`UsersGuide_227_help')} Converting existing movies

Including required system elements

{button ,JI(`',`UsersGuide_228_help')} Basics
{button ,JI(`',`UsersGuide_229_help')} Video for Windows and QuickTime

Naming files and folders

{button ,JI(`',`UsersGuide_230_help')} Basics
{button ,AL(`conceptLink_7_10_help')} Choosing file names and folder names for Windows 3.1

Exporting digital video

{button ,JI(`',`UsersGuide_232_help')} Basics

Creating Shockwave Movies for the Web-Basic Concepts

Shockwave for Director

{button ,JI(`',`UsersGuide_246b_help')} Basics

Setting streaming options

{button ,JI(`',`UsersGuide_251_help')} Basics

Authoring issues for Shockwave movies

{button ,AL(`conceptLink_8_4_help')} Basics

Downloading considerations

{button ,JI(`',`UsersGuide_259_help')} Basics

Shockwave browser compatibility

{button ,JI(`',`UsersGuide_261_help')} Basics

Shockwave HTML requirements

{button ,JI(`',`UsersGuide_262_help')} Basics
{button ,JI(`',`UsersGuide_263_help')} Parameters for OBJECT and EMBED tags
{button ,JI(`',`UsersGuide_265_help')} Multiple movies in an HTML document

Using Xtras-Basic Concepts

{button ,JI(`',`UsersGuide_268_help')} Basics
{button ,JI(`',`UsersGuide_269_help')} Types of Xtras
{button ,JI(`',`UsersGuide_270_help')} Installing Xtras
{button ,JI(`',`UsersGuide_271_help')} Checking which Xtras are available

Using XCMDs and XFCNs

Note: For this development release of Director, this information in this help topic may not be up-to-date. It is based on facts for Director 5, and it will be updated soon.

Related topics:

Differences between Xobjects and XCMDs

Differences between XObjects and XCMDs

Learning to use XCMDs

Using an XCMD or XFCN

XCMDs and callbacks

XCMD and XFCN callback requests

Lingo lets you use HyperCard's XCMDs and XFCNs in your movies. Using XCMDGlue-part of Director's *Standard.xlib library of XObjects- you can access XCMDs and XFCNs from Lingo scripts. This lets you can extend Director's capabilities by using the many XCMDs and XFCNs available from HyperCard.

Most XCMDs and XFCNs work automatically with XCMDGlue, but some may not.When the XCMD's primary purpose is to perform a HyperCard-specific action-such as handling cards, HyperTalk scripts, or other parts of the HyperCard interface-the XCMD or XFCN might generate an error message when used in Director.

XCMDs and XFCNs are closely related. For convenience, Director Help refers to them collectively as XCMDs.

Note: XCMDs provide an interface to external code modules but are not capable of ensuring that the external code modules themselves perform as intended. You must make sure that the external code modules perform correctly to have them produce the desired results in Director.

Differences between XObjects and XCMDs

Note: For this development release of Director, this information in this help topic may not be up-to-date. It is based on facts for Director 5, and it will be updated soon.

XCMDGlue works differently from XObjects. You don't create instances of XCMDGlue to work with specific XCMDs. Instead, XCMDGlue acts as an interpreter between Lingo and the XCMD.

A major difference between XCMDs and XObjects is that an XObject can have multiple instances:

- One XObject can be used to create a number of independent objects, each capable of performing different operations.
- An XCMD cannot create new instances, so it can perform only one function at a time.

For these cases, you can use Lingo to create a special mechanism which may solve the problem. For further information, see the **XCMDs and callbacks** topic.

Learning to use XCMDs

Note: For this development release of Director, this information in this help topic may not be up-to-date. It is based on facts for Director 5, and it will be updated soon.

Related topics:

Opening XCMD resources

Viewing XCMD resources

Closing XCMD resources

Like using XObjects, using an XCMD involves three basic steps:

1. **Opening the XCMD**
2. **Exchanging messages with the XCMD to perform some function**
3. **Closing the XCMD.**

One of the best ways to learn about XCMDs is to use them in Director's message window. In this section, you'll see how to open, view the contents of an XCMD resource by exchanging a message with the XCMD, and close an XCMD.

Opening XCMD resources

Note: For this development release of Director, this information in this help topic may not be up-to-date. It is based on facts for Director 5, and it will be updated soon.

XCMDs can be located in two places: in an external file or in a Director movie.

When an XCMD resource is stored in the current movie's resource fork, the XCMD is automatically opened when the movie is opened. This is similar to the way *Standard.xlib is automatically opened when you launch Director. You can copy XCMD resources into your Director movie using a resource editor like ResEdit.

When an XCMD resource is stored in an external file such as a resource file or stack, you can open it with the openXlib command. If the file is in another folder, you must specify a full pathname to the folder. The easiest way to access the file is to place it in the same folder as your Director movie or the Director application.

To open an XCMD using the openXlib command:

1. **Launch Director.**
2. **Open the message window and type openXlib followed by the name of the XCMD resource file.**
3. **Press Return.**
   The resource file you specified opens.

   One resource file can contain multiple XCMDs. When you use the openXlib command, all XCMDs stored in the specified XCMD resource file are opened. The XCMD resource file can be a HyperCard stack, a resource file, or even a TeachText document containing XCMD resources. Notice that this is the same command used to open regular XObjects.

Viewing XCMD resources

Note: For this development release of Director, this information in this help topic may not be up-to-date. It is based on facts for Director 5, and it will be updated soon.

After you've opened the XCMD, you can use the showXlib command to display all open resource files that contain XCMDs as well as XObjects.

To display a list of all open resource files that contain XCMDs and Xobjects, ype showXlib in the message window, and then press Return.

To display the contents of a specific XCMD resource file, type showXlib followed by the name of the resource file, and then press Return.

Closing XCMD resources

Note: For this development release of Director, this information in this help topic may not be up-to-date. It is based on facts for Director 5, and it will be updated soon.

The closeXlib command lets you close all open resource files that contain XCMDs and XObjects.

To close all open resource files that contain XCMDs and XObjects, type closeXlib in the message window, and then press Return.

To close a specific resource file that contains XCMDs, type closeXlib followed by the name of the resource file in the message window, and then press Return.

Using an XCMD or XFCN

Note: For this development release of Director, this information in this help topic may not be up-to-date. It is based on facts for Director 5, and it will be updated soon.

In many cases, once you open an XCMD, you can use the XCMD in your Lingo scripts the same way you would use it in a HyperTalk script. XCMDGlue does everything else by converting the XCMD for you. For example, the following handler would let you use the MIDIplay XCMD (from Opcode Systems) to play a MIDI file from Director:

```
on startMIDIplayback

  openXlib (the pathname & "MIDIplay")

  -- opens the MIDIplay XCMD

  -- Use Lingo's "pathname" function to find

  -- resource files

  -- in the same folder as your movie

  MIDIplay "open","MyDrive:MyFolder:myMIDIfile"

  -- opens the MIDI file to be played

  MIDIplay "start"

  -- starts playback of the MIDI file

end startMIDIplayback
```

This handler stops the playback of the MIDI file:

```
on stopMIDIplayback

  MIDIplay "stop"

  closeXlib (the pathname & "MIDIplay")

end stopMIDIplayback
```

XCMDs and callbacks

Note: For this development release of Director, this information in this help topic may not be up-to-date. It is based on facts for Director 5, and it will be updated soon.

Related topics:

Using a callback handler

Defining the callback factory

Creating the callback object

Specifying the callback handler

Not all XCMDs can be used with XCMDGlue in a completely transparent manner. Occasionally, XCMDGlue is unable to properly convert the XCMD. When you attempt to use an XCMD's syntax in a script, an error message is displayed.

Certain XCMDs may call on HyperCard to internally perform some tasks while the XCMD is executing. Most of these are conversion routines and are used to conveniently convert information to and from different formats. The remaining callbacks either involve the HyperTalk interpreter or access information stored in HyperCard-specific entities such as fields, or they do both. The table of HyperCard callback requests at the end of this appendix lists specific technical information regarding these callbacks.

Lingo automatically supports all callbacks that are not overly specific to HyperCard. Still, some HyperCard-specific callbacks are supported when Lingo provides a direct equivalent. The remaining callbacks that are not automatically supported (a total of nine) are so specific to HyperCard that they cannot be resolved automatically unless the application calling the XCMD is virtually identical to HyperCard. Even in such cases, it is still possible to use an XCMD by using a user-defined mechanism called a callback handler.

Using a callback handler

Note: For this development release of Director, this information in this help topic may not be up-to-date. It is based on facts for Director 5, and it will be updated soon.

A callback handler uses a Lingo factory to accept and respond to messages that correspond to HyperCard callback requests. A factory is a set of scripts that can be used to create an object. In Director 4.0, the functionality of factories has largely been replaced by parent scripts. For more information on parent scripts, see Chapter 10, "Parent Scripts and Child Objects." In this specific case, however, a factory provides the best way to respond to callbacks. This section shows you the steps necessary to create a callback factory, and to call that factory from a handler.

Essentially, a callback handler provides a mechanism that some XCMDs already expect to be available. The XCMD expects that when it sends or receives a callback message, something will be there to receive it and possibly return another message. (Usually HyperCard does this.) A callback handler defined in Lingo simply intercepts and returns these messages when appropriate. Whether you choose to use this information depends on your understanding of the purpose of the callback.

Fortunately, when XCMDGlue does not understand a callback request, it indicates the name of the callback in the error message. Once you know which callback your XCMD needs to deal with, you can create a callback handler for it.

There are three basic steps to creating a callback handler:

1. **Defining a callback factory**
2. **Creating the callback object**
3. **Specifying the XCMD to be used with the callback object (with the setCallBack command that is part of XCMDGlue).**

Defining the callback factory

Note: For this development release of Director, this information in this help topic may not be up-to-date. It is based on facts for Director 5, and it will be updated soon.

The first step in creating a callback factory is to define it. The following example factory includes methods for all the callbacks that are not supported by XCMDGlue. This factory does not attempt to do anything with the callback requests other than create a record of them in the message window. As you'll see later, you can use this information to process callbacks. This factory should be placed in a movie script:

```
factory callBackFactory

method mNew

me(mPut, 1, "SendCardMessage")

me(mPut, 2, "EvalExpr")

me(mPut, 3, "StringLength")

me(mPut, 4, "StringMatch")

me(mPut, 5, "SendHCMessage")

me(mPut, 6, "ZeroBytes")

me(mPut, 7, "PasToZero")

me(mPut, 8, "ZeroToPas")

me(mPut, 9, "StrToLong")

me(mPut, 10, "StrToNum")

me(mPut, 11, "StrToBool")

me(mPut, 12, "StrToExt")

me(mPut, 13, "LongToStr")

me(mPut, 14, "NumToStr")

me(mPut, 15, "NumToHex")

me(mPut, 16, "BoolToStr")

me(mPut, 17, "ExtToStr")

me(mPut, 18, "GetGlobal")

me(mPut, 19, "SetGlobal")

me(mPut, 20, "GetFieldByName")

me(mPut, 21, "GetFieldByNum")

me(mPut, 22, "GetFieldByID")

me(mPut, 23, "SetFieldByName")

me(mPut, 24, "SetFieldByNum")

me(mPut, 25, "SetFieldByID")

me(mPut, 26, "StringEqual")

me(mPut, 27, "ReturnToPas")

me(mPut, 28, "ScanToReturn")
```

```
me(mPut, 31, "FormatScript")
me(mPut, 32, "ZeroTermHandle")
me(mPut, 33, "PrintTEHandle")
me(mPut, 34, "SendHCEvent")
me(mPut, 35, "HCWordBreakProc")
me(mPut, 36, "BeginXSound")
me(mPut, 37, "EndXSound")
me(mPut, 38, "RunHandler")
me(mPut, 39, "ScanToZero")
me(mPut, 40, "GetXResInfo")
me(mPut, 41, "GetFilePath")
me(mPut, 42, "FrontDocWindow")
me(mPut, 43, "PointToStr")
me(mPut, 44, "RectToStr")
me(mPut, 45, "StrToPoint")
me(mPut, 46, "StrToRect")
me(mPut, 47, "GetFieldTE")
me(mPut, 48, "SetFieldTE")
me(mPut, 49, "GetObjectName")
me(mPut, 50, "GetObjectScript")
me(mPut, 51, "SetObjectScript")
me(mPut, 52, "StackNameToNum")
me(mPut, 53, "Notify")
me(mPut, 54, "ShowHCAlert")
me(mPut, 100, "NewXWindow/GetNewXWindow")
me(mPut, 101, "CloseXWindow")
me(mPut, 102, "SetXWIdleTime")
me(mPut, 103, "XWHasInterruptCode")
me(mPut, 104, "RegisterXWMenu")
me(mPut, 105, "BeginXWEdit/EndXWEdit")
me(mPut, 106, "SaveXWScript")
me(mPut, 107, "GetCheckPoints")
me(mPut, 108, "SetCheckPoints")
me(mPut, 109, "XWAllowReEntrancy")
me(mPut, 110, "SendWindowMessage")
```

```
me(mPut, 111, "HideHCPalettes")
me(mPut, 112, "ShowHCPalettes")
me(mPut, 113, "XWAlwaysMoveHigh")
me(mPut, 200, "GoScript")
me(mPut, 201, "StepScript")
me(mPut, 202, "AbortScript")
me(mPut, 203, "CountHandlers")
me(mPut, 204, "GetHandlerInfo")
me(mPut, 205, "GetVarInfo")
me(mPut, 206, "SetVarValue")
me(mPut, 207, "GetStackCrawl")
me(mPut, 208, "TraceScript")
method mEvalExpr x
  put "mEvalExpr:" && x
method mSendHCMessage x
  put "mSendHCMessage:" && x

method mSendCardMessage x
  put "mSendCardMessage:" && x
method mGetFieldByName card, name
  put "mGetFieldByName:" && card && name
method mGetFieldByNum card, num
  put "mGetFieldByNum:" && card && num
method mGetFieldByID card, id
  put "mGetFieldByID:" && card && id
method mSetFieldByName card, name, value
  put "mSetFieldByName:" && card && name && value
method mSetFieldByNum card, num, value
  put "mSetFieldByNum:" && card && num && value
method mSetFieldByID card, id, value
   put "mSetFieldByID:" && card && id && value
method mUnknown which
  put me(mGet, value(which)) into callBackName
  put "mUnknown:" && which && "(" & Â
  callbackName & ")"
```

You do not need to specify every callback handled in this factory. You are required to define methods only for the callbacks that are indicated in error dialogs generated by the XCMD. For example, the mEvalExpr callback may be the only callback you need to account for.

As indicated in this example, the put statements in each method are optional. They are there to let you know what the XCMD or XFCN is attempting to tell HyperCard. You can use this information in any way you want. Sometimes, a callback requires a value (message) to be sent back to HyperCard. If you know what that value should be, use return at the end of the specific callback method's script. For example, if a callback required HyperCard to return TRUE or FALSE you could use a method similar to the following:

```
method callBackMethod

  if test then return TRUE else return FALSE

end callBackMethod
```

Some XCMDs use a large amount of processor time. In this situation, using a put statement in your script slows down whatever the XCMD does, because the put statement has to be evaluated and written into the message window. You can optimize the callback factory in this case by removing the put statements.

When a callback error occurs, the XCMD usually stops running after you click OK in the error dialog box. However, because of the design of certain XCMDs, the XCMD sometimes continues to execute. You still need to create a callback handler for these XCMDs. Otherwise, unexpected results could occur.

Creating the callback object

Note: For this development release of Director, this information in this help topic may not be up-to-date. It is based on facts for Director 5, and it will be updated soon.

After you have defined a callback factory, you can create a factory object using the following statement:

```
put callbackFactory(mNew) into callbackObject
```

Specifying the callback handler

Note: For this development release of Director, this information in this help topic may not be up-to-date. It is based on facts for Director 5, and it will be updated soon.

Finally, you specify the callback handler with the following statement:

```
setCallBack XCMD/XFCNname, callbackObject
```

The setCallBack command is part of the XCMDGlue XObject.

The XCMD or XFCN should now function properly. If you later use other elements of the XCMD's syntax, you might still need to deal with other callbacks. You can accomplish this easily by adding the appropriate method to your callback factory.

XCMD and XFCN callback requests

Note: For this development release of Director, this information in this help topic may not be up-to-date. It is based on facts for Director 5, and it will be updated soon.

The following are HyperCard's callback requests. The symbol in the rightmost column identifies which level of support is provided for each callback.

Number	HyperCard callback	Type*
1	SendCardMessage	-
2	EvalExpr	-
3	StringLength	4
4	StringMatch	4
5	SendHCMessage	-
6	ZeroBytes	4
7	PasToZero	4
8	ZeroToPas	4
9	StrToLong	4
10	StrToNum	4
11	StrToBool	4
12	StrToExt	4
13	LongToStr	4
14	NumToStr	4
15	NumToHex	4
16	BoolToStr	4
17	ExtToStr	4
18	GetGlobal	4
19	SetGlobal	4
20	GetFieldByName	-
21	GetFieldByNum	-
22	GetFieldByID	-
23	SetFieldByName	-
24	SetFieldByNum	-
25	SetFieldByID	-
26	StringEqual	4
27	ReturnToPas	4
28	ScanToReturn	4
31	FormatScript	-
32	ZeroTermHandle	-

33	PrintTEHandle	-
34	SendHCEvent	-
35	HCWordBreakProc	-
36	BeginXSound	-
37	EndXSound	-
38	RunHandler	-
39	ScanToZero	4
40	GetXResInfo	-
41	GetFilePath	-
42	FrontDocWindow	-
43	PointToStr	-
44	RectToStr	-
45	StrToPoint	-
46	StrToRect	-
47	GetFieldTE	-
48	SetFieldTE	-
49	GetObjectName	-
50	GetObjectScript	-
51	SetObjectScript	-
52	StackNameToNum	-
53	Notify	-
54	ShowHCAlert	-
100	NewXWindow/GetNewXWindow	-
101	CloseXWindow	-
102	SetXWIdleTime	-
103	XWHasInterruptCode	-
104	RegisterXWMenu	-
105	BeginXWEdit/EndXWEdit	-
106	SaveXWScript	-
107	GetCheckPoints	-
108	SetCheckPoints	-
109	XWAllowReEntrancy	-
110	SendWindowMessage	-
111	HideHCPalettes	-
112	ShowHCPalettes	-
113	XWAlwaysMoveHigh	-

200	GoScript	-
201	StepScript	-
202	AbortScript	-
203	CountHandlers	-
204	GetHandlerInfo	-
205	GetVarInfo	-
206	SetVarValue	-
207	GetStackCrawl	-
208	TraceScript	-

***** 4: Automatically supported by Lingo

-: Requires a callback handler. Some messages and expressions (such as EvalExpr) may be evaluated by XCMDGlue in a manner compatible with HyperTalk. Other messages and expressions (such as GetFieldByName) always assume HyperCard entities for which there are no counterpart in Director.

Editing a movie's Fontmap.txt file

To define the font mapping information for a movie, it's best to edit the Fontmap.txt file before you begin authoring a movie, since Director automatically uses the information stored in the Fontmap.txt file when you open a new movie. (If you've already created the movie, you can still edit the font map file, but you will then have to manually load the file into the movie to have Director apply it to the movie.)

To define the font mapping information for a new movie:

1 **Using any application that can edit text, open the sample Fontmap.txt file that's in the same folder as the Director application.**

When you installed Director, this file was placed in the same folder as the Director application. If the file is missing, you can either re-install it or create it from scratch. See the end of this section for an example of a Fontmap.txt file.

2 **For each Macintosh font remapping entry, type on one line:**

```
Mac:MacFontName=>Win:WinFontName Â
      [MAP (NONE|ALL)] [MACfontsize=>WINfontsize]
```

where MacFontName is the name of the Macintosh font, and WinFontName is the name of the Windows font being substituted for the Macintosh font.
The two arguments enclosed in brackets are optional. `MAP ALL` or `MAP NONE` specifies whether you want to remap characters with ASCII values greater than 127 or just pass them through. The default is `MAP ALL`.

You can specify how you want the characters to be remapped, as described in Step 3. The sample Fontmap.txt file contains mappings for a few commonly used graphical characters.

The last argument, `[MACfontsize=>WINfontsize]`, consists of one or more pairs of numbers, separated by a space, that let you map a Macintosh font size to a Windows font size.

Because font sizes appear smaller on a PC, you might want to map Macintosh font sizes to larger Windows font sizes.

3 **For each Macintosh special character that you want to remap, type:**

```
Mac:=>Win: OLDCHAR=>NEWCHAR OLDCHAR=>NEWCHAR ...
```

where OLDCHAR is the ASCII value of the Macintosh special character, and NEWCHAR is the ASCII value of the Windows character being substituted for it. You can enter as many remapping pairs as you want by separating each one with a space.
You can only remap characters whose ASCII values are greater than 127 and less than 255.

If you didn't specify MAP ALL for any of the font remapping entries, as described in Step 2, you can skip this step.

4 **Save the file as ASCII text, in the same folder as the Director application.**

5 **Open a new movie in Director.**

When you open a new movie, Director looks for the font map file named Fontmap.txt in the same folder as the Director application. All new movies will use the font mapping information in the Fontmap.txt file. You can edit this file on a movie-by-movie basis, as necessary.

Existing movies continue to use the font map information (if any) stored within the movie rather than the font mapping specified in the Fontmap.txt file.

To change the font mapping for an existing movie:

1 **Using any text editing application, edit the Fontmap.txt file as described in "Editing a movie's Fontmap.txt file," earlier in this section.**

Save this file using any name of your choice.

2 **Open the movie whose font mapping you want to change.**

3 **Choose Movie Properties on the Modify menu.**

4 **Click Load from File.**
This option lets Director load the font mapping assignments specified in the font map file.

5 **In the dialog box, select the font map file you just edited and click Open.**

6 **Click OK in the Movie Info dialog box.**

7 **Save the movie and close it.**

8 **Open the movie again.**

The movie now uses the font map information specified in the fontmap.txt file.

Note: If you edit a text, field, or button cast member on a Windows system, in a movie created on a Macintosh, the text cast member loses its original Macintosh font information. Similarly, if you edit a text, field, or button cast member on the Macintosh for a movie created on a Windows system, the cast member loses its original Windows font information. If you plan to edit a movie on both the Macintosh and Windows platforms, make sure that the font mapping file specifies that each Macintosh font has only one substitute font in Windows, and vice versa. This one-to-one font mapping ensures that Director will be able to assign the appropriate substitute font when you edit a text cast member on one platform and then open the movie on the other platform.

A movie's Fontmap.txt file might look like this:

```
; This is a sample Fontmap.txt file

; Comments are denoted by using ";" or "--" to startÂ the line

; The format for Font Mapping is:

; Platform:FontName => Platform:FontName [MAP (NONE | ALL)] [OLDSIZE => NEWSIZE]

-- The format for specific Character Mapping is

-- Platform: => Platform:  OLDCHAR => NEWCHAR ...

; Here are sample mappings for the standard Mac fonts:

Mac:Chicago       => Win:"MS Sans Serif"

Mac:Courier       => Win:"Courier New"

Mac:Geneva        => Win:System Map All

Mac:Helvetica     => Win:Arial

Mac:Monaco        => Win:Terminal

Mac:"New York"    => Win:"MS Serif" Map None

Mac:Symbol        => Win:Symbol

Mac:Times         => Win:"Times New Roman" 14=>12 18=>14 24=>18 30=>24

; Here are sample mappings for the stock Windows fonts

Win:Arial             => Mac:Helvetica Map All

Win:Courier                                      => Mac:Courier

Win:"Courier New"     => Mac:Courier

Win:"MS Serif"        => Mac:"New York" Map None

Win:"MS Sans Serif"   => Mac:Chicago

Win:Symbol            => Mac:Symbol
```

```
Win:System            => Mac:Geneva

Win:Terminal          => Mac:Monaco

Win:"Times New Roman" => Mac:Times 12=>14 14=>18 18=>24 24=>30

; Note: From Windows to Mac, Courier and Courier New map onto Courier. When
coming back to Windows only Courier New will be used.

; Here is a sample character mapping for the bullet char

Mac: => Win: 165=>149

Win: => Win: 149=>165
```

Note that:

- Comment lines must begin with two dashes (--) or a semicolon (;)
- Only one font mapping definition can be specified on a line
- Arguments must be separated by spaces or tabs
- If a font name consists of more than one word, it must be enclosed in quotation marks.

{button See Also,AL(`Editing_Fontmap')}

Cast members and sprites

Cast members are a movie's basic elements. Cast members include bitmap images, text, sounds, buttons, digital videos, and more. You create most cast members by importing media such as bitmaps, sounds, and digital videos created in other programs. Cast members are organized into casts. Make as many casts as you need to organize the media in a movie.

Sprites are objects representing when, where, and how cast members appear in the movie. By creating multiple sprites, you can make a single cast member appear in different places and times in a movie. Create a sprite by dragging a cast member to the Stage or Score.

For a demonstration of these concepts, see the **Cast Members and Sprites** movie.

Creating a Director movie is largely a process of defining where sprites appear on the Stage, when they appear in the movie, how they behave, and what their properties are.

{button See also,AL(`UsersGuide_001_help')}

Using the Stage and Score

The Stage and Score are where you assemble a movie. The Stage shows what is happening in the movie at a particular time. The Score shows what happens in the movie over time.

The Stage

The Stage is what the viewer sees when a movie is complete. While creating a movie, you can perform many actions directly on the Stage. Unlike other windows, the Stage has no title or scroll bars, and can extend to the edges of the screen. To set the size of the Stage, choose Modify > Movie > Properties. The default size is 640 by 480 pixels.

Note: The menu bar is active but invisible when only the Stage is showing. To display a menu, move the pointer to the top of the screen to the menu's position, and hold down the mouse button. In Windows, the Stage will not hide the menu bar unless the Director window is maximized.

The Score

The Score displays the state of all the elements in your movie over time. Choose Window > Score to open the Score.

When you create a new sprite, Director places the sprite's image on the Stage and displays information about when the sprite appears in the Score. Sprites appear in the Score as bars extending across all the frames in which the sprite appears.

{button Illustration,PI(`',`UG_illustration_1_100')}

The numbers across the top of the Score show the number of each frame. Frames represent a single step in the movie, like the frames in a traditional film. You set the playback speed of a movie by specifying the number of frames to be displayed per second. See the **Controlling movie playback** help topic.

{button Illustration,PI(`',`UG_illustration_1_800')}

The playback head moves through the Score to show what frame is currently displayed on the Stage. The playback head moves to any frame you click in the Score.

When you click a sprite to select it, a small image of the sprite's cast member appears in the upper left corner of the Score. Information such as the size and location, and start and end frame of the selected sprite appears to the right.

{button Illustration,PI(`',`UG_illustration_3_6303')}

Using Score channels

Different channels in the Score store different types of information.

{button Illustration,PI(`',`UG_illustration_1_005')}

When the Score first appears, the effects channels are not visible, only the sprite and script channels are visible. Click the Show/Hide Effect Channels button on the right side of the Score to show or hide the effect channels.

{button Illustration,PI(`',`UG_illustration_3_740a')}

{button See also,AL(`UsersGuide_002_help')}

Controlling movie playback

Control how movies play back with the Control Panel, buttons on the toolbar, or keyboard shortcuts.

{button Illustration,PI(``,`UG_illustration_2_060')}

The Control Panel provides most of the basic functions you need to play back movies. Choose Window > Control Panel to open this floating panel.

Note: Be sure the movie is stopped before editing.

{button See also,AL(`UsersGuide_003a_help')}

Cast basics

A cast is a library of graphics, sounds, color palettes, behaviors and Lingo scripts, buttons, transitions, digital video movies, and text used in a Director movie. Each movie's cast can contain up to 32,000 cast members.

You can create as many casts as you need for a movie. Use casts to:

- Separate different types of media.
- Organize sections of a movie.
- Share casts between movies.
- Work on the same movie with other people without having to merge changes back into the same file.
- Reuse casts in other movies.
- Dynamically update a movie with new media published on the internet.

{button See also,AL(`UsersGuide_004_help')}

Understanding internal and external casts

There are two types of casts: internal and external. The cast that appears when you create a new movie is an internal cast.

Internal casts are stored inside movie files and can't be shared with other movies.

External casts are stored outside the movie file and can be shared with other movies.

Use internal casts to store media not shared with other movies. Use external casts to store shared media or commonly used elements. See the **Using external casts** help topic.

{button See also,AL(`UsersGuide_005_help')}

Creating casts

To create a new cast, choose File > New > Cast. When the New Cast dialog box appears, choose either the Internal or External option.

If you choose External, choose a setting for the Use in Current Movie option to determine if the cast is linked to the current movie. See the **Using external casts** help topic.

{button See also,AL(`UsersGuide_006_help')}

Working with the Cast window

The Cast window displays the members of the current cast. You can open as many windows as you need to display the different casts in your movie, or you can use the cast pop-up to select a different cast for display in the current window. The name of the current cast appears in the Cast window title bar.

There are several buttons at the top of the Cast window to help you work with cast members:

Most of these buttons also appear at the top of all the windows you use to edit cast members. There are only a few differences:

The Cast window displays information for each cast member.

For every occupied position in the Cast window, Director displays an icon that represents the cast member's type.

You control the size of the Cast window, the size of the thumbnail images, and other optional settings with File > Preferences > Cast.

{button See also,AL(`UsersGuide_007_help')}

Opening Cast windows

To	Do this
Open an existing cast in a new Cast window	Choose Window > Cast or press Control-3 (Windows), or Command-3 (Macintosh). If you have more than one cast, select a cast name from the cast submenu.
Change the cast displayed in the current Cast window	Click the cast button and choose a cast from the pop-up.
Create a new Cast window	Select a Cast window and choose Window > New.

{button See also,AL(`UsersGuide_008_help')}

Selecting cast members

To	Do this
Select a range of cast members	Click the first cast member in the range. Shift-click the last one in the range.
Select multiple non-adjacent cast members	Control-click (Windows) or Command-click (Macintosh) all the cast members you want to select.

{button See also,AL(`UsersGuide_009_help')}

Moving cast members

You move cast members between positions in any open Cast windows by dragging and dropping.

To	Do this
Move a cast member to a new position or a different cast	Drag the cast member to a new position in any open Cast window. A highlight bar appears to show you where the cast member will be placed.
Cut, copy, and paste cast members to a new position or a different cast	Select a cast member, choose Cut or Copy from the Edit menu, select a position in any open Cast window, and then choose Paste.
Move a cast member to a position not presently visible	Select the cast member you want to move. Scroll the Cast window to display the destination position. Drag from the drag well to the destination position and release the mouse button.

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. However, Director doesn't automatically update references to cast member numbers in Lingo scripts. To avoid problems with moving cast members and Lingo, name cast members and refer to them by name in Lingo. See the **Naming cast members** help topic.

{button See also,AL(`UsersGuide_010_help')}

Importing cast members

To import cast members:

1. **Select the window for the cast to which you want to import.**

   If you want cast members in a certain place in the cast, select the position. Otherwise, Director places the new cast members in the first available position.

2. **Choose File > Import.**

3. **Use the Import dialog box to select files.**
   The pop-up in the dialog box displays all the file types you can import.

   You can switch folders and import files from different folders at the same time. To import from the internet, click Internet and enter a URL.

4. **Choose an option from the Media pop-up at the bottom of the dialog box.**

Choose	To
Standard Import	Import all selected files so they are stored inside the movie file.
Link to External File	Create a link to the selected files and import the data each time the movie runs. For more information, see the **Linking to a file**UsersGuide_013_help help topic..
Include Original Data for Editing	Preserve the original data within the movie file for use with an external editor. For more information, see the **Launching external editors** help topic.
Import Pict File as PICT	Do not convert PICT files to bitmaps.

5. **When you've finished selecting the files, click Import.**
   Director prompts you if it needs more information to complete the import, such as color depth and palette options.

   For some types of media, you will need to convert or modify files before importing them into Director. For example, to import images from vector graphics programs like FreeHand and Illustrator, you must first convert them to bitmaps.

For details about importing	Read
Bitmaps	**Changing the color depth and palette during import**
Digital video	**Importing digital videos**
Director movies	**Importing movies**
Sound	**Using sounds**
Text	**Importing text**

Tip: You can also import cast members by dragging files from the desktop to the Cast window.

{button See also,AL(`UsersGuide_012_help')}

Linking to a file

For most types of imported media, you have the choice of importing the data into the movie or linking to the external file. To link to an external file, choose Link to External File from the Media pop-up at the bottom of the Import dialog box when you import the file. With this option turned on, Director imports the media every time the movie runs.

Linking is especially useful for showing media from the internet that changes frequently. It also makes it easier to use bulky media such as long sounds and large bitmaps. When you link to an external file, Director creates a cast member that stores the name and location of the file.

Only the link to a linked cast member is saved as part of the movie. If you distribute a movie, you need to include all linked cast members as well. The linked cast members must remain in the same file system position relative to the movie. For more information, see the **Assigning pathnames** help topic.

Director uses different Xtras to import media of each type. The Xtras that import the media types for all linked cast members must be present when a movie runs. In most cases, Director handles this automatically. For more information, see the **Managing Xtras for distributed movies** help topic

Note: It's best to keep linked files in a folder that's close to the original movie file. Pathnames are restricted to 255 characters by the system. URLs can be up to 260 characters. (You can enter URLs up to 4000 characters in Lingo.) If you store files too many folders away from the movie, or at a very long URL, it may fail to link correctly.

{button See also,AL(`UsersGuide_013_help')}

Importing from the internet

Including linked media from the internet provides a way to dynamically update movies. Linked media might include pictures that change often, sports scores, stock quotes, and so on.

You can import any supported file directly from the internet by clicking Internet in the Import dialog box and entering a URL.

If you import media from the internet using the Standard Import option, Director immediately retrieves the file from the internet (if a connection is available). Director stores the media inside the movie and does not update it if the source material changes.

To dynamically update media from the internet, choose Link to External File from the Media pop-up at the bottom of the Import dialog box when you import a file. Using this import option, Director imports the media from the specified location every time the movie runs.

Linking to external media is also useful for making movies download faster. Movies often supply content that users may never choose to view. If this type of content is linked externally, it won't be downloaded unless it's needed.

There are several issues to consider when using linked media on the internet:

- The media must be present at the specified URL when the movie runs. Since you can never be certain an internet transaction will be successful, be sure to make some provision for the link failing. Use the included Net Show Proxy behavior to specify an alternative cast member to display if a linked cast member is not available. For more information, see the "Display Proxy Until Loaded" behavior in the **Behaviors for controlling media** help topic..
- Certain Xtras must be included with a projector for Director to be able to connect to the internet during playback. You can include these Xtras automatically by turning on Include Network Xtras in the Projector Options dialog box. Movies playing in web browsers do not require these Xtras. For more information, see the **Creating projectors** help topic.
- Director uses different Xtras to import media of each type. The Xtras that import the media types for all linked cast members must be present when a movie runs. The Shockwave players for Netscape Navigator and Microsoft Internet Explorer include the Xtras required to import GIF and JPEG graphics; and AIFF (compressed and uncompressed), Shockwave Audio, and WAV (uncompressed only) sounds. Shockwave movies playing in web browsers can import these media types without downloading Xtras. To import other types of media, the required Xtras must first be downloaded and installed.

  For projectors, the Xtra for each type of file being imported must be included with the projector. In most cases, Director handles this automatically. For more information, see the **Managing Xtras for distributed movies** help topic

Note: Use File > Preferences > Network to define standard network settings for the Director authoring environment. See the **Preferences > Network command** help topic for a description of Network Preferences.

{button See also,AL(`UsersGuide_014_help')}

Supported file types

Because file formats are often modified and new formats are created, this list may not be complete. See the Director Developers Center web site for possible updates to this information.

Type of file	Supported formats
Image	BMP, GIF, JPEG, LRG (xRes), Photoshop 3.0, MacPaint, PNG, TIFF, PICT
	Windows only: Photo CD, PCX, WMF, PostScript
Multiple image files	Windows only: FLC, FLI
	Macintosh only: PICS, Scrapbook
Sound	AIFF, WAV, uncompressed and IMA compressed
	Macintosh only: System 7 sounds
Video	QuickTime
	Windows only: AVI
Text	RTF, ASCII (often called "Text Only")
Palettes	Windows only: PAL

{button See also,AL(`UsersGuide_015_help')}

Importing issues

- If your movie imports directly from external files at runtime, you must include the appropriate importing Xtra. This can be accomplished automatically with the Check Movie Xtras option in Projector Options. See the **Creating projectors** help topic.
- Movies created with an older version of Director may have used compressed PICT files with a JPG extension. Since Director now supports JPEG files, there may be a conflict since both file types use the JPG extension. To use compressed PICT files with Director 6, rename them with the PIC extension.
- GIF and progressive JPEG images appear as single images. They do stream while loading as they do in some web browsers.
- To avoid low memory errors, save often during importing, especially when importing multiple file images (FLC, FLI, PICS).

{button See also,AL(`UsersGuide_016_help')}

Creating cast members within Director

Create certain types of cast members within Director using the commands on the Media Element, Control, or Other submenus on the Insert menu.

Choose	To
Insert > Media Element > Bitmap	Create a new bitmap cast member and open the Paint window for editing.
Insert > Media Element > Text	Create a new text cast member and open the Text window for editing. See the **Working with text** help topic.
Insert > Media Element > Color Palette	Create a new color palette cast member and open the color palette window for editing. See the **Changing color palettes** help topic
Insert > Media Element > Sound	Create a new sound cast member. See the **Creating sound cast members** help topic.
Insert > Controls > Custom Button	Create a custom button with Button Editor. See the **Using the Button Editor** help topic .
Insert > Controls Push Button, Radio Button, or Check Box	Create a button cast member and a sprite on the Stage. See the **Using shapes** help topic.
Insert > Controls > Field	Create a field cast member and create a sprite on the Stage. See the **Field window** help topic.
Insert > Others > SWA Streaming Xtra	Create a Streaming Shockwave Audio cast member. See the **Streaming external Shockwave Audio files** help topic.

You can also create simple shape cast members using the tools on the tools palette. See the **Using shapes** help topic.

Use the Add button in any of the media editing windows (Paint, Text, Script, and so on) to create a cast member of the corresponding type.

When you choose any of the commands for creating a cast member, Director places the new cast member in the first empty position at or after the current selection in the Cast window. To place the cast member in a specific position, select the position first.

{button See also,AL(`UsersGuide_017_help')}

Using Xtra cast member types

Xtras can provide new types of cast members. Macromedia provides Xtras such as the Button Editor; others are provided by third-party developers. Documentation is usually provided with the Xtra.

Xtras that create cast members appear on the Insert menu.

{button See also,AL(`UsersGuide_018_help')}

Viewing cast member properties

The Cast Member Properties dialog box displays the selected cast member's name, number, type, size in kilobytes, and special options for different cast member types. Use the options in the dialog box to define the appearance and behavior of the selected cast members.

There are several ways to open the Cast Member Properties dialog box:

- Select a cast member in the Cast window and choose Modify > Cast Member > Properties.
- Right-click (Windows) or Control-click (Macintosh) a cast member and choose Cast Member Properties from the menu that appears.
- In the Cast window, the Sprite Inspector, the Sprite toolbar, or any of the editing window (for example the Paint window), click the Cast Member Properties button, .

{button See also,AL(`UsersGuide_019_help')}

Naming cast members

To prevent problems in Lingo scripts and behaviors that refer to cast members, provide a unique name for each cast member and use it to refer to the cast member instead of the cast member number. There is no difference in performance when referring to cast members by name. If you refer to a cast member by name, you don't need to worry about its number changing when you move it in the cast. The cast member's name stays the same even if its number changes.

To name a cast member:

- Select the cast member in the cast and enter a name in the name field at the top of the Cast window, or in any of the editing windows.
- Enter a name in the Cast Member Properties dialog box.

{button See also,AL(`UsersGuide_020_help')}

To find cast members:

1. **Choose** Edit > Find > Cast Member.

2. **In the Find Cast Member dialog box, choose a cast to search from the Cast pop-up.**
   Choose All Casts to search every cast in the movie.

3. **Choose one of the search options.**
   If you choose Name, enter search text in the field to the right. If you choose Type or Palette, choose an option from the pop-up:

4. **Once Director displays the cast members you're looking for, you can:**
   - 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 the listed cast members in the Cast window.

   {button See also,AL(`UsersGuide_022_help')}

To find a cast member in the Score:

1. **Select the cast member you want to search for.**

   Select a cast member in the cast or the Score. If you select a sprite, Director searches for the first cast member in the sprite. Open the sprite to select a cast member other than the first.

2. **Choose Edit > Find > Selection, or press Control-H (Windows) or Command-H (Macintosh).**
   Director searches the Score and highlights the first found Score cell.

3. **Choose** Edit > Find Again to find the next occurrence of the cast member in the Score.

{button See also,AL(`UsersGuide_023_help')}

To sort the cast:

1. **Bring the Cast window you want to sort to the front.**
2. **Select the cast members you want to sort, or choose** Edit > Select All.
3. **Choose** Modify > Sort.
   The Sort Cast Members dialog box appears.
4. **Choose one of the sorting methods.**
5. **Click Sort.**
   Director reorders the cast members according to the sorting method you selected.

The Score automatically adjusts to the new cast member numbers.

It's helpful to use Sort to bring all the cast members of one type together. You can then select each category separately and use Sort again to alphabetize the category.

Note: When you sort a cast, Director moves many cast members to new positions in the Cast window. If you've written Lingo scripts that refer to cast members by number, Lingo will not be able to find cast members that have been moved. It's best to name those cast members and then change the references in the scripts to the cast members' names. Otherwise, you'll need to update the reference numbers one by one every time you sort the cast.

Using external casts

External casts are stored as separate files outside the movie file. They have many useful applications:

- Storing elements used by different movies
- Creating libraries of commonly used cast members such as buttons and behaviors
- Switching entire groups of media at runtime, for purposes such as switching languages

{button See also,AL(`UsersGuide_025_help')}

Linking and unlinking casts

External casts can be linked or unlinked to movies. Use linked external casts for media that is shared between several movies. Once Director links an external cast to a movie, it opens the cast every time it opens the movie.

Use unlinked external casts as libraries to store commonly used elements for authoring like scripts, buttons, and so on.

To link an external cast to a movie, choose Modify > Movie > Casts and then click Link in the Movie Casts dialog box . You can link to casts on your local disk, or to casts stored at any URL. Click Network to enter a URL for a linked external cast.

When you drag a cast member from an unlinked external cast to the Stage or Score, a dialog box offers you the choice of linking the cast to the movie or copying the cast member to an internal cast.

To use a cast member from an external cast without creating a link to the cast file, first copy the cast member to an internal cast.

Use Modify > Movie > Casts to unlink casts from the current movie. See the **Movie Casts** help topic for more information.

{button See also,AL(`UsersGuide_026_help')}

Using external casts as libraries

External casts can serve as libraries of commonly used elements. Use the word "library" as the last word in the name of the cast to instruct Director to use the cast as a library (for example, Behavior Library, Button Library, Palette Library). 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. The library is not linked to the movie. If you place an external cast library in the Xtras folder, it appears on the Xtras menu. This is how the Behavior Library and other libraries included with Director were created.

{button See also,AL(`UsersGuide_027_help')}

Opening and saving external casts

If you don't link an external cast to a movie, you must open and save the file separately with the Open and Save commands.

Save All saves the movie and all open cast files, linked and unlinked.

{button See also,AL(`UsersGuide_028_help')}

Distributing movies with external casts

When you distribute a movie that uses an external cast, either on disk or on the web, you must include the external cast file. For disk-based movies, the cast must be in the same position relative to the movie as it was when the movie was created. For Shockwave movies on the web, the cast must be at the specified URL.

{button See also,AL(`UsersGuide_029_help')}

Referring to external casts in Lingo

Always refer to casts by name in Lingo. The cast number is not a reliable means of identifying a cast. Director assigns numbers to casts for each movie in order of creation. External casts may have different numbers when used in different movies.

{button See also,AL(`UsersGuide_030_help')}

Creating sprites

To create a new sprite:

1. **Select a cell in the Score where you want the sprite to begin.**

2. **Drag a cast member from the Cast window to the Stage or Score.**
   Director creates a new sprite with a duration of 28 frames. You can change this setting with File > Preferences > Sprite. See the **Sprite Preferences** help topic.

If you drag the cast member to the Score, Director places the new sprite in the center of the Stage.

To make a sprite only one frame long, hold down Alt (Windows) or Option (Macintosh) as you drag the cast member.

The same cast member can appear in multiple sprites. A sprite can also contain several cast members.

Tip: When two bitmap sprites overlap on the Stage, you often notice a white box around the image. Use the Matte or Background Transparent Score inks to remove the white box. See the **Applying ink effects to sprites** help topic.

{button See also,AL(`UsersGuide_032_help')}

Default sprite duration

Director assigns each new sprite a default duration of 28 frames. You can change this setting with File > Preferences > Sprite. If the Terminate at Markers option is turned on in the Sprite Preferences dialog box, Director makes new sprites end two frames before a marker. See the **Using markers** help topic.

{button See also,AL(`UsersGuide_033_help')}

Moving and resizing sprites

To move several sprites at once, select them on the Stage, or in the Score, and drag any one of them.

Hold down Shift as you drag to constrain the movement to vertical or horizontal.

Use the arrow keys to move selected sprites one pixel at a time. This works whether you select the sprite on the Stage or in the Score.

To precisely resize a sprite or to scale it by a certain percentage, use Modify > Sprite > Properties. See the **Viewing and changing sprite properties** help topic.

Changing the size or location of a sprite on the Stage doesn't change the cast member that the sprite is based on. Once you've changed the size of a sprite, changing the size of the sprite's cast member does not affect the size of the sprite.

Note: Resizing sprites can dramatically slow down movies. If you need a sprite to be a particular size, you should attempt to 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.

{button See also,AL(`UsersGuide_034_help')}

Changing the stacking order of sprites

Use Arrange commands on the Modify menu, to change the order of sprites on the Stage. Sprites appear on the Stage in order according to their channel number. A sprite in channel two appears on top of a sprite in channel one, and so on.

{button Illustration,PI(``,`UG_illustration_3_740b')}

To move the contents of an entire channel, first select the channel by double-clicking the channel number.

{button See also,AL(`UsersGuide_035_help')}

Selecting sprites

To select	Do this
An entire sprite	Click the sprite on the Stage. In the Score, click the horizontal line within a sprite bar, (not the keyframes*, or the start and end frame).
Keyframes*	Click the keyframe indicators.
A contiguous range of sprites	Select a sprite at one end of the range, then Shift-click a sprite at the other end.
Discontiguous sprites	Control-click (Windows) or Command-click (Macintosh) discontiguous sprites.
Keyframes* and sprites at the same time	Control-click (Windows) or Command-click (Macintosh) the elements you want to select.
A frame within a sprite that isn't a keyframe	Alt-click (Windows) or Option-click (Macintosh) a frame within the sprite.
Areas in the score that include Sprites and empty cells	Alt-click (Windows) or Option-click (Macintosh) empty frames and then drag across sprites, or use Shift to extend the selection.

*Keyframes are where a tweenable property of a sprite changes. See the **Tweening sprites** help topic.

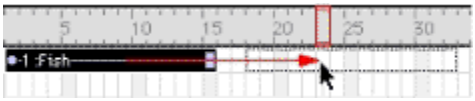
Changing when sprites appear

Use the Score to change when sprites appear. Choose Window > Score to open the Score window. You change when sprites appear by moving them between frames and stretching or shrinking the sprite bars.

{button See also,AL(`UsersGuide_037_help')}

Moving sprites between frames

The horizontal bars in the Score show the frames in which a sprite appears. To make a sprite appear at a different time, drag the bar to a different frame.



Drag a sprite to change when it appears

You can also use the Sprite Inspector to change the start or end frame. See the **Using the Sprite Inspector** help topic.

You can't move a sprite to a frame already occupied by another sprite unless you hold down Control (Windows) or Command (Macintosh) as you drag a sprite. This instructs Director to replace any sprites occupying the destination frames.

{button See also,AL(`UsersGuide_038_help')}

To change sprite duration

To change when a sprite appears or disappears, drag the start or end frame. The start and end frame are the first and last frames in a sprite. You can also use the Sprite Inspector to change the start and end frame. See the **Using the Sprite Inspector** help topic.

Note: See the **Tweening sprites** help topic for a complete description of keyframes.

Alt-drag (Windows) or Option-drag (Macintosh) a keyframe at the end of a sprite to extend the sprite and leave the last keyframe in place.

{button See also,AL(`UsersGuide_039_help')}

Extending sprites

The Extend command extends a selected sprite to the current location of the Playback Head. Select sprites to extend, click in the frame channel to move the Playback Head, and then choose Modify > Extend Sprite.

Extend Sprite works on multiple selected sprites. This is useful for aligning several sprites at a certain frame.

Extend Sprite can shorten a sprite if you move the Playback Head inside the sprite.

Extend Sprite moves the sprite's start frame if you place the Playback Head to the left of the sprite.

{button See also,AL(`UsersGuide_040_help')}

Viewing and changing sprite properties

Director can display the most important sprite properties directly on the Stage. To turn on this view, choose View > Sprite Overlay > Show Info.

Sprite information displays in small panels below the sprite. Use View > Sprite Overlay > Settings to specify whether sprite information appears for all sprites, when the pointer rolls over sprites, or for sprites that are selected.

Tip: Drag the horizontal line on the right side of a sprite information panel to change the panel's opacity.

In the Sprite Overlay panel, click the icons on the left to edit the data.

Click	To
	Open the Cast Member Properties dialog box for the Sprite's cast member. See the **Viewing cast member properties** help topic.
	Open the Sprite Properties dialog box for the current sprite.
	Open the Behavior Inspector. See the **Creating and changing behaviors** help topic.

{button See also,AL(`UsersGuide_041_help')}

Using the Sprite Inspector

Use the Sprite Inspector to view and edit sprite properties.

You can use the Sprite Inspector as a floating window or as a toolbar at the top of the Score.

- Choose Windows > Inspectors > Sprite to display the Sprite Inspector in a floating window.
- Choose View > Sprite Toolbar to show or hide the Sprite toolbar in the Score.

The Sprite Inspector shows properties for the currently selected sprite. When you select multiple sprites, the Sprite Inspector displays any settings all the selected sprites have in common. If you enter new settings, the change affects all selected sprites.

To change the width and orientation of the Sprite Inspector, drag the lower right corner to resize the window. You can display the Sprite Inspector in a narrow vertical or horizontal state, or in a wider, double-column state

Changes made in the Sprite Inspector take place immediately and affect all selected sprites.

For a description of all the sprite properties, see the **Sprite Inspector** help topic.

Sprite coordinates

You can precisely locate a sprite on the Stage using various coordinates. To arrange some sprites, you might want to specify the coordinates for the top edge; for others, the registration point may be more important.

{button Illustration,PI(``,`UG_illustration_3_715')}

All sprite coordinates are measured in pixels, with 0,0 at the upper left corner of the Stage. Sprite coordinates include the width and height; the top, left, right, and bottom edges of the sprite's bounding rectangle; and the X and Y coordinates of the sprite's registration point.

The registration point is a point within the boundaries of a cast member that Director uses to align cast members of different sizes and shapes. Director places the registration point in the center of the bounding rectangle for all bitmapped cast members; for other types of cast members, the registration point is at the upper left corner. You can easily move a cast member's registration point in the Paint window. See the **Using registration points** help topic.

You can change settings for these coordinates in the Sprite Properties dialog box, the Sprite Inspector, or from Lingo. When you change the setting for the top, bottom, left, or right edges of a sprite, Director resizes the sprite. It's best to use the X and Y coordinates (the registration point) of the sprite to move the sprite without resizing it.

{button See also,AL(`UsersGuide_043_help')}

Applying ink effects to sprites

Apply ink effects to change the way sprites appear on the Stage. Inks most often are used to remove bounding boxes from images, but they can also create many interesting and useful effects.

The Score window inks fall into two main categories:

- General purpose-Copy, Matte, Transparent, Reverse, Ghost, Not Copy, Not Transparent, Not Reverse, Not Ghost, and Mask
- Color bitmaps only-Background Transparent, Blend, Darkest, Lightest, Add, Add Pin, Subtract, and Subtract Pin

Note: Text cast members only support Copy, Background Transparent, and Blend inks.

For a demonstration and description of all the inks, see the **Ink Effects** movie.

To change the ink for a sprite, select the sprite and choose an ink from the Ink pop-up in the Sprite Inspector. Your choice replaces the previous ink assigned to the sprite. The Ink pop-up also shows the ink applied to selected sprites in the Score.

The normal ink is Copy. When you use Copy, a white bounding box appears around the sprite. To eliminate the bounding box that appears around a sprite, select the sprite in the Score and choose Matte or Background Transparent ink. Matte ink removes all the white pixels around the edges of an image. Background Transparent removes all the background color pixels in the entire image. Use it when an image has holes or windows through which you want the background to show.

Note: If Background Transparent or Matte inks don't 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.

For a detailed description of each type of ink, refer to the **Ink pop-up** help topic.

Blending sprites

Use blending along with tweening to make sprites fade in or out of a background. (See the **To fade a sprite in or out** help topic.) Change a sprite's blend setting in the Sprite Inspector or in the Sprite Properties dialog box. Blend settings are not visible unless you apply the Blend ink.

Scaling sprites

Use the Scale option in the Sprite Properties dialog box to scale sprites by a certain percentage. This is useful when you want to resize several sprites by the same amount. To scale a sprite, select it and choose Modify > Sprite > Properties, and then enter a percentage in the Scaling box.

{button See also,AL(`UsersGuide_046_help')}

Tweening sprites

Tweening" is a traditional animation term that describes when a lead animator draws only the frames where major changes take place, and assistants draw all the frames in between.

In Director, tweening is the simplest way to make sprites change as the movie plays. You define certain properties for sprites in keyframes and Director automatically changes the sprite in the frames in between.

A keyframe is where any tweenable property changes. Properties that can be tweened are position, size, foreground and background color, and blend setting. Each keyframe defines a value for all of these properties, even if you only explicitly define one.

For a demonstration of tweening, see the movie **Tweening Sprites**.

{button See also,AL(`UsersGuide_047_help')}

To tween a sprite:

1. **Move a sprite on the Stage to the place you want the motion to start.**

   This places the start frame of the sprite in the proper location. The start frame is also the first keyframe in the sprite.

2. **In the Score, click a frame within the sprite and choose Insert > Keyframe. (If the Score isn't open, choose Window > Score.)**
   Director inserts the keyframe at the Playback Head. Choose the frame in which you want the sprite to stop moving or change direction.

3. **On the Stage, drag the sprite to where you want it to stop moving or change directions.**

4. **Repeat steps 2 and 3 to define additional keyframes.**
   Note that the end frame is *not* a key frame unless you create one there.

5. **Rewind and play the movie to see how the sprite moves between the points you defined in the keyframes.**

Adjust the path of a sprite at any time by selecting a keyframe in the score and then moving the sprite on the Stage. Remember, if you select the sprite on the Stage without first selecting a keyframe in the Score, the entire sprite is selected; moving the sprite in this state moves the entire sprite, not just the keyframe location. The best way to adjust the path of a moving sprite is with View > Sprite Overlay > Show Paths. For more information, see **Showing and editing sprite paths**.

You can also move keyframes back and forth in the Score to adjust when the change occurs.

Change how the sprite moves and define curved paths with Modify > Sprite > Tweening.

Tip: To speed up tweening, Alt-click (Windows) or Option-click (Macintosh) a frame within a sprite to select only the frame, and then simply move the sprite on the Stage. This creates a new keyframe and defines the sprite location at the same time.

{button See also,AL(`UsersGuide_048_help')}

Tweening options

By default, sprites curve between the keyframes you define. Use Modify > Sprite > Tweening to specify tweening option to make sprites circle, speed up, or slow down. You can also make a sprite change size, shape, orientation, color, or blend value as the sprite is tweened.

Combine tweening options to create different effects. For example, a sprite could show a bird moving along a curving path, getting smaller, and fading out.

Director tweens the size and position for all sprites by default. To change this setting, choose File > Preference > Sprite and turn off Tween Size and Position. For movies created in Director 5, Tween Size and Position is off by default.

Tip: Turn off all tweening options to make a sprite jump instantly between settings in different keyframes.

{button See also,AL(`UsersGuide_049a_help')}

To change the curve of a sprite:

1. **Position a sprite where you want the motion to start and make sure it spans all the frames in which you want the sprite to move.**
2. **Create at least three keyframes in which the sprite occupies a different position.**
   Remember, the last frame in the sprite is not a keyframe unless you place one there. To make the sprite move in a circle, use the same start and end point.
3. **Choose Modify > Sprite > Tweening.**
4. **Use the Curvature slider to define how the sprite curves. If you want the sprite to move in a circle, turn on Continuous at Endpoints.**

{button See also,AL(`UsersGuide_049b_help')}

To make a sprite shrink or grow (tweening the size or shape):

1. **Position a sprite and make sure it spans all the frames in which you want the sprite to change.**
2. **Select the start frame of the sprite in the Score and change the size of the sprite.**
3. **Create a keyframe at the end of the sprite and change the size of the sprite.**
4. **Choose Modify >Sprite > Tweening and make sure Size is turned on the Sprite Tweening dialog box.**

{button See also,AL(`UsersGuide_049c_help')}

To fade a sprite in or out:

1. **Position a sprite and make sure it spans all the frames in which you want the sprite to change.**
2. **Apply the Blend ink to the sprite.**
3. **Select the start frame of the sprite in the Score and define the beginning blend setting in the Sprite Inspector.**
   Choose 0 to make the sprite fade in or 100 to make it fade out. For more information, see the **Blending sprites** help topic.
4. **Create a keyframe at the end of the sprite in the Score and define the ending blend setting.**
5. **Choose Modify > Sprite > Tweening and make sure Blend is turned on the Sprite Tweening dialog box.**

{button See also,AL(`UsersGuide_049f_help')}

To tween a sprite's foreground or background color:

1. **Position a sprite and make sure it spans all the frames in which you want the sprite to change.**
2. **Select the start frame of the sprite in the Score and specify the beginning color in the first frame of the sprite.**
   Use the color pop-up on the tool palette to specify the colors. See the **Using shapes** help topic.
3. **Select the end frame of the sprite in the Score and specify the ending color in the last frame of the sprite.**
4. **Choose Modify > Sprites > Tweening and make sure Foreground Color or Background Color is turned on in the Sprites Tweening dialog box.**

Tweening colors works best with 1-bit (black-and-white) images or shapes, but it can also create interesting effects with 8-bit images. Director tweens colors by replacing the image's foreground or background colors with colors from the current palette that are between the colors defined in the sprite's keyframes. Control what colors appear during the tweening by arranging the color palette.

{button See also,AL(`UsersGuide_049d_help')}

To accelerate or decelerate a sprite:

Use the Ease-in or Ease-out options in the Sprite Tweening dialog box to create more natural motion in tweened sprites.

Use one of the tweening methods to create a moving sprite.

Use the Ease In and Ease Out sliders to specify the percentage of the sprite's path through which it should accelerate or decelerate.

Turn on View > Sprite Overlay > Show Paths to see how far the sprite moves between each frame.

{button See also,AL(`UsersGuide_049e_help')}

Tweening speed

The Speed settings in the Sprites Tweening dialog box control how Director moves a sprite between keyframes. The Ease In and Ease Out controls have a similar effect on the complete length of the sprite's path.

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 an unequal number of frames apart from each other in the Score, or different amounts of space apart from each other on the Stage, this can cause abrupt changes in speed as the sprite moves between keyframe locations.

You can smooth out these speed changes by choosing the Smooth Changes option or eliminate them with the Constant option. To see what these options actually do in an animation, see the "Tweening Sprites" movie in Director Help.

This option	Does this
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.
Constant	Adjusts the sprite's keyframes so the sprite can maintain a constant speed as it moves between the points.

{button See also,AL(`UsersGuide_050_help')}

Showing and editing sprite paths

Choose View > Sprite Overlay > Show Paths to display and edit the path of any moving sprite. With Show Paths turned on, Director displays the path of moving sprites on the Stage. Keyframes appear as hollow circles. Small tick marks show the sprite's position in tweened frames.

- To change the sprite's path, drag a keyframe.
- To create a new keyframe in a sprite's path, hold down Alt (Windows) or Option (Macintosh)and move the pointer over a tick mark on the sprite path. When the pointer changes to cross-hair, click to create a keyframe.

Use View > Sprite Overlay > Settings to specify whether sprite paths appear for all sprites, only when the pointer rolls over sprites, or only for selected sprites.

{button See also,AL(`UsersGuide_051_help')}

Tweening suggestions and shortcuts

- To define keyframes more quickly, Alt-click (Windows) or Option-click (Macintosh) a frame in the score within a sprite to select only the frame, and then simply move the sprite on the Stage. This creates a new keyframe and defines the sprite location at the same time.
- 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.
- Choose View > Sprite Overlay > Show Paths to display and edit the path of any moving sprite.
- Alt-drag (Windows) or Option-drag (Macintosh) keyframes to quickly make duplicates. This is useful when you want the start frame and end frame 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.
- Alt-drag (Windows) or Option-drag (Macintosh) a keyframe at the end of a sprite to extend the sprite and leave the last keyframe in place.
- Control-click (Windows) or Command-click (Macintosh) multiple keyframes to select them and then move the sprite on the Stage to move all keyframe positions at once.
- To make the animation look smoother, use an image editor to blur the edges of bitmaps.
- Turn off all tweening options to make a sprite jump instantly between settings in different keyframes.
- For tweening sprites that have a series of cast members, consider using a film loop instead. See the **Using film loops** help topic.

{button See also,AL(`UsersGuide_052_help')}

To exchange cast members:

1. **Select a sprite.**
2. **Open the Cast window and select the cast member you want to use next in the animation.**
3. **Choose Edit > Exchange Cast Members.**

Director replaces the cast member for the entire sprite.

You can exchange the cast members in sprites and maintain all other sprite properties. This 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.

{button See also,AL(`UsersGuide_053_help')}

Editing sprite frames

The Edit Sprite Frames command changes how a sprite is selected and how keyframes are created. Use Edit Sprite Frames with sprites that have animation you need to adjust frequently. It is 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.

{button Illustration,PI(``,`UG_illustration_3_962')}

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.

{button Illustration,PI(``,`UG_illustration_3_963')}

- To use Edit Sprites 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 them and choose Edit > Edit Entire Sprite. You can also Alt-double-click (Windows) or Option-double-click (Macintosh) a frame within the sprite.

Tip: To select the entire sprite when it is open, Alt-click (Windows) or Option-click (Macintosh) the sprite in the Score or on the Stage.

{button See also,AL(`UsersGuide_054_help')}

Animating with a series of cast members

Use a series of cast members to create animation more complex than is possible with simple tweening. Sprites usually refer to only one cast member, but they can also include several cast members. For example, an animation of a man walking may include several cast members showing the man in different positions. By placing all the images within a single sprite, you can work with the animation as if it were a single object.

{button Illustration,PI(``,`UG_illustration_3_725')}

Tip: The best way to prepare cast members for use in multiple cast member animation is with onion skinning in the Paint window. See **the Using onion skinning** help topic.

{button See also,AL(`UsersGuide_055_help')}

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. See the **Score viewing options** help topic.

4. **Choose View > Sprite Labels > Changes Only.**
   This changes the view of the Score to show the name of sprite's cast member when it changes. This makes it easy to identify frames where the cast member changes.

5. **Choose Edit > Edit Sprite Frames.**
   Edit Sprite Frames makes it easier to select frames within a sprite. See the **Editing sprite frames** help topic.

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 member's 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 bitmapped cast member's bounding rectangle.

For information about aligning registration points, see the **Using registration points** help topic. You can also align sprites relative to their bounding rectangles. See the **Aligning sprites** help topic.

{button See also,AL(`UsersGuide_056_help')}

Animating with Cast to Time

Use Cast to Time to move a series of cast members to the Score as a single sprite. This 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 series of images for use in animation. See the **Using onion skinning** help topic.

{button See also,AL(`UsersGuide_057_help')}

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.

   {button See also,AL(`UsersGuide_058_help')}

Using Space to Time

Use the Space to Time command on the Modify menu to move sprites from adjacent channels to a single sprite. 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.

Onion skinning provides a similar benefit in the Paint window as Space to Time does on the Stage. See the **Using onion skinning** help topic.

{button See also,AL(`UsersGuide_059_help')}

To use the Space to Time command:

1. **Choose File > Preferences > Sprite and set the Span Duration to 1 frame.**
   You can 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 on 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 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 (usually 1).**
   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.

Tip: 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 Space to Time from the Modify menu, and add 10 to 20 cells between each cast member to produce a smooth curve.

{button See also,AL(`UsersGuide_060_help')}

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 moves across the Stage at the same time.

You can also use film loops to consolidate Score data. Film loops are especially helpful when sprite channels are in short supply. You can combine several Score channels into a film loop in a single channel.

To better understand film loops, see the **Film Loops**. movie.

If you delete any cast member used in the film loop from the Cast window, Director can't run the film loop. You can edit or reposition the cast members in the Cast window, but the cast members in the film loop must remain in the same cast for the film loop to work.

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.
- When you lengthen or shorten a sprite containing a film loop, it doesn't affect how the fast the film loop plays. It changes how many 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. For a comparison of all methods, see the **Comparing film loops, digital videos, linked movies, and MIAWs** help topic.

{button See also,AL(`UsersGuide_061_help')}

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 the sound channel. Select sequences in all the channels you want to be part of the film loop. You can select fragments of sprites if you first select them 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.

Tip: Drag a selection from the Score to the Cast window to quickly create a film loop cast member in that position.

{button See also,AL(`UsersGuide_062_help')}

Editing film loops

If you need to edit a film loop and you've deleted the original Score data it was based upon, 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.

{button See also,AL(`UsersGuide_063_help')}

To split an existing sprite:

1. **In the Score, click the frame within a sprite where you want the split to occur.**
   This moves the Playback Head to the frame.

2. **Choose Modify > Split Sprite.**
   Director splits the old sprite and creates two new sprites.
   {button See also,AL(`UsersGuide_065_help')}

To join separate sprites into a single sprite:

1. **Select all the sprites you want to join.**
   There can be gaps between the sprites, but the sprites must be in the same channel.
2. **Choose Modify > Join Sprite.**

{button See also,AL(`UsersGuide_066_help')}

Step recording

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. It is useful for creating sprites that follow irregular paths.

{button See also,AL(`UsersGuide_067_help')}

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 to begin animating.**
4. **Choose Control > Step Recording.**
5. **Press 3 on the number pad, or click on the control panel.**
   The movie advances to the next frame. If you reach the last frame in a sprite, Director extends the sprite 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, or by dragging the Playback Head- 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. **Rewind the movie or click a new frame in the Score to stop recording.**

{button See also,AL(`UsersGuide_068_help')}

Real-time recording

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 cursor, or for quickly creating a complex motion for later refinement.

For better control when you're recording in real time, use the Tempo control in the Control Panel to record at a speed that's slower than normal.

{button See also,AL(`UsersGuide_069_help')}

To use real-time recording:

1. **Click a frame in a sprite where you want animation to begin.**

   It's usually best to click a frame at the start or end of a sprite. Recording will begin at the playback head. Be sure to select a sprite in a completely empty score channel.

   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.**
   Recording begins as soon as you drag the sprite on stage, so be prepared to move the mouse.

3. **Drag the sprite on stage to record a path for the sprite.**
   Director records the path.

4. **Release the mouse button to stop recording.**

Tip: If you select Trails in the Score, you can also use real-time recording to simulate handwriting.

{button See also,AL(`UsersGuide_070_help')}

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.

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.

{button See also,AL(`UsersGuide_072_help')}

To align sprites:

1. **Select the sprites you want to align on the Stage or in the Score.**

   Select entire sprites, keyframes, or frames within open sprites in as many different frames or channels as you need. All of the elements will be aligned to the last sprite or frame you select. Director always aligns sprites to the location in the first frame.

2. **Choose Modify > Align.**
   The vertical and horizontal alignment options appear on two pop-ups.

3. **Select the options you want and click Align.**

The Align panel stays open until you close it.

Tip: You can also double-click the regions in the preview area to set alignment options.

{button See also,AL(`UsersGuide_073_help')}

To align sprites to the grid

Use the grid to position sprites on the Stage. When you move sprites while Snap to Grid is on, their edges and registration points snap to the nearest grid line.

Choose View > Grids > Snap To to turn on Snap to Grid.

Tip: Press and hold down G while moving or resizing a sprite to temporarily turn Snap to Grid off or on. You can also click Control-Alt-G (Windows) or Command-Option-G (Macintosh) to turn Snap to Grid off and on.

{button See also,AL(`UsersGuide_074_help')}

To add or remove 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. If there are sprites in the frames you select, they are tweened or extended.

Use Insert > Remove Frames to remove frames.

{button See also,AL(`UsersGuide_075a_help')}

Selecting, moving and deleting frames in the Score

To move or delete all the contents of a range of frames, double-click and drag in the frame channel. If you cut or delete the selected frames, Director removes the frames and closes up the empty space.

When you select frames, any sprite within the range is selected, even if it extends beyond the range. If you switch to the Director 5 Score display, only the sprite frames within the range are selected. For information about using the Director 5 Score display, see the **Using Director 5 Score options** help topic.

You can paste frames in any location as long as you select a frame first.

{button See also,AL(`UsersGuide_075b_help')}

Using markers

Markers identify fixed locations at a particular frame in a movie.

Click in the markers channel to create a marker. To delete a marker, drag it up or down out of the markers channel.

Markers are vital to navigation in movies. Using Lingo or draggable behaviors, you can instantly move the Playback Head to any marker frame. You can jump to markers while authoring using the Next and Previous marker buttons on the left side of the markers channel, by pressing the 4 and 6 keys on the number pad, or by choosing markers from the Markers menu.

Note: If the Terminate at Markers option is turned on in the Sprite Preferences dialog box, Director makes newly created sprites end two frames before a marker.

When you create a marker, an insertion point appears to the right of the marker so you can type a short name. Use the markers window to review and edit marker names.

Once you've marked a frame in the Score, you can use the marker name in your 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 frame numbers, which may change if you insert or delete frames in the Score.

{button See also,AL(`UsersGuide_076_help')}

The Markers window

Use the Markers window to write comments associated with markers you set in the Score. Double-click one of the markers in the Score window to open the Markers window to the comment associated with that frame.

The left column of the Markers window displays the marker names in the Score. Click on a marker in this list to move the Playback Head to that location in the Score. Advancing through the markers with the left or right arrows will also move the Playback Head. Comments associated with the markers appear in the right column.

To enter a comment, click a marker name and then enter your comments beginning at the insertion point that appears in the right column of the Markers window. By default, the marker name appears as the first line of text in the right column. If you don't want to edit the marker name, press Enter (Windows) or Return (Macintosh) to start a new line.

Tip: Use Control-left or -right arrow (Windows) or Command-left or -right arrow (Macintosh) to move to the previous or next marker.

{button See also,AL(`UsersGuide_077_help')}

Score viewing options

There are many ways to change the view of the Score to make your work easier. You can view multiple windows, turn channels off and on, zoom in and out, and change the data displayed within Score cells.

{button See also,AL(`UsersGuide_078_help')}

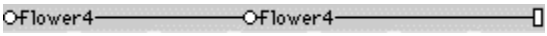
Zooming the Score

Change the zoom percentage to narrow or widen the Score. Zooming in widens each frame so it's possible to see more data in each frame. Zooming out shows more frames in less space. Zooming out is useful when moving around large blocks of Score data. To change the zoom setting, choose an option on the Zoom submenu of the View menu, or on the Zoom pop-up to the right of the Score.

Using sprite labels

Sprite labels are the information that appears within the sprite bars in the Score. They display vital information for troubleshooting a movie. 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 may also want to track the precise location of a sprite in every frame using Extended display.

The following figure shows the different options for displaying sprite labels. All these options are available on the View > Sprite Labels submenu.



Keyframes

Changes Only (shown at 800%)

Every frame (shown at 800%)

First Frame

{button See also,AL(`UsersGuide_080_help')}

Display options

Use the various display options to change the information in the sprite labels. Choose the display type from the Display pop-up in the Score, or the View > Display submenu.

Display option	Displays
Member	The name and/or number of the sprite's cast member (change the setting in File > Cast > Preferences)
Behavior	The behavior attached to the sprite
Location	The X and Y coordinates of the sprite's registration point
Ink	The ink effect applied to each sprite
Blend	The blend percentage
Extended	Any combination of display options -choose options with File > Preferences > Score

{button See also,AL(`UsersGuide_081_help')}

Data tips

Data tips are small pop-up panels that display the cast member name and number when the pointer is over a sprite for a few seconds.

Turn data tips off or on with the Data Tips option in the Sprite Preferences dialog box.

{button See also,AL(`UsersGuide_082_help')}

Using multiple Score windows

View different parts of a movie at the same time by opening additional Score windows. To open a new Score window, activate the current Score window and choose Window > New Window. This creates a new Score window that you can scroll to a different location in the Score. Use a second Score window to work on two places in the movie at once without scrolling.

Note: Only the first Score window automatically scrolls to show the Playback Head location.

{button See also,AL(`UsersGuide_083_help')}

Turning a channel on and off

Click the button next to a channel number to turn the channel off temporarily. None of the sprites in the channel appear in the movie and any behaviors are ignored. This can be useful for testing performance or for working on complex overlapping animations.

If you turn the script channel off, Director ignores all scripts during playback. (This is the same as checking the Disable Scripts command in the Control menu.)

These settings are not saved with a movie, they last only one session.

{button See also,AL(`UsersGuide_084_help')}

Showing and hiding the effects channels

There are five special Score channels above the markers channel. These channels are for tempo settings, palette changes, transitions, sounds, and frame scripts.

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

{button See also,AL(`UsersGuide_085_help')}

Using Director 5 Score options

The improvements in the Score for Director 6 make work easier for most users, but if you are familiar with older versions of Director you may be more comfortable using the older version of the Score for some types of work. This is especially true when working on movies that were created with older versions of Director.

By setting various preferences it's possible to simulate the functioning of the old Score. You may find many new Score features useful in the old Score and vice versa. For example, you may want to use the old Score view with the new Sprite toolbar or use the new Score with Edit Sprite Frames on for all sprites. You can mix new and old functions by selecting different Score and sprite preferences.

{button See also,AL(`UsersGuide_086_help')}

To make the Score work as it did in Director 5:

1. **Choose File > Preferences > Score and set options as shown in the following table.:**

Set this option	To
Director 5 Style Score Display: On	* Change the view of the Score so frames appear as a plain grid. * Make frames selectable and editable within sprites.
Script Preview: On	Make the Script Preview box appear at the top of the Score. The Script Preview also works in the new Score view.
Drag-and-Drop: Optional	Enable drag-and-drop of score sections in the Director 5 Score view.

2. **Choose File > Preferences > Sprite and set options as shown in the following table.**

Set this option	To
Stage Selection: Current Frame Only	Change the way sprites are selected so that clicking a sprite on the Stage selects only the current frame instead of the sprite's complete span.
Display Sprite Frames: On (not necessary if Director 5 Style Score display is on)	Make all frames within new sprites selectable and editable, as they were in Director 5. See the **Editing sprite frames** help topic.
Tween Size and Position: Off	Turn off automatic tweening of size and position for new sprites.

The following steps are optional:

- To turn on the Sprite toolbar, choose View > Sprite Toolbar.
- If existing sprites are being tweened automatically, select them and turn off all tweening options in the Sprites Tweening dialog box.
- Alt-drag (Windows) or Option-drag (Macintosh) from empty cells to select empty cells and sprite frames in the new score as you would in the old score.

{button See also,AL(`UsersGuide_087_help')}

Understanding behaviors

Behaviors are special cast members that define operations or procedures. Most behaviors are made to respond to a simple event like a sprite being clicked or the Playback Head entering a frame. When the specified event occurs, the behavior carries out an action, such as moving the Playback Head to a different frame or playing a sound.

For more information about using behaviors, see the **Behaviors** movie .

Attach behaviors to sprites or frames. You can attach as many behaviors as you need to a sprite, but you can only attach one behavior to a frame.

Attach a behavior to a frame to make an action occur when the Playback Head enters or exits the frame. Behaviors attached to frames are best suited to actions that affect the whole movie. For example, you might attach Go to Movie to the last frame in a movie to make it run a new movie when it reaches the end.

Attach a behavior to a sprite to cause actions directly related to the sprite. The most common use for sprite behaviors is creating buttons for navigating a movie.

Note: For Lingo programmers, behaviors are the same as score scripts. To edit behaviors in the Script window instead of the Behavior Inspector, choose File > Preference > Editor. Choose Behaviors in the Editor Preferences dialog box. Click Editor and choose Script Window.

{button See also,AL(`UsersGuide_089_help')}

Attaching behaviors

Attach behaviors to sprites or frames. To open the library of include behaviors, choose Xtras > Behavior Library. To see descriptions of the included behaviors, choose Window > Inspectors > Behavior to open the Behavior Inspector, and then select behaviors in the Cast window. For more information about included behaviors, see the **Using included behaviors** help topic.

You can only attach one behavior to a frame. If you attach a behavior to a frame that already has a behavior, the new behavior replaces the old.

Some behaviors are written to work only when applied to either a sprite or a frame; read the behavior descriptions to learn more.

{button See also,AL(`UsersGuide_090a_help')}

To attach a behavior to a sprite:

1. **Drag a behavior from the Behavior Library (or any Cast window) to a sprite on the Stage or in the Score.**

   {button Illustration,PI(`',`UG_illustration_4_400')}

2. **Enter parameters for the behavior in the Parameter dialog box.**
   You can attach as many behaviors as you need to a single sprite.

To attach the same behavior to several sprites at once, select the sprites in the Score and drag a behavior to any one of them.

Note: If you attach a behavior from the included Behaviors cast, Director copies it to an internal cast. This prevents you from accidentally changing the original behavior.

{button See also,AL(`UsersGuide_090b_help')}

To attach a behavior to a frame:

1. **Drag a behavior from the Behavior Library (or any Cast window) to a frame in the Score's script channel.**

   {button Illustration,PI(`',`UG_illustration_4_401')}

2. **Enter parameters for the behavior in the Parameter dialog box.**

You can only attach one behavior to a frame. If you attach a behavior to a frame that already has a behavior, the new behavior replaces the old.

Note: If you attach a behavior from the included Behaviors cast, Director copies it to an internal cast. This prevents you from accidentally changing the original behavior.

{button See also,AL(`UsersGuide_090c_help')}

To attach behaviors using the Behavior Inspector:

1. **Choose Window > Inspectors > Behavior to open the Behavior Inspector.**
2. **Select sprites or a frame in the script channel.**
3. **Choose a behavior from the Behavior pop-up.**
   Director attaches the behavior you choose to the sprite or frame.

{button See also,AL(`UsersGuide_090d_help')}

Entering behavior parameters

Many behaviors require additional information to work properly. For example, the Go to Marker behavior requires that you enter the name of a marker. When you attach one of these behaviors to a sprite or frame, the Parameters dialog box for the behavior appears. Use it to enter required parameters.

The parameters you enter apply only to the behavior as it is attached to the current sprite or frame. They do not affect how the behavior works when attached elsewhere. In this regard, they are similar to other sprite properties such as size and location that do not affect cast members.

To change parameters for a behavior that is already attached, select the sprite or frame where the behavior is attached, and then double-click the behavior name in the Behavior Inspector.

{button See also,AL(`UsersGuide_091_help')}

Getting information about behaviors

To see a description of a behavior, open the Behavior Inspector, select a behavior, and 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 with Director have descriptions. Behaviors from other sources may not. You can add descriptions to behaviors you create yourself.

{button See also,AL(`UsersGuide_092_help')}

To change the order of the behaviors attached to a sprite:

1. **Select the sprite in the Score or on the Stage.**
2. **Open the Behavior Inspector.**
3. **Select a behavior from the list.**
4. **Click the arrows in the toolbar to move the behavior.**

Director executes behaviors in the order they were attached to a sprite. It's often necessary to change the sequence of behaviors so that actions occur in the proper order.

{button See also,AL(`UsersGuide_093_help')}

Using included behaviors

Director includes a variety of behaviors for basic functions. To see all the included behaviors, choose Xtras > Behavior Library. When you attach any of these behaviors to a sprite or frame in a movie, Director copies them to an internal cast.

There are behaviors for more specialized functions in the Goodies folder on the Director CD.

{button See also,AL(`UsersGuide_094_help')}

Behaviors for navigating in a movie

These behaviors all move the playback head to specified locations in the movie, or to a URL.

To find these behaviors, choose Xtras > Behavior Library.

For more information about any included behavior, select the behavior, open the Behavior Inspector, and read the description in the description pane.

Behavior	Description	Example
Hold on Current Frame	Loop the Playback Head on the current frame until another action triggered by a behavior or script moves the Playback Head elsewhere.	Place in a frame containing a menu to make the movie stop playing while users make a selection.
Go to Frame	Move the Playback Head to a specified frame number.	Attach to a sprite to make it function as a button that moves the Playback Head to a certain frame number.
Go to Marker	Move the Playback Head to the specified marker.	Attach to a sprite to make it function as a navigation control to jump to another scene.
Go to Previous Marker	Move the Playback Head to the previous marker in the Score.	Attach to a sprite to make it function as a button that switches back to the previous scene.
Go to Next Marker	Move the Playback Head to the next marker in the Score.	Attach to a sprite to make it function as a button that advances to the next scene.
Go to Movie	Switch to a new movie.	Arrange a series of small movies on the web to reduce download time.
Go to Page	Loads a network page from the specified URL and displays it in a browser during authoring or when playing from a projector. If used in movie already playing in browser, the specified page is loaded in the browser.	Attach to a menu item so that on Mouse Up, a net page is displayed in the browser.
Go to Net Movie	Runs a Director movie at the specified URL.	Create a series of small movies to reduce download time and chain them together.
Open Movie in a Window	Opens a movie in a window.	Create a window that floats over the Stage for displaying help. For more information, see **Playing a movie in a window**.
Play Done	Identifies the end of a movie or sequence and returns the	Create a reusable control to place at the end of every

Playback Head to where the movie or sequence was started from.	scene in your movie that sends the Playback Head to wherever the scene was started from.

{button See also,AL(`UsersGuide_095_help')}

Behaviors for sending messages

Behaviors described here send messages to other sprites. Many behaviors are set to respond to events like the mouse being clicked or the playback head entering a frame. Other behaviors run when they receive a message from another behavior or from a Lingo script.

Sending messages is useful to make a behavior to run when something happens to a different sprite, or to make a whole group of behaviors attached to different sprites respond to a certain message.

To find these behaviors, choose Xtras > Behavior Library.

For more information about any included behavior, select the behavior, open the Behavior Inspector, and read the description in the description pane.

Behavior	Description	Example
Message Sprite	Send a specified message to a sprite in a certain channel.	Attach to a button to make it send a message to another sprite that makes it switch cast members.
Message All Sprites	Send a specified message to all sprites in the current frame.	Send a message to several sprites at once to make them all switch cast members.
Message Movie in a Window	Send a message to a movie in a window.	Create controls that play, stop, and rewind a movie in a window.

{button See also,AL(`UsersGuide_096_help')}

Behaviors for controlling media

These behaviors are for controlling media such as sound and digital video, and for switching cast members.

To find these behaviors, choose Xtras > Behavior Library.

For more information about any included behavior, select the behavior, open the Behavior Inspector, and read the description in the description pane.

Behavior	Description	Example
Sound Beep	Play the system beep on a specified event.	Attach to a button to make it beep when clicked.
Sound Play Cast Member	Play a sound cast member on a specified event.	To play a click sound when a button is pressed, drag this behavior to the button and set Which Event to mouseDown and click the Start Immediately checkbox.
Sound Play File	Play an external sound file on a specified event.	Place in a frame in the script channel to stream a long sound track directly from disk. For more information about streaming sound, see **Importing sounds**.
Image Switch Cast Members	Switch a sprite's cast member when the switchstates message is received. Enter a name to identify groups of sprites the behavior has been applied to.	Create a group of sprites that show cows, and a group of chickens. Attach the Image Switch Cast Member behavior to all the sprites and enter cows or chickens as the names for the behaviors attached to each group of sprites. Create a button and attach the Message All Sprites behavior. Send the message switchStates cows to make only the cows switch cast members. Do the same for the chickens, only send the message switchStates chickens.
Image Cycle Cast Members	Cycle through a range of contiguous cast members when the cycleState message is received. Enter a name to identify groups of sprites the behavior has been applied to.	Create a slide show that cycles through a series of cast members. Create a button that sends the cycleState message to the sprite that cycles cast members.
Media Preload	Preload a specified range of cast members into memory.	Place in the frame before a new a scene begins to load all the scene's cast members and improve performance.
Net Show Proxy	Specifies a graphic to be displayed until the real cast member of the sprite is downloaded. The real cast member appears automatically when it becomes available.	For a streaming Shockwave movie, define a small graphic to display while a larger image is downloading. Place the proxy cast member near the beginning of the Score to make sure it is one of the first cast members downloaded.
Net Hold Until Frame Ready	Loops the movie on the current frame until all the media required for the frame has been downloaded.	Place in a frame in the script channel at the beginning of a scene in a streaming movie to make sure all the cast members for the scene have been downloaded before the scene plays.

Net Get Text	Sets the text of a field sprite to the text retrieved from a specified URL.	Display text that is updated on the internet such as sports scores or stock prices in a movie. Enter a time-out value so the movie doesn't hang if there is no response from the URL.

{button See also,AL(`UsersGuide_097_help')}

Behaviors for making controls

These behaviors are for creating common types of controls such as push button, toggle buttons, and radio buttons.

To find these behaviors, choose Xtras > Behavior Library.

For more information about any included behavior, select the behavior, open the Behavior Inspector, and read the description in the description pane.

Behavior	Description	Example
Pushbutton	Creates a standard two-state push button. When the button is clicked it switches to the cast member specified by Down Member. If the pointer is dragged off the button, the original cast member reappears.	Attach this behavior to a sprite to create a button. Attach a second behavior to make the button do something.
Toggle button	Creates a standard toggle button that remains in the same state after being clicked. Replaces the Up member with the Down member on mouseUp. If the sprite is rolled off while the mouse is down, the change is canceled.	Use for buttons that define a series of options before carrying out an action, like the settings for a game.
Radio Group Item	Connects a group of radio buttons or checkboxes so that only one can be checked at a time. Sets the value of WhichRBSelected Group to the text of the selected radio button in the group.	Assign three radio buttons the same group name to insure that only one is checked at any time.
Video Play	Starts digital video playing on MouseUp.	Attach to a button to make a digital video sprite begin playing.
Video Stop	Stops digital video on mouseUp, does not rewind.	Attach to a button to make a digital video sprite stop playing.
Video Rewind	Rewinds a digital video sprite to the beginning on mouseUp.	Attach to a button to make it rewind a digital video.

{button See also,AL(`UsersGuide_098a_help')}

Behaviors for pointers, dragging, and rollovers

These behaviors change the pointer and affect moveable sprites.

To find these behaviors, choose Xtras > Behavior Library.

For more information about any included behavior, select the behavior, open the Behavior Inspector, and read the description in the description pane.

Behavior	Description	Example
Pointer Change	Replaces the system pointer with the specified cast member.	Define distinctive pointers for different levels of a game.
Pointer Animate	Replaces the system pointer with a cycling range of cast members. Start and stops cycling when the startCursor or stopCursor messages are received. The cast members must be 1-bit bitmaps and adjacent to each other in the cast window.	Create a watch pointer with hands that move while the movie is downloading cast members.
Rollover Change Pointer	Changes the system pointer while the pointer is over a certain sprite. The new pointer must be a 1-bit bitmap.	Make the pointer switch to a finger while over a button.
Rollover Change Member	Switch the cast member of a sprite when the pointer rolls over.	Make a button highlight when the pointer rolls over.
Drag Snap to Sprite	Make a moveable sprite snap to the registration point of a specified sprite.	Create clothes that snap to place on a certain figure.
Drag Snap to Sprite List	Make a moveable sprite snap to the registration points of a list of sprites.	Make clothes that snap to place on a number of different figures.

{button See also,AL(`UsersGuide_098b_help')}

Creating and changing behaviors

Use the Behavior Inspector to create and change behaviors.

All behaviors work by detecting an event, and then performing one or more actions in response. The Behavior Inspector lists the most common events and actions used in behaviors.

Even 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, it's a good idea to read Chapter One, "Script Basics" in the Learning Lingo book.

Using the Behavior Inspector in this way is a good way to begin learning 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.

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.

{button See also,AL(`UsersGuide_100_help')}

To create or modify a behavior:

1. **Do one of the following:**

   - Select a behavior in the Behavior Inspector
   - Click the Behavior pop-up, choose New Behavior, and enter a name for the new behavior.

   If you create a new behavior, it appears in the currently selected cast. Select an empty cast position first if you want it to appear in a certain place.

2. **Click the arrow to expand the editing pane of the Behavior Inspector.**
   {button Illustration,PI(`',`UG_illustration_4_406')}

   The editing pane shows the events and actions in the current behavior. If you're creating a new behavior, no events or actions appear. There are several steps you can take at this point.

To	Do this
Add a new event/action group to the behavior	Choose an event from the Event pop-up and then choose actions for the event from the Actions pop-up. You can choose as many actions as you need for a single event.
Change an existing event/action group	Choose an event from the list and then add or remove actions in the actions list.
Delete an event/action group	Choose the event and press Delete.
Change the sequence of actions in an event/actions group	Choose an event from the list, choose an action from the action list, and then click the up and down arrows above the action list to change the order of actions.

The included actions and events are basic building blocks you can use to create simple or complex behaviors. For more descriptions and examples of creating behaviors, see the Director Developers Center web site.

If you are familiar with Lingo, you can also edit a behavior's script directly.

{button See also,AL(`UsersGuide_101_help')}

Included Events

Event	Description
Mouse Up	A mouse button was released.
Mouse Down	A mouse button was clicked.
Right Mouse Up	The right mouse button was released. (On the Macintosh, Director treats Control-clicking the same as clicking the right mouse button on Windows system's keyboard.)
Right Mouse Down	The right mouse button was clicked.
Mouse Enter	The pointer entered a sprite's region.
Mouse Leave	The cursor left a sprite's region.
Mouse Within	The cursor is within the sprite's region.
key Up	A key was released.
key Down	A key was pressed.
Prepare Frame	The playback head has left the previous frame, but has not yet entered the next frame.
Enter Frame	The playback head entered the current frame.
Exit Frame	The playback head exited the current frame.
New Event	A specified message was received from a script or behavior.

{button See also,AL(`UsersGuide_102a_help')}

Included Actions

Action	Description
Go to Frame	Move the playback head to the specified frame.
Go to Movie	Open and play the specified movie.
Go to Marker	Move the playback head to the specified marker.
Go to Net Page	Go to the specified URL.
Wait on Current Frame	Wait at the current frame until another behavior or script advances to the next frame.
Wait Until Click	Wait at the current frame until the mouse button is clicked.
Wait Until Key Press	Wait at the current frame until a key is pressed.
Wait for Time Duration	Wait at the current frame for the specified time.
Play Cast Member	Play the specified sound cast member.
Play External File	Play the specified external sound file.
Beep	Play the current system beep.
Set Volume	Set the system volume level to the specified setting.
Change Tempo	Change the movie's tempo to the specified setting.
Perform Transition	Perform the specified transition.
Change Palette	Change to the specified palette.
Change Location	Move the current sprite to the specified coordinates.
Change Cast Member	Switch the sprite's cast member to the specified cast member.
Change Ink	Switch to the specified ink.
Change Cursor	Change the pointer to the specified cursor number. .
Restore Cursor	Restore the current system pointer.
New Action	Execute any Lingo function or send a message to a handler.

{button See also,AL(`UsersGuide_102b_help')}

Using the Button Editor

The Button Editor creates custom buttons that display separate down, up, rollover, and disabled conditions. The Button Editor creates single cast members that display all these conditions automatically, without any scripting or references to other cast members.

A button can be either a push button or toggle button. Push buttons return to the same state after they are clicked. Toggle buttons remain in a "toggled" state after they are clicked. There are different options available for each type of button.

Buttons do not do anything until you attach a behavior or Lingo script to them.

{button See also,AL(`UsersGuide_106_help')}

Creating and editing custom buttons

It's easiest to create a button in two stages. First create the button by defining the button type, label text, and other options; then arrange the bitmaps for the button in different states.

To create a custom button:

1. **Choose Insert > Controls > Custom Button.**
   The Button Editor dialog box appears.
2. **Enter a label for the button.**
   This is the text that appears on the button.
3. **Click Font to set the text formatting for the label text.**
   Specify the font, size, style, and kerning for the text that appears on the button.
4. **Choose either Push Button or Toggle Button.**
   - If you choose Push Button, click Enabled to make the button enabled by default.
   - If you choose Toggle Button, click Toggled to make the button toggled by default.
6. **Click OK to close the Button Editor and create the new custom button cast member.**

The new cast member appears on the stage, but you haven't yet defined the bitmaps for the button.

To define button bitmaps:

1. **Create or import a series of bitmapped cast members that represent the button in different states.**
2. **Select all the bitmapped cast members and choose Edit > Copy.**
3. **Double-click the button cast member to reopen the Button Editor.**
4. **Click the Bitmaps tab in the Button Editor.**
   The Button Editor changes to display the bitmap fields for the button in all of its states.
5. **Select the first bitmap field and choose Edit > Paste.**
   The bitmapped cast members are pasted into the fields. You can cut and paste them between fields in the Button Editor to change their positions. You can also repeat these steps to copy single images.

Change any settings for an existing button by double-clicking the button in the cast to open the Button Editor. You can also open the Button Editor by clicking Options in the Xtra Cast Member Properties dialog box for a particular button.

Understanding color depth

The number of colors a graphic image or a computer system can display is called the color depth. Your computer monitor, the Director movie you're working on, and every cast member in the movie can have its own color depth setting.

Color depth is expressed as the number of bits used to specify the color of each pixel, or as the number of colors that can be displayed at one time.

Bit depth	Number of colors
1-bit	black and white
2-bit	4
4-bit	16
8-bit	256
16-bit	32,768
24-bit	16.7 million
32-bit	16.7 million plus eight channels of special effects

There are several reasons why you should create your movies using 8-bit graphics:

- Many low-end systems support only 8-bit graphics. This will be less true in the future, but if you intend to distribute to a wide audience, 8-bit is the safest choice.
- Graphics with lower color depth use less storage space and memory, and animate faster. Because graphics with higher color depths (16-, 24-, and 32-bit) contain so much information about color, they can rapidly use up all available memory and storage space.
- Director movies are especially demanding because they contain so many graphics.
- When playing movies on Windows 3.1 systems, Director supports only 8-bit graphics. Director for Windows 95 and NT supports higher bit depths.

If you are making a movie that will be played only on your own system, these issues may not concern you.

For instructions on changing the color depth of the movie and cast members, see the **Solving color palette problems** help topic.

Tip: Cast members with a bit depth lower than 8-bit often reduce the performance of 8-bit movies because Director must convert them to 8-bit before displaying them on the screen. Also, Director compresses 8-bit cast members based on the actual number of colors in the image so they are as small as 1-, 2- and 4-bit images.

{button See also,AL(`UsersGuide_112_help')}

Understanding color palettes

When your system is set to 8-bit color depth (256 colors) or less, it uses a limited set of colors called a color palette. The color palette determines all the colors that can be shown on the screen at the same time. This includes not only the colors for the images you're working with, but system elements like title bars, dialog boxes, tools, and so on.

For a full-color demonstration of how color palettes affect your work in Director, see the **Color Palettes** movie.

Note: If your system is set to display 16-, 24-, or 32-bit color (thousands or millions of colors), color palettes don't affect Director movies; they serve only as a means for selecting colors in the Paint window. Because they refer directly to a complete spectrum of colors, 16-, 24-, and 32-bit graphics do not require color palettes to achieve accurate color.

Color graphics with 2-, 4-, or 8-bit color depth don't store any information about the hue, saturation, or brightness of any particular color. They identify colors by referring to positions in the current color palette.

{button See also,AL(`UsersGuide_113_help')}

Conflicting color palettes in Director movies

Only one palette can be active on your computer at a time. This causes problems in Director when you try to put two bitmapped cast members that have different palettes on the Stage at the same time; the colors for one of the images will be wrong.

While a movie is playing, the palette channel in the Score determines the active palette. When the playback head reaches a frame containing a new palette, it changes the active palette. To make this change using Lingo, see the **puppetPalette** topic.

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. The new palette becomes the active palette, and it remains in effect until you set a different palette in the palette channel. For more information about using the palette channel, see the **Changing color palettes** help topic.

For a full-color demonstration of how color palettes affect your work in Director, see the **Color Palettes** movie.

{button See also,AL(`UsersGuide_115_help')}

Palettes in web browsers

When playing movies in web browsers, Shockwave movies don't take over the active color palette of a user's system the way a normal Director movie does. Shockwave remaps the colors in the Director movie to the most similar colors in the palette active in the user's system. See the **Managing color palettes for browsers** help topic.

{button See also,AL(`UsersGuide_116_help')}

Solving color palette problems

Here are some guidelines for solving problems caused by conflicting color palettes:

- Make sure all cast members on the Stage at the same time refer to the same palette.
- Simplify your work and avoid frequent palette changes by mapping all the images in your movie to as few palettes as possible.
- If possible, create a palette that contains all the colors you need in your movie. Do this within Director by modifying an existing palette in the color palette window. You can copy and paste colors from one palette to another. For information on using the color palette window, see the **Using the Color Palettes window** help topic. You can also use image editing programs like Debabelizer and the PlanetColor Xtra to scan groups of images, create an optimal palette, and then remap all of the images to the new palette.
- Remap existing cast members to a new color palette using the Transform Bitmap command.
- As you import cast members, you can remap them to new palettes using the Image Options dialog box.
- If the Import option for Palette is not available in the Import Options dialog box, 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.

  Black or white must be the first or last colors in the palette.

  There must be only one black and one white in the entire palette.
- Don't change colors used by your system software for interface elements. On Windows these color always appear as the first ten, and last ten, colors in the palette.
- If you don't understand what has been discussed so far, or if you're using images with simple colors, you can avoid all of these complexities by using Transform Bitmap to remap all of your images to the Windows or Macintosh system palette.

Note: Changing the color scheme in the Windows Display control panel has no effect on color display within Director.

{button See also,AL(`UsersGuide_117_help')}

To change the color depth and palette of bitmap cast members:

1. **Select the cast members you want to change in the cast.**
2. **Choose** Modify > Transform Bitmap.
3. **Choose a new depth setting from the Color Depth pop-up.**
   If you change 16-, 24-, or 32-bit cast members to 8-bits or less, you need to remap them to an existing color palette. See the **Understanding color palettes** help topic.
4. **Choose a new palette from the Palette pop-up.**
5. **Choose either Remap Colors or Dither.**
   - 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.
6. **Click Transform to change the palette.**

You can also remap images to new palettes with image editing programs such as xRes and PhotoShop.

{button See also,AL(`UsersGuide_118_help')}

Changing the color depth and palette during import

If you import a bitmap with a color palette or depth different than the current movie, the Image Options dialog box appears. You can choose to import the bitmap at its original color depth or at the Stage color depth.

{button See also,AL(`UsersGuide_119_help')}

Changing movie color depth

The color depth of a Director movie is determined by the setting of the color depth of the monitor when the movie is saved. Change the color depth of the movie by changing your monitor's color depth, and then save the movie.

{button See also,AL(`UsersGuide_120_help')}

Changing the active palette while authoring

While working on a movie, you can change the active palette with the Palette window or by displaying a cast member with a different palette. 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.

Tip: Since the palette you choose affects everything displayed on the monitor, including the interface, you may find it difficult to see what you're doing after you select a palette other than the system palette. Use the Classic Look (Monochrome) option in the General Preferences dialog box to display all of Director's screen elements in black and white.

{button See also,AL(`UsersGuide_121_help')}

Image resolution

When creating and 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 72 to 96 dots per inch with your image editing program before you import it into Director.

To resize a bitmapped cast member:

1. **Select the bitmapped cast member you want to change.**

   You can select as many cast members as you want to change in the Cast window. You can also change a cast member displayed in the Paint window.

2. **Choose Transform Bitmap from the Modify menu.**
   When the Transform Bitmap dialog box appears, do one of the following:

   - To change the size to a specific height or width, enter the new dimensions (in pixels) in Width and Height. If you click Maintain Proportions, the settings automatically change for the other field when you enter a number.

   - To change the size by a certain percentage, enter the percentage in Scale.

This procedure changes the size of the cast member itself. All associated sprites change size as well, unless they were resized individually.

Note: 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.

For information about resizing sprites, see **the Moving and resizing sprites** help topic.

{button See also,AL(`UsersGuide_124_help')}

Changing selected areas

Once you have selected part of an image with the lasso or selection rectangle, you can change the selected area in several ways.

Move the crosshair inside the selected area until the crosshair turns into an arrow pointer and drag the selected area to reposition it. Several key combinations affect the selected area when you drag it.

Effect	Mouse or key combination
Copy	Alt-drag (Windows) Option-drag (Macintosh)
Stretch (selection rectangle only)	Control-drag (Windows) Command-drag (Macintosh)
Stretch proportionally (selection rectangle only)	Control-Shift-drag (Windows) Command-Shift-drag (Macintosh)
Copy and stretch (selection rectangle only)	Control-Alt-drag (Windows) Command-Option-drag (Macintosh)
Constrain to horizontal or vertical	Shift-drag
Clear	Backspace (Windows) Delete (Macintosh)

You can also use the keypad arrow keys to move the selected area one pixel at a time.

{button See also,AL(`UsersGuide_127_help')}

Using registration points

Registration points are the main way of locating images in Director. A registration point provides a fixed reference point within a bitmapped image. 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. Use the Registration tool to move the registration point.

Moving the registration point is most 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 hand-place the sprites on the Stage. Use onion skinning when setting registration points to where images are placed in relation to each other. See the **Using onion skinning** help topic.

{button See also,AL(`UsersGuide_128_help')}

To set a registration point:

1. **Display the cast member you want to change in the Paint window.**

2. **Click the registration 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 crosshair 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.

Tip: To reset the default registration point at the center of the cast member, double-click the registration tool.

{button See Also,AL(`UsersGuide_129_help')}

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, choose View > Rulers.

To change the location of the zero point, drag right or left along the ruler at the top of the window or up or down along the ruler at the side.

{button See also,AL(`UsersGuide_130_help')}

Zooming in and out in the Paint window

Use the Zoom commands on the View menu to zoom in or out at four levels of magnification. The shortcut for zooming in is Control-plus (Windows) or Command-plus (Macintosh). The shortcut for zooming out is Control-minus (Windows) or Command-minus (Macintosh).

To zoom in on a particular feature of the image, Control-click (Windows) or Command-click (Macintosh) the image, or position the pointer over the feature before choosing Zoom In.

{button See also,AL(`UsersGuide_131_help')}

Using the Effects toolbar

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 selection Marquee. Some effects work only with the Marquee.

Tip: Press Control-Y (Windows) or Command-Y (Macintosh) to repeat any of these effects.

Use this button:	To:
Flip Horizontal, Flip Vertical	Flip the selected area from right to left or from top to bottom.
Rotate Left/Right	Rotate the selected area 90 degrees clockwise or counterclockwise.
Free Rotate	Rotate the selected area by hand in either direction. When you click the button, handles appear at the corners of the selection rectangle. Drag any handle in the desired direction.
Skew	Skew the selected artwork. Handles appear at the corners of the selection rectangle. Dragging a handle moves the opposing corner an equal amount in the same direction, maintaining a parallelogram shape.
Warp	Distort the shape of the selected area. Handles appear at the corners of the selection rectangle. Drag any handle in any direction independent of the other corners.
Perspective	Create a perspective effect. Handles appear at the corners of the selection rectangle. Drag one or more handles to create the effect you want. For example, you can bring the two top handles closer together to create the illusion of linear perspective.
Smooth	Soften the edges of the selected artwork by adding pixels of blended color to the artwork's edges. This can help animation appear smoother.
Trace Edges	Create an outline around the edges of the selected artwork. The outline is the same color as the selected line, if the line is a solid color. If the original line is multicolored, an outline is created for each section of the line. You can add multiple outlines by choosing Trace Edges or Repeat repeatedly.
Invert Color	Reverse the colors of the selected area.
Lighten/Darken Color	Increase or reduce the brightness of the selected area.
Fill Color	Fill the selected area with the current foreground color and pattern.
Switch Colors	Change each pixel that is the currently selected foreground color to the currently selected destination color. Works on 8-bit images only.

{button See also,AL(`UsersGuide_132_help')}

Using onion skinning

Onion skinning derives its name from a technique used by conventional animators who draw on very thin "onion skin" paper so that they can see one or more of the previous images in the animation.

For a demonstration of onion skinning, see the **Onion Skinning** movie.

Onion skinning in Director allows you to 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 one or more cast members blended into the image.

You can use onion skinning:

- To trace over an image or to create a series of images all registered with a particular image.
- When drawing each cast member of an animation, 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. A series of images serves as the background while you paint a series of foreground images.

{button See also,AL(`UsersGuide_133_help')}

Tracing a cast member with onion skinning

Follow these steps to trace a new cast member using another cast member as a background image:

1. **Open the Paint window and** choose View > Onion Skin.
2. **Open the cast member in the Paint window that you want to use as the reference image or background.**
3. **Click Toggle Onion Skin in the** Onion Skin toolbar to turn the feature on .
4. **Click Set Background to set the background image.**
5. **Click the Add button in the Paint window to create a new cast member.**
6. **Click Show Background in the** Onion Skin toolbar .
   The original cast member now appears as dimmed image in the Paint window. All painting operations now take place "on top of" the original cast member's image.
7. **Paint the new cast member using the background image as a reference.**

{button See also,AL(`UsersGuide_135_help')}

Creating new cast members using previous cast members as reference images

You can create new cast members using other cast members as reference images. Use the Preceding and Following Cast Members settings in the Onion Skin toolbar to help you step through a series of cast members created for your animation.

Drawing a series of images using another series of images as a reference

Follow these steps to use a series of images that serve as the background while painting a series of foreground images:

1. **Arrange the series of cast members you want to use as your background** in consecutive order in the cast.

   Cast members in each of the foreground and 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.

3. **Click Toggle Onion Skin in the** Onion Skin toolbar to turn onion skinning on .
   Make sure all values in the Onion Skin toolbar are set to 0.

4. **Open the cast member in the Paint window that you want to use as the first background cast member in the reference series and click Set Background.**

5. **Select the position in the cast where you want the first cast member in the foreground series to appear and click the add 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 Track Background in the** Onion Skin toolbar .
   The corresponding member of the reference series appears as a reference image.

7. **Paint the new cast member using the background image as a reference.**

8. **When you have finished drawing the cast member, click the add button again to create the next cast member.**
   When Track Background is enabled, Director advances to the next cast member in the reference series and its image appears in the background in the Paint window.

9. **Repeat step 8 until you have completed drawing the cast members in the series.**

{button See also,AL(`UsersGuide_137_help')}

Onion skinning and registration points

Onion skinning uses registration points to align images to each other. 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. For more information, see the **Using registration points** help topic.

{button See also,AL(`UsersGuide_138_help')}

To create a custom tile:

1. **Create a bitmapped cast member to use as a tile and display it in the Paint window.**
2. **Click the Pattern Chip in the Paint window and choose Tile Settings from the bottom of the Pattern pop-up.**
3. **Click an existing tile 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 on the lower left. The box on the right shows how the image appears when tiled. To dotted rectangle inside the cast member image shows the area used for the tile.

   If you want to choose a different cast member, use the arrow buttons to the right of the Cast Member button to switch 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 Tools palette to fill shapes.

{button See also,AL(`UsersGuide_138a_help')}

Using Auto Distort

Auto Distort generates tweened cast members for any cast member that is free rotated, made into a perspective, slanted, distorted, or stretched. After artwork has been altered with one of these five effects, and before you deselect the artwork, choose Xtras > Auto Distort, and enter the number of cast members you want to create. The new cast members are placed in the next available cast member positions.

For a rotated bitmap image, Auto Distort uses the center of the image as the rotation point. Consequently, you will have to use the paint window's registration tool to reset the registration point of each tweened cast member created by the Auto Distort operation.

{button See also,AL(`UsersGuide_139_help')}

Using bitmap filters

Bitmap filters are plug-in image editors that apply effects to bitmapped images. You can install Photoshop-compatible filters to change images within Director.

To install a filter, place the filter in the Xtras folder in the Director application folder. If you want to use the filter in other Macromedia applications, place the filter in the Xtras folder in the Macromedia folder in either the Windows folder or the Macintosh System Folder.

You can apply a filter to a selected portion of a bitmapped image, to the entire cast member, or to several cast members at once.

{button See also,AL(`UsersGuide_139_help')}

To apply a bitmap 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 selection marquee or the lasso 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.**
   Choose All as the category list to view all the filters at once.

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.

Tip: Bitmap filters are applied to the bounding box of the image. To make the rectangle less visible, use the Lasso tool to select the image before choosing Xtras > Filter Bitmap. You can also draw single dots in the upper left and lower right corners of the image to extend the bounding box away from the image itself, and then erase the edge dots in the filtered images.

{button See also,AL(`UsersGuide_141_help')}

Using Auto Filter

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. You define a beginning and ending setting for the filter, and then Auto Filter applies an intermediate filter value to each cast member.

For instructions on installing filters, see the **Using bitmap filters** help topic.

Note: Not every image filter supports auto filtering. The dialog box only lists those that do.

{button See also,AL(`UsersGuide_142_help')}

To use Auto Filter:

1. **Select a bitmapped cast member, or a range of cast members, and then choose** Xtras > Auto Filter.

   If you want to change only a portion of a bitmapped cast member, use the selection rectangle or the lasso 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.

   {button See also,AL(`UsersGuide_143_help')}

Working with text

You can create and edit text cast members directly on the Stage or in the Text window.

Note: There are three types of text in Director, each with distinct advantages for certain applications. The type of text described in this section, rich text, looks the best on screen and is the easiest to distribute. It also uses the most memory. For a comparison of all types of text, see the **Using different types of text in Director** help topic.

To open the Text window:

- Choose Text from the Window menu.
- Click the Text window button on the toolbar .
- Press Control-6 (Windows) or Command-6 (Macintosh).

{button See also,AL(`UsersGuide_144_help')}

Creating text cast members

There are three ways to create text cast members:

- Create text cast members directly on the Stage using the text tool in the tool palette . Click the text tool and then drag the pointer on the Stage to define the width of the text. When you release the mouse button, a text insertion point appears in the area you just defined and you can begin entering text. The new text cast member is placed in the first available position in the current cast. The sprite is placed in the first open Score cell in the current frame.
- Create text cast members in the Text window by choosing Media Element and then Text from the Insert menu. Text you enter appears in the first available cast position, but it is not automatically placed on the Stage.
- If the Text window is already open, click the Add button to create a new text cast member.

{button See also,AL(`UsersGuide_145_help')}

Importing text

You can import text from any application that saves text in the rich text format (RTF).

When importing text, Director creates a new cast member each time it encounters a page break or column break in the file.

The amount of text in a cast member is limited only by the memory available in the playback system.

{button See also,AL(`UsersGuide_147_help')}

Formatting text

You can change most of the text formatting settings with controls on the text inspector.

To display the Text Inspector, choose Text Inspector from the Window menu, or press Control-T (Windows) or Command-T (Macintosh). The same controls also appear at the top of the Text window, and in the Font and Paragraph dialog boxes.

Formatting characters

To	Do this
Specify the font of selected text	Choose a font from the pop-up.
Make selected text bold, italic, or underlined	Click the appropriate button in the text inspector.
Change the point size of selected text	Increase or decrease the size with the size controls in the text inspector.
Change the spacing between selected characters	Increase or decrease the character spacing with the control in the text inspector.
Change the color of selected text	Choose Modify > Font and use the pop-up color palette or use the color chip on the tool palette to select a new color.

{button See also,AL(`UsersGuide_148_help')}

Formatting paragraphs

You can specify alignment, indent, tabs, and spacing for each paragraph in a text cast member.

To make any formatting changes to a paragraph, first place the cursor in the paragraph you want to change, or select multiple paragraphs.

To	Do this
Set tabs	Click the tab well until the type of tab you want appears, and then click the ruler to place the tab. If the ruler isn't displayed, choose View > Ruler.
Move a tab	Drag the tab marker on the ruler.
Remove a tab	Drag the tab marker up or down off of the ruler.
Set margin	Drag the indent markers on the ruler.
Set line spacing	Change the setting with the line spacing control in the Text Inspector. Director adjusts line spacing to match the size of the text you're using. If you change the space setting, Director stops making automatic adjustments. To resume automatic adjustment of spacing, enter 0 in the line spacing box.
Set paragraph alignment	Click one of the alignment buttons in the Text Inspector.
Set spacing before and after paragraphs	Choose Modify > Paragraph and use the Spacing Before and After controls.

{button See also,AL(`UsersGuide_149_help')}

Using anti-aliased text

The anti-alias text feature smoothes the edges of text on the screen so that it appears without jagged angles and curves. Director blends the color of text with the background by adding pixels of intermediate colors. Anti-alias text is turned on by default. You change the setting with the Text Cast Member Properties dialog box. Select the cast member you want to change and then choose Modify > Cast Member Properties.

Anti-aliasing 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're using.

Director can anti-alias all outline (TrueType and PostScript) fonts but not bitmapped 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.

{button See also,AL(`UsersGuide_150_help')}

Using different types of text in Director

There are three ways of working with text in Director: rich text, fields, and bitmapped text. Each has distinct advantages for different applications.

{button See also,AL(`UsersGuide_151_help')}

Understanding rich text

Rich text is text that is editable in the authoring environment, but converted to an anti-aliased bitmapped graphic on the Stage when the movie plays. Rich text is what you create with the text tool or in the Text window. All the documentation, and Director itself, refers to rich text is as simply "text."

- Rich text offers paragraph formatting and definable tabs for each paragraph in a cast member.
- Rich text is always editable in the authoring environment, but when the movie plays it is turned into a graphic.
- Because rich text is converted to a graphic, it offers faster animation and does not require the same fonts to be installed on every system where the movie is played.
- Large fonts can be anti-aliased to improve on-screen appearance. No jagged lines appear at corners or along angles.
- You can import from any application that can save text in the standard rich text format (RTF). When importing text Director creates a new cast member each time it encounters a page or column break in the file.
- Rich text requires more storage space than fields, but much less than text created in the Paint window or an image editor.
- Text cast members support only three Score inks: Copy, Background Transparent, and Blend.

There are some drawbacks to using rich text:

- Rich text cast members use considerably more memory and storage space than field cast members. A rich text cast member uses four bits per pixel. Field text uses one byte per character. For example, a rich text cast member containing enough 12-point text to fill a normal page might take over 60K of memory. The same amount of text in a field cast member would use around 2 or 3K. If your memory or storage space is limited, there are many cases where using a field cast member may be more appropriate. For larger blocks of text and text in smaller point sizes, fields are usually preferable to rich text.
- Anti-aliasing provides no benefit to most typefaces smaller than 14 points. In some cases, it makes the text less legible. In general, anti-aliased rich text is best for headings, menu items, controls, and other items where you need small amounts of large, clear text.
- Rich text can't be edited while a movie is playing.
- The text in rich text cast members isn't accessible or controllable from Lingo. You can't change words at runtime or retrieve text.

{button See also,AL(`UsersGuide_152_help')}

Understanding fields

Field text is standard text controlled by your system software. A field is what you create with the field tool on the tool palette or by choosing Insert > Control > Field. In general, you should use fields for large blocks of small text (12-point or less). Use fields if you need text that is editable while the movie plays, or if you need to manipulate the text with Lingo.

The important things to know about fields are:

- The text displayed in fields is controlled by the system software, so the same fonts must be installed in each system every time the movie or projector is played. Try to use only standard system fonts for field text, like Arial for Windows and Helvetica for the Macintosh.
- Unlike text cast members, fields can be edited while a movie plays. They also can be controlled by Lingo in ways not possible with text cast members. See Chapter 6, "Working with Fields and User Input," in Learning Lingo. You specify that a field is editable in the Field Cast Member Properties dialog box, or from Lingo.
- Field text can't be anti-aliased.
- Paragraph formatting and tabs are not available.
- Fields animate more slowly than text cast members.
- Fields require less storage space than all other types of text.

{button See also,AL(`UsersGuide_153_help')}

Understanding bitmapped text

Bitmapped text is a raster image that happens to look like text. You can create graphics that look like text in the Paint window or any image editor, but you cannot edit it as text once you create it. However, you can change it using any paint functions. For example, you can create dramatic effects by applying Photoshop filters to bitmapped text. Use Modify > Convert to Bitmap to convert text and field cast members to bitmaps.

{button See also,AL(`UsersGuide_154_help')}

Creating fields

You can create a field cast member by choosing Insert > Control > Field, or with the Field tool in the Tools palette.

{button See also,AL(`UsersGuide_156_help')}

Editing and formatting fields

Just like 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.

The Field window does not have a ruler for setting tabs and indents.

To open the Field window, choose Field from the Window menu, or double-click a field cast member in the Cast window.

Note: Field cast members can include up to 32K of text.

{button See also,AL(`UsersGuide_157_help')}

Mapping fonts between platforms

When creating movies for cross-platform applications, be sure to consider how fonts are changed when the movie is moved between different systems.

Director converts text cast members to bitmaps, so text cast members appear exactly the same on any platform as they did on the system where they were created. You can work on a movie in Director that was created on a different platform and the text cast members still display their original fonts. Director assigns a new font only when you try to edit a text cast member and the original font is unavailable.

Field cast members always depend on the system software to display the proper font. For either authoring or playback on any platform (including Shockwave movies playing in web browsers), the font must be installed in the system. For this reason you should only use common system fonts for field cast members, or license and install custom fonts.

See the **Using different types of text in Director** help topic if you are unfamiliar with the difference between text and field cast members.

Director uses a file named Fontmap.txt to map fonts 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 table that follows. These settings provide the best equivalents of common system fonts on both platforms.

Windows font	Macintosh font
Arial	Helvetica
Courier	Courier
New Courier	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 like bullets and symbols are translated between platforms. Again, the default settings should be correct for nearly all applications, but you can edit the settings if necessary.

{button See also,AL(`UsersGuide_158_help')}

Working with digital video

Director supports QuickTime movies on Windows and Macintosh, and Video for Windows (AVI).

Director can also export movies or portions of movies as digital video. See the **Exporting digital video** help topic

{button See also,AL(`UsersGuide_159a_help')}

Using Direct to Stage

Director plays digital video using a feature called Direct to Stage. Direct to Stage allows Video for Windows or QuickTime to completely control the video playback.

Direct to Stage often provides the best frame rate, 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.

Use Cast Member Properties to turn off Direct to Stage for the current cast member. Turn off Direct to Stage when you want to place other sprites in front of a video and use ink effects on the video. When Direct to Stage is off, Director layers a digital video on the Stage exactly like other sprites, and the Background Transparent ink works normally. (Matte ink does not work for digital videos.)

Direct to Stage is the only available option for using QuickTime digital video cast members in Director for Windows. However, Direct to Stage can be turned off for Video for Windows (AVI) digital video cast members. On Macintosh, Direct to Stage can be disabled for QuickTime digital video cast members.

The Play Every Frame option in the Cast Member Properties dialog box ensures that no frames are skipped when the video plays. Depending on the data rate of the digital video, the digital video sprite may not play any more smoothly than before. In addition, playing every frame may cause the digital video to take more time to play. Note also that a digital video's soundtrack will not play if Play Every Frame is turned on.

Note In some cases a Direct to Stage digital video causes redraw problems on the Stage after it finishes playing. This occurs when Video for Window or QuickTime does not return control of the stage to Director. You can avoid this condition by placing a sprite the same size as the video (or larger) in the frame after it finishes playing. You can also use a simple Lingo script. For more information, see "Clearing the stage after a digital video finishes" in Chapter Seven of Learning Lingo.

{button See also,AL(`UsersGuide_159b_help')}

Importing digital videos

You import a digital video the same way you import any other type of cast member. The only difference is that digital videos always remain linked to the original file on disk. When you move a movie or projector to a different system, you must always make sure to include all digital videos.

{button See also,AL(`UsersGuide_160_help')}

Opening the Video window

You view digital videos in the Video window.

To open the Video window:

- Double-click a digital video cast member.
- Choose Window > Video.
- Press Control-9 (Windows) or Command-9 (Macintosh).
  The Video window appears:

Note: The video controller appears only for QuickTime videos.

{button See also,AL(`UsersGuide_161_help')}

Using video in Director movies

You add a video to a Director movie the same way you add any other cast member. Videos begin playing when the playback head reaches the frame containing the video.

Use Cast Member Properties to make the movie pause or loop.

Tip: If there's a white bounding box around the video, use the Background Transparent ink to remove it. (Matte ink doesn't work with digital video.)

A 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.

There are three ways to make sure that Director plays a video until the movie is finished:

- Create a tempo setting in the tempo channel using the Wait for End of Digital Video option in the Frame Properties: Tempo dialog box. This option keeps the playback head from moving to the next frame until the entire video has finished playing. You can also make the playback head wait for cue points within the video. See the **Using sound and video cue points** help topic.
- 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.
- Extend the video through enough frames to give it time to play all the way through.

Note: Lingo provides several more advanced digital video controls. See **Lingo elements - digital video**.
{button See also,AL(`UsersGuide_162_help')}

To crop a video:

1. **Select the cast member in the Cast window.**
2. **Choose** Modify > Cast Member Properties.
   The Digital Video Cast Member Properties dialog box appears.
3. **Select Crop.**
   If you select Center, as well as Crop, Director automatically centers the video inside the rectangular area you set as the size of the movie's image.
4. **Click OK.**
5. **Select the video in the Score.**
6. **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.

Cropping doesn't permanently remove the portions you crop; it just hides them. It's like adjusting shutters until just the part of the image you want to show is visible.

{button See also,AL(`UsersGuide_163_help')}

Using shapes

Shape cast members are similar to objects in drawing programs. You can resize them and change their shape and color right on the Stage after you create them.

Because shapes use less memory than bitmaps, they are especially useful for movies that will be distributed on the internet. However, they animate more slowly.

The tools for creating shapes are in the tool palette. Choose Window > Tool Palette.

To create a shape, select a cell in the frame where you want to draw a shape. Choose color, line thickness, and pattern settings with the controls in the tool palette. 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.

The Radio Button, Checkbox, and Button tools in the tool palette create simple buttons similar to those defined by your system software. Do not confuse them with more sophisticated Custom buttons created by the Button Editor. See the **Using the Button Editor** help topic.

Use the Field button to create Field cast members directly on the Stage.

{button See also,AL(`UsersGuide_164_help')}

Using OLE cast members

You can place linked and embedded (OLE) objects in Director for Windows movies as cast members. An OLE cast member appears as a bitmap inside of Director. For this reason, you should use OLE objects primarily to show images, text, or numbers. Sound and video OLE objects are not effective.

The OLE Object command on the Insert menu creates OLE cast members. You create a new OLE object by launching the source application from Director, or by embedding an existing file.

{button See also,AL(`UsersGuide_165_help')}

To create an OLE cast member:

1. **Choose** Insert > New OLE Object.

2. **In the Insert Object dialog box, choose Create New or Create from File.**
   - If you choose Create New, choose the type of OLE object you want from the Object Type list. The objects available depend on the OLE-compatible applications installed in your system.
   - If you choose Create from File, enter the path to the file or click Browse to select a file.

3. **Click OK to continue.**
   If you chose Create New, the source application of the OLE object appears. When you finish creating the object, choose File > Update. The object then appears in Director as a cast member.

   If you chose Create from File, the object is immediately placed in the cast.

{button See also,AL(`UsersGuide_166_help')}

Working with OLE cast members

You can use OLE objects as ordinary cast members: place them on the Stage, move them around, tween them in the Score, and so on.

There are only a few special considerations:

- You can use OLE cast members in movies on any system, but you can edit them only in Windows 95 and NT.
- Double-click an OLE cast member to launch the application that created it. When you make changes in the source application and choose Update from the File menu, the OLE cast member is updated within Director.
- OLE cast members are converted to normal bitmaps when you create a projector. They are no longer linked to their source applications.

{button See also,AL(`UsersGuide_167_help')}

Using Paste Special with OLE objects

Paste Special is the only way to create cast members that display selected portions of OLE objects instead of the whole object. This is useful for showing items like cells in a spreadsheet, or particular fields in a database instead of the entire record.

Begin by copying a selected portion of a document to the Clipboard in another application that supports OLE. Within Director, choose Edit > Paste Special > Using OLE. The OLE object appears as a cast member and is updated when the source document changes.

{button See also,AL(`UsersGuide_168_help')}

Launching external editors

You can specify external applications to edit bitmap, video, and sound cast members. Once you have set up an external editor for a particular media type, Director launches or switches to the other application when you double-click a cast member of that type. When you finish editing a cast member in an external editor and then close the file, Director re-imports the cast member.

If you know you will be using an external editor to work on a particular file, choose Include Original Data for External Editing from the Media pop-up in the Import dialog box when you import the file. With this option on, Director retains a copy of the original data. When you edit the cast member with an external editor, Director sends the original data to the external editor. This ensures that all of the editor's capabilities to modify the file are preserved. For example, if you specify xRes to edit PICT images, Director maintains all of the xRes object data.

None of the original data is included when you create a projector or a Shockwave movie, so you don't need to worry about it increasing the size of your final work. It only increases the size of the source file.

If you change an image in the Paint window and then edit the image with an external editor such as PhotoShop or xRes, the changes you made in the Paint window will be lost. Director warns you if this is a possibility. Registration points that you set in the Paint window are not lost if you later edit the image externally.

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. To edit externally, double-click the cast member or choose Edit > Launch External Editor.

{button See also,AL(`UsersGuide_169_help')}

To define external editors:

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 Use External Editor.**
5. **Click Browse or Scan to locate the application.**
   You can specify any application capable of editing the selected type of media.

Tip: 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. To edit externally, double-click the cast member or choose Edit > Launch External Editor.

{button See also,AL(`UsersGuide_170_help')}

Using movies within Director movies

You can import a Director movie into another movie as an internal or linked cast member. You can also play a Director movie in a window inside another Director movie.

{button See also,AL(`UsersGuide_172_help')}

Importing movies

You import Director movies just as you import other types of media. See the **Importing cast members** help topic. As with other media types, you can link to an external movie file, or import it so that it becomes internal media. Click the Link to File checkbox to link to an external movie file. The way you choose to import a movie affects its properties.

- For linked movies, cast scripts and behaviors (sprite scripts) function as before. Turn on Enable Scripts in the Cast Member Properties dialog box to make them work. 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, scripts do not work and the movie appears as a film loop.

For both types of imported movies, the host movie controls the tempo settings, palette settings, or transitions and settings in the imported movie are ignored.

Once it is imported, the entire movie appears as a single cast member in the Cast window. You can animate the cast member just as you would any graphic cast member, film loop, or digital video.

{button See also,AL(`UsersGuide_172_help')}

Playing a movie in a window

You can use a behavior or write a short Lingo script to make Director play another Director movie in a window. The movie can be played from a local disk, or streamed from the internet. A movie in a window retains any interactivity you've built into it. Tempo settings, palette settings, and transitions are lost.

Use a movie in a window as a floating controller for your movie, to display help text, or simply to show several movies at once.

You cannot use a movie in a window in a Shockwave movie. Web browsers do not allow Shockwave to open new windows.

Tip: A movie-in-a-window (MIAW) cannot take over the host movie's color palette. While preparing a movie for use as a movie in a window, choose Modify > Movie > Properties and turn on Remap Palettes When Needed. This option instructs Director to remap the movie's color palette to the host movie's color palette if there is a conflict.

You can create a basic movie in a window with the Open Movie in a Window behavior, and control it with the Message Movie in a Window behavior. For more information, see **Using included behaviors**.

For information about using Lingo to define and control movies in a window, see Chapter Ten, "Movies in a Window" in Learning Lingo.

{button See also,AL(`UsersGuide_173_help')}

Comparing film loops, digital videos, linked movies, and MIAWs

This information will help you decide which approach you want to use when you need to include a completed piece of animation in a Director movie.

To	Use Film loop	Digital video	Linked Director movie	Movie in a window
Include tempo settings		x		
Include palette settings		x		
Include transitions		x		
Include sounds	x	QuickTime only	x	x
Include interactivity	x		x*	x
Attach a cast member script to the movie's cast member	x	x	x	
Attach a sprite script to the movie's sprite	x	x	x	
Edit Score data from original animation	x		x	x
Move the movie across the Stage	x	x	x	x
Edit the movie		x**	x***	x

* Select Enable Scripts in the Cast Member Properties dialog box.

** Cut, copy, and paste individual frames in Director.

*** Close the current movie and open the linked movie in Director.

{button See also,AL(`UsersGuide_174_help')}

Understanding tempo settings

The tempo setting in the tempo channel is the speed Director tries to achieve. Many factors can make movies play slower than the specified tempo, such as:

- Animating several large sprites at the same time
- Animating sprites that have blend values
- Running the movie on a slower computer

You can always slow down animation with a tempo setting, but you can't make it go faster than the computer allows.

Tempo settings don't affect the duration of any transitions you've set in the transition channel, and they don't control the speed at which a sound plays. They control the maximum speed at which the playback head moves from frame to frame.

{button See also,AL(`UsersGuide_177_help')}

Using tempo channel settings with interactive elements

With settings in the tempo channel, you can make the movie pause while a sound or video finishes playing. While the movie is paused, some interactive elements in the movie such as buttons or moveable sprites don't work.

You can avoid this situation by controlling the tempo with Lingo. Lingo provides more sophisticated control of the speed of the movie while maintaining interactivity. For more information, see Learning Lingo.

For simple movies, these issues may cause no problem and using the tempo channel may be the best way to define tempos.

It's also a good idea to avoid setting a tempo, a transition, and a palette change all in the same frame, because it will be difficult to predict their playing order. Instead, duplicate the frame and assign a different effect to each frame.

{button See also,AL(`UsersGuide_178_help')}

Using the tempo channel

You enter tempo changes in the tempo channel at the top of the Score. 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 Lingo command. See Chapter 4, "Controlling Score Channels from Lingo," in Learning Lingo.

{button See also,AL(`UsersGuide_179_help')}

To add a tempo setting:

1. **In the Score, double-click the cell in the tempo channel where you want the new tempo setting to appear.**
   You can also choose Modify > Frame > Tempo.

2. **Select the option you want to use in the Frame Properties: Tempo dialog box.**

To	Do this
Set a new tempo for the movie	Use the arrows or drag the slider.
Pause the movie at the current frame for a specified amount of time	Use the arrows or drag the slider to change the Wait setting.
Pause the movie until the user clicks the mouse or presses a key	Click Wait for Mouse Click or Key Press.
Pause the movie until a cue point in a sound or digital video passes	See the **Using sound and video cue points** help topic.

3. **Click OK**.
   A number that matches the setting you've chosen appears in the tempo channel.

{button See also,AL(`UsersGuide_180_help')}

Using sound and video cue points

Use the Wait for Cue Point option in the Tempo dialog box to pause the playback head until a specified cue point in a sound or digital video is reached.

For example, you can use cue points to make a bullet point appear at the same time a voice-over reads text. First, use Macromedia Sound Edit 16 to place cue points corresponding to the bullet point text in the sound file. In Director, use the Tempo dialog box to pause the playback head at the frame where a bullet appears until the voice-over reaches the proper cue point.

For a demonstration of using cue points, see the **Synchronized Media** movie.

Use Macromedia SoundEdit 16 to define cue points in both sounds and digital videos. You can define cue points in AIFF, Shockwave Audio sounds, and QuickTime digital videos. QuickTime for Windows 2.5 or later is required to use cue points on the Windows platform.

SoundEdit 16 works only on the Macintosh, but it can place cue points in AIFF and QuickTime files for use on either Windows or Macintosh systems. AVI digital video does not support cue points.

There are also several Lingo commands that provide additional ways of using cue points. See chapter 8, "Controlling Sound and Digital Video," in Learning Lingo.

{button See also,AL(`UsersGuide_181_help')}

To use cue points:

1. **Place cue points in a sound file or in the audio track of digital video.**
   Use Macromedia SoundEdit 16 to define cue points in both sounds and digital videos.
2. **Import the sound or digital video into Director.**
   The sound can be imported internally using the Standard Import option, or linked to an external file with the Link to External File option in the Import dialog box. Cue points work the same way in both cases.
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.**
7. **Choose the cue point to wait for from the Cue pop-up.**
   Select the Start or End cue points, the Next cue point, or any named or numbered cue point sound or digital video.

When the movie plays, the playback head pauses at the frame until the cue point passes.

There are more advanced ways to use cue points with Lingo. See Chapter 8, "Controlling Sound and Digital Video," in Learning Lingo.

{button See also,AL(`UsersGuide_182_help')}

To compare the actual speed of a movie with the tempos you've set:

1. **Use the Step Forward button to step through the movie frame by frame.**

2. **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 (--).

If the difference between specified and actual tempo is too great, make changes to the movie to make it perform better. See "Performance tips" in chapter 8, "Completing Movies," of Using Director.

Using sounds

You can control sounds in Director using two basic methods:

- Use the sound channels at the top of the Director Score to add simple sound tracks to your movie. The sound channel controls the starting and stopping of sounds at particular frames in the movie. This approach is described later in this section.
- Control sounds more precisely and use more sounds at once with Lingo commands such as puppetSound, sound close, sound fadeIn, sound fadeOut, sound playFile, sound stop, soundBusy, soundEnabled, soundLevel, and the volume of sound. For more information, see Chapter 8, "Controlling Sound and Digital Video" in Learning Lingo or the **Lingo elements-sound** help topic.

{button See also,AL(`UsersGuide_185_help')}

Importing sounds

Director imports AIFF and WAV sounds, both compressed and uncompressed, and Macintosh System 7 sounds.

For best results, use sounds that are 8- or 16-bit depth, with a sampling rate of 44.1, 22.050, or 11.025 KHz.

Note: If your Macintosh has an audio input or microphone attached, you can record sounds by choosing Insert > Media Element > Sound. The Sound command opens the Macintosh sound recording dialog box. There is no equivalent function for Windows.

{button See also,AL(`UsersGuide_186_help')}

Using internal and external sounds

Internal sounds are sounds that have been imported to a cast (it doesn't matter whether the cast itself is internal or external). For internal sounds, Director stores all of the sound data inside the movie or cast file. For external sounds, Director maintains a link to an external file.

Director "streams" external sounds. Streaming sounds begin to play while the rest of the sound continues to load from its source, whether on disk or over the internet.

Director can stream only external sounds; it loads all internal sounds into RAM before playing them. It's best to link large sounds as external files so they can be streamed. Import smaller sounds that you want to play instantly as internal cast members.

Streaming sound is especially important for playing larger sounds in movies distributed on the internet. Shockwave Audio provides powerful compression and streaming capabilities. See the **Compressing and streaming sounds with Shockwave Audio** help topic.

{button See also,AL(`UsersGuide_187_help')}

Placing sounds in the Score

Place a sound in a sound channel just as you would create any other type of sprite. Drag a sound cast member to a frame in the sound channel or drag it to the Stage to place it in the first available sound channel for the current frame.

Sounds play only as long as the playback head is in the frame containing the sound sprite. Once a sound begins playing, it plays at its own speed. Director cannot speed up or slow down sounds.

Note Do not place a streaming external Shockwave Audio sound in the sound channels. Place it in a numbered sprite channel. For more information, see **Streaming external Shockwave Audio files**.

New sound sprites are assigned the default sprite length defined in Sprite Preferences. You may need to adjust the length of the sprite to make sure the sound plays completely, or use a tempo setting to make the Playback Head wait for sound to finish playing. For more information, see **Using sound and video cue points**.

The Score displays two channels for sound. Director, however, can mix additional sound channels simultaneously, but the additional channels are accessible only from Lingo or by using sounds in digital videos. RAM and processing power are the real constraints for the number of sound channels Director can use effectively.

Note: In Windows, the default number of sounds that Director can mix is four. This can be increased by modifying the value for MinMaxChannels in the Director.ini file in the Director folder. For multiple sounds to play simultaneously, the Macromix.dll file must be installed on your computer.

In Windows, a sound that's already playing in either sound channel overrides the sound in a video and 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.

Tip: Sounds can take up considerable disk space. If you know you're going to use the same sounds in several movies, put the sounds in a shared external cast or use linked external sound files. You can then use the same sounds in several movies and use much less memory. For information about creating an external cast, see the **Creating casts** help topic.

{button See also,AL(`UsersGuide_188_help')}

Repeating a sound

You may find that you want to repeat a sound, such as a footstep, over and over to create one continuous sound effect-for example, 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.

To make a sound repeat, select the Looped checkbox in the Cast Member Properties dialog box for the sound.

Tip: In Director, sound in the last frame of a movie continues to play or loop until the next movie begins or you exit the application. This sound can be a useful transition while Director loads the next movie. You can stop the sound using the Lingo puppetSound 0 command, the sound stop Lingo command, or by using the MCI stop command when an MCI device controls the sound.

{button See also,AL(`UsersGuide_189_help')}

Setting sound volume

Sound volume settings in Windows work differently than on the Macintosh. On the Macintosh, a sound volume set too high produces a loud but clear sound. In Windows, high sound volume settings usually produce distorted sound.

The sound volume in Macintosh movies is commonly set to the highest value in an attempt to get loud, clear sound. For best results when the movie plays back in Windows, avoid setting the sound volume with Lingo from within the movie. Instead, assume that the user has set the appropriate Windows sound volume. If you do set the sound level from within the movie, use a medium sound volume setting.

In Director for Windows, the sound volume setting of QuickTime for Windows videos is relative to the WAVE driver sound volume setting. As a result, many QuickTime for Windows videos can play back louder or softer than the intended sound volume.

To control the volume of a QuickTime for Windows video when the movie plays back in Director for Windows, use the volume of sprite command.

{button See also,AL(`UsersGuide_190_help')}

Synchronizing sound

You can synchronize sounds to the rest of your movie by making the playback head wait for cue points in sound files. See the Using sound and video cue points help topic. There are also many ways to synchronize sounds with movie events using Lingo. See Chapter 8, "Controlling Sound and Digital Video," in Learning Lingo.

Compressing and streaming sounds with Shockwave Audio

Shockwave Audio compresses sound and streams it over the internet or off disk. Its compression technology makes sounds a small fraction of their original size without sacrificing fidelity.

"Streaming" sound means that Shockwave Audio begins playing while it is still downloading. When used properly, Shockwave Audio's compression and streaming capabilities provide almost instant playback of high-quality audio, even for users with 28.8 modem connections to the internet.

Shockwave Audio can compress both internal and external sounds. Internal sounds have been imported to a cast. External sounds are linked sound files that remain separate from the movie file.

Director can stream only external Shockwave Audio files; it loads internal sounds into RAM before playing them. It's best to convert large sound into the Shockwave Audio format so they can be streamed. Import smaller sounds as internal cast members.

{button See also,AL(`UsersGuide_192_help')}

Understanding compression quality

Shockwave Audio compresses sounds using advanced methods that are quite different from most other sound compression schemes. Some of the techniques you currently use to make sounds smaller may be inappropriate for use with Shockwave Audio.

There is often no advantage to reducing the sampling rate of source sounds before compressing with Shockwave Audio. Reducing the sampling rate degrades the quality of the final compressed sound and makes no difference in its size.

When sound is decompressed, it returns to its original size, so make sure internal sounds are not too large to decompress in available memory. The best audio settings to use for general purpose source material are 16 bit, 22 kHz mono. Internal sounds have to be loaded into RAM and decompressed before playing. Larger sounds should be converted to external Shockwave Audio files and streamed.

Set the amount of compression for Shockwave Audio by choosing a bit rate setting in any of the Shockwave Audio Xtras. The bit rate specifies the number of bits per second Shockwave Audio uses to render the sound, regardless of the source material. One second of any type of sound compressed to 16 Kbps = 16,000 bits = 2 K bytes. The bit rate does not specify the compression ratio or the sampling rate.

For example, 10 seconds of 16 bit audio sampled at 22 Megahertz would be 440K in the AIFF format. Compressed at 16 Kbps by Shockwave Audio, the same sound would end up being 20 K.

(Do not confuse Kilo bits per seconds (Kbps) with Kilobytes (K). A byte is eight bits. Kilo bits per second is commonly used to indicate the speed of data transmission; Kilobytes is used for fize sizes. This is, unfortunately, the accepted industry terminology.)

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. Try compressing the same sound at several different bit rates to see how the sound changes and how much compression is achieved.

Choose the bit rate appropriate for the intended delivery system (modems, ISDN, CD-ROM, hard disk, and so on), the type of movie, and the nature of the sound itself. A voice-over, for example, may not need to be as high quality as music. Test the sound on several systems to find the right balance between quality and performance.

The more compressed a sound is, the better it streams. If you choose to use a high bit stream rate, a slow delivery system may not be able to send the data fast enough. This causes gaps during playback. Most developers choose 16 Kbps for the best results for streaming over the internet with 28.8 modems. For smaller internal sounds, choose 64 Kbps for the best balance of size and quality.

The table below 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.

Delivery	Bit rate	Quality
T1	64-128 Kbps	Equal to source material
ISDN or CD-ROM	32-56 Kbps	FM stereo to CD
28.8 modem	16 Kbps	FM mono or good quality AM
14.4 modem	8 Kbps	Telephone

Note: Any sound compressed at less than 48 Kbps is converted to mono.

The technology for delivering audio over the internet is developing rapidly. For the latest information, see the Director Developers Center web site.

{button See also,AL(`UsersGuide_193_help')}

Compressing internal sounds

Shockwave Audio can compress any internal sounds in a movie. (If you're unfamiliar with internal sounds, see the **Using internal and external sounds** help topic.) Although internal sounds are not streamed, compressing them with Shockwave Audio dramatically decreases the size of the movie, shortens the download time from the internet, and saves disk space.

Use the Shockwave Audio Settings Xtra to specify compression settings for internal sound cast members. The compression settings you choose apply to all internal sound cast members. You can't specify different settings for different cast members.

Shockwave Audio works only with compressed Director movies. You can choose compression settings at any time, but compression occurs only when the Director movie is compressed with either the Create Projector, Save As Shockwave Movie, or Update Movies command.

When you distribute a movie that contains sounds compressed with Shockwave Audio, you must include the required Xtras to decompress and play the sounds. In most cases, Director handles this automatically. For more information on packaging Xtras, see the **Managing Xtras for distributed movies** help topic.

{button See also,AL(`UsersGuide_194_help')}

To compress internal sound cast members:

1. **Choose Xtras > Shockwave for Audio Settings.**

2. **Choose Enabled to turn on compression.**

3. **Choose a setting from the Bit Rate pop-up.**
   The bit rate determines both file size and quality of output. It specifies the number of bits per second Shockwave Audio uses to render the sound. The lowest available setting, 32 Kbps, provides the best compression possible without noticably altering the original sound.

   See the **Understanding compression quality** help topic if you're not sure what to choose.

4. **Click the Normal or High option for the Accuracy setting.**
   High provides noticably better quality when lower bit rates are used, but it takes more time to process during compression. Unless sound quality is critical to your work, leave this option set to Normal.

5. **Click Convert Stereo to Mono if you want to convert a stereo file to mono.**
   At rates lower than 48 bits, all sounds are converted to mono.

6. **Click OK to close the dialog box**.
   Director doesn't compress the sound cast members until you create a projector or a Shockwave movie. When creating a projector, Director only compresses sounds 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.

Note: Shockwave Audio does not compress IMA compressed sounds.

{button See also,AL(`UsersGuide_195_help')}

Streaming external Shockwave Audio files

Director streams external sounds that have been compressed with Shockwave Audio, either from a local disk or a URL. Create external Shockwave Audio files with the Create Shockwave Audio File Xtras for Director (Windows only) or SoundEdit 16 (Macintosh only).

To stream an external Shockwave Audio sound:

1. **Choose Insert > Media > Shockwave Audio.**

   This creates an Xtra cast member that controls the streaming Shockwave Audio.

   The Cast Member Properties Options dialog box appears.

2. **Enter a URL in the Link Address box or click Browse and choose a Shockwave Audio file on a local disk.**
   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 according to your needs and click OK when done**

Use this option	To
Volume	Set the volume of the sound.
Sound Channel	Choose the sound channel for the sound. For sounds in cross-platfrom movies, choose Any.
Preload Time	Specify the size of the stream buffer. Director attempts to load enough sound data to play for the specified time. Increasing this time can help avoid gaps in playing sounds on slow or halting 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.**

Streaming Shockwave Audio files don't work in the sound channels. The sound streams from the source location when the movie plays. To make sure the sound finishes, use a tempo channel setting to make the playback head wait for the end of the sound.

Shockwave Audio technology is evolving rapidly. For the latest information, check the Director Developers Center web site.

{button See also,AL(`UsersGuide_196_help')}

Working with transitions

Director provides dozens of transitions. For example, you can dissolve from one scene to the next, display a new scene strip by strip, or switch to it 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.

You can add new transitions to Director as Xtras. Xtra transitions appear with special icons in the Frame Properties: Transitions dialog box. Install Xtra transitions by placing them in the Xtras folder in the Director application folder. The Transition Xtras must be present when the movie runs.

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.

{button See also,AL(`UsersGuide_197_help')}

Creating transitions

Transitions, like tempos, palettes, and sounds, have a channel set aside for them in the Score.

Before you select the frame in which you want to set a transition, it's important to understand how transitions work. 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 do a dissolve between two scenes, set the transition in the first frame of the second scene, not the last frame of the first scene.

{button See also,AL(`UsersGuide_198_help')}

To add a transition:

1. **In the Score, go to the transition channel and 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.**
   The Frame Properties: Transition dialog box appears. It lists the transitions you can choose from.

   Scroll to the transition you want to select in the Transition dialog box by typing the first letter of the transition's name.

3. **Select the transition you want.**
   For many of the transitions, there is a default setting for Duration and Smoothness. You can adjust the sliders to change the settings.

   For many of the transitions, you can also select whether the transition affects the entire Stage or just the area that's changing (the area where any sprites that weren't on the Stage in the previous frame appear).

   Xtra transitions might offer extra options provided by the developer. If the Options button is available when you choose an Xtra transition, click the Options button to view and change the transition options.

4. **Click Set.**
   Director displays the number that corresponds to the transition in the transition channel. The transition also appears in the cast.

{button See also,AL(`UsersGuide_199_help')}

Tips for using 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.
- Palette transitions (including fade to black and fade to white) and bit and pixel dissolves do not work in 16-, 24-, and 32-bit environments. These features require that the monitor is set to display 256 colors ("Hundreds of Colors" on the Macintosh).
- If you export a movie that contains transitions as a digital video or PICS file, the transitions might not be preserved.
- A transition that occurs while a sound is decompressing may require more system resources than available on less powerful systems. This may cause the sound to stop playing. If you notice this while testing on low-end systems, try making the transition shorter or not using more complex transitions like Dissolve.

{button See also,AL(`UsersGuide_200_help')}

Changing color palettes

By switching between palettes in a movie you can create a number of color effects without animation. For example, you can simulate the sun setting by switching to a palette with dimmer colors.

Palettes only work if your system is set to display 256 colors ("Hundreds of Colors" on the Macintosh). Palette changes and effects may not work for movies playing in web browsers, because they do not control the system palette the way a stand-alone movie does. See the **Managing color palettes for browsers** help topic.

You change the active palette in a movie by placing a new palette in the palette channel.

Tip: You can also switch palettes with the puppetPalette command. Refer to the **Lingo** help topic.

You can edit color palettes in the color palettes window.

For conceptual information about color and using color palettes in Director, see the **Understanding color palettes** help topic or the **Color Palettes** movie.

Note: Don't change palette colors that are used by your system software for interface elements, especially if your movie will be used at the same time as other applications. On Windows these colors always appear as the first ten, and last ten, colors in the palette.

{button See also,AL(`UsersGuide_201_help')}

Changing palettes in a movie

The palette channel in the Score determines which palette is active at any point in a movie.

Use the palette channel to create color effects and specify palette transitions that make the change of palettes less abrupt. A palette transition can mask a change from one palette to another by fading the screen to black during the change. It can also gradually change the palette between frames or over a series of frames, shifting every color gradually through a range of intermediate colors to its counterpart in the new palette.

You can also use a palette setting to cycle through a range of colors you've selected in a palette. Cycling colors is a great way to represent flowing, spinning, or pulsing objects.

If you place a cast member with a different palette on the Stage-and it's the first cast member with a different palette in the frame-Director automatically adds 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 on an 8-bit monitor at any one time. You may see a palette flash if you try to display bitmaps with more than one palette.

{button See also,AL(`UsersGuide_202_help')}

To create a palette setting in the palette channel:

1. **In the Score, select the frame in the palette channel where you want the new palette setting to take effect.**

   To extend a palette transition over time, select the range of frames during which you want the transition to take place.

   You must select at least four frames to make a palette transition span selected frames.

2. **Choose Modify** > Frame > Palette.
   You can also double-click a cell in the palette channel.

   When the Frame Properties: Palette dialog box appears, you have many options:

To	Do this
Set a new palette without any transition	1. Choose a new palette from the pop-up. 2. Click the Palette Transition option. If you've added any palettes to the cast, they appear in the Palette pop-up.
Hide a palette change within a fade	1. Choose a new palette from the pop-up. 2. Click the Palette Transition option. 3. Select Fade to Black or Fade to White. 4. Use the Rate slider to set the speed of the fade.
Stop the movie while the palette changes	1. Choose a new palette from the pop-up. 2. Click the Palette Transition option. 3. Select Between Frames. 4. Use the Rate slider to set the speed of the transition. Any animation stops during this type of palette transition.
Extend a palette transition over time	1. Choose a new palette from the pop-up. 2. Click the Palette Transition option. 3. Choose Span Select Frames. Animation continues during this type of palette transition. This option is only available if you first select a range of frames in the palette channel.
Cycle the colors in a selected range of the palette.	See the **To use color cycling** help topic.

3. **Click Set.**
   The palette you chose now appears in the cell you selected in the palette channel of the Score. It remains in effect in the movie until you set a different palette in the palette channel.

Tip: Since the palette you choose affects everything displayed on the monitor-including Director's interface-you may find it difficult to see what you're doing after you select a new palette. Use General Preferences on the File menu to turn on the Classic Look (Monochrome) option for the user interface. This will make the interface controls black and white.

You can get a better idea of how the palette transitions and color cycling works if you open the Palette window and observe the transition there.

{button See also,AL(`UsersGuide_203_help')}

To use color cycling:

1. **Create cast members using a specific range of colors in the current color palette.**

   It's important that other cast members that will be on the Stage at the same time not use these colors, unless you want them to cycle also.

2. **Create sprites in the Score using the cast members you created earlier.**
3. **In the palette channel, choose the range of frames in which the sprites you want to cycle appear and then choose Modify > Frame > Palette.**
4. **Choose the palette to cycle from the Palette pop-up. (You don't have to choose a new palette to cycle colors.)**
5. **Click the Color Cycling option.**
6. **Select the colors to cycle in the palette shown on the left side of the dialog box.**
7. **Choose Between Frames or Span Selected Frames.**
   - If you choose Between Frames, use the Rate slider to set the speed of cycling. Colors cycle completely during each frame in the transition.
   - If you choose Span Selected Frames, the cycle occurs only once across all selected frames.
8. **Enter the number of cycles to complete in the Cycles box.**
9. **Choose Auto Reverse or Loop to specify the sequence of colors.**
10. **Use the Color Palettes window to reserve the colors used for cycling.**
    This prevents new cast members from using the colors that cycle. See the Selecting and reserving colors in the **Using the Color Palettes window** help topic.

Color cycling is an excellent way of creating simple movement in a movie without doing any real animation or using system resources. It's especially effective for creating flashing banners, water effects, and fireworks.

{button See also,AL(`UsersGuide_205_help')}

Using the Color Palettes window

Use the Color Palettes window to change and rearrange color palettes.

To open the Color Palettes window:

- Choose Color Palettes on the Window menu.
- Press Control-Alt-7 (Windows) or Command-Option-7 (Macintosh).

All the functions in the Color Palettes window involve changing the currently active color palette. You choose the active palette by selecting a palette from the pop-up.

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 Movie Properties on the Modify menu to choose the movie color palette, or use Frame Palette on the Modify menu to change the color palette at a particular frame.

When you change 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 change a palette already used by cast members in the movie, use the following procedure.

{button See also,AL(`UsersGuide_206_help')}

To edit a palette already used in a movie:

1. **Duplicate and rename the palette.**

2. **Edit the palette.**
   Use any of the methods discussed later in this section.

3. **Select all the cast members that use the old version of the palette.**
   Use Find to locate all cast members using a particular palette.

4. **Choose Modify > Transform Bitmap and remap all the cast members to the new palette.**
   Be sure to remap and not dither.

{button See also,AL(`UsersGuide_207_help')}

Selecting and reserving colors in the Color Palettes window

When making fine adjustments to the colors in cast members, you often need to select colors that appear in images. You also need to select colors in order to reserve them. Reserve colors you intend to use for cycling to prevent them from being used by non-cycling cast members. Reserved colors are not used in transform bitmap and import remapping of cast members to new palettes. They are also not used for gradients. Reserved colors appear striped in the color palette.

To	Do this
Select colors	1. Click them in the Color Palettes window. If the selection arrow is not active, click the selection arrow tool at the bottom of the window. 2. Drag across colors or Shift-click to select a range. Control-click (Windows) or Command-click (Macintosh) to select multiple discontinuous colors.
Match the color of any pixel on the Stage with the same color in the palette	1. Click the Eyedropper tool. 2. Select any color in the Color Palettes window. 3. Without releasing the mouse button, drag to any point on the Stage. The selection in the Color Palettes window and the foreground color in the Tools palette change to the color at the pointer location.
Reserve selected colors in the palette	1. Select the colors you want to reserve. 2. Click the Reserve Selected Colors button.
Select all reserved colors in the palette	Click the Select Reserved Colors button. To make all the reserved colors available again, click the Select Reserved Colors button and choose All Colors Available in the dialog box.
Select colors in the palette used in the current cast member	1. Select a cast member or open it in the Paint window. 2. Click the Select Used button.
Select all colors not currently selected	Click the Invert Selection button.

{button See also,AL(`UsersGuide_208_help')}

Rearranging and blending colors in the Color Palettes window

To	Do this
Move colors within the palette	Click the Hand tool and drag the selected colors to a new location.
Sort the selected colors in the palette by hue, saturation, or brightness	1. Select a range of colors and then click the Sort button. 2. When the Sort dialog box appears, choose Hue, Saturation, or Brightness. It's best to sort colors in a palette before using the palette in a movie. Cast members that use the palette will also change color as the colors are sorted. You can fix this by remapping to the sorted palette with Transform Bitmap.
Reverse the order of the selected colors	Select the colors you want to reverse and then click the Reverse Colors button. The colors themselves remain unchanged.
Cycle selected colors one position to the left	Select a range of colors and click the Cycle button. The leftmost color wraps around and appears at the last right square. Each time you click the Cycle button, the selected colors shift by one more square. It is the same as the movement of colors you see while colors cycle with the Cycle ink.
Create a blend of the first and last colors of a selected range	Select a range of colors in the Palette window and then click the Blend button.

{button See also,AL(`UsersGuide_209_help')}

Setting colors

You can define a new color in 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 controls (the Windows Color dialog box or the Macintosh Color Picker).

To edit selected colors in the Color Palettes window using the HSB system, click the arrows at the bottom of the window to increase or decrease the value of hue, saturation, or brightness.

- 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 even 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 the brightness is reduced, the color gets darker as though more black were added. If brightness is reduced to 0, then no matter what the values are for Hue or Saturation, the color will be black.

For instruction on using the Windows Color dialog box, or the Macintosh Color Picker, see your system documentation.

{button See also,AL(`UsersGuide_210_help')}

Printing movies

You can print a movie while in authoring mode in a variety of ways. You can print an image of the Stage, 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 all of the Cast window.

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

If you're printing the Stage, click the Options button to specify storyboard printing format.

{button See also,AL(`UsersGuide_211_help')}

Preparing a movie for distribution

Distribute movies either as projectors, Shockwave movies (DCRs), or protected movies (DXRs). You should not distribute source movies (DIRs) unless you want your users to be able to change the movie in the Director authoring environment.

- Use projectors for any movie distributed on disk. A projector is a self-running version of a movie. Projectors contain all the software required to play a Director movie, but none of the software to edit the movie. You can package several movies, external casts, Xtras, and linked media in a single projector. You can also compress the movie data within a projector using Shockwave compression. Projectors appear in the system desktop as applications.
- Use Shockwave movies (DCRs) to distribute on the internet for playback in a web browser. You can also use them when you want to compress movies distributed on disk that are not contained in a projector. Saving a movie as a Shockwave movie removes all the information needed to edit the movie and does not include the software that plays the movie. Therefore, Shockwave movies can only be played in a web browser with the proper Shockwave player installed, or by a projector.
- Use protected movies (DXRs) when you want to distribute uncompressed movies on disk, but you still want to prevent users from editing the source file. Protected movies load faster than Shockwave movies from 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 needed to edit the movie or the software that plays the movie. They can only be played by a projector.

You cannot edit a projector, Shockwave movie, or protected movie in Director. You must edit the source file and then create a new movie in one of the distribution formats. Always save your source files.

{button See also,AL(`UsersGuide_213_help')}

Distributing movies on disk

Whenever a movie plays from 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 the movie was created. The same is true for Xtras. To make sure you don't forget any linked media or Xtras when you distribute a movie on disk, place linked files in the same folder as the projector, or in a folder inside the Projector folder.

{button See also,AL(`UsersGuide_214_help')}

Distributing movies on the internet

When distributing a movie on the internet for playback in a web browser, all linked media must be in the specified URLs when the movie plays. An HTML document must include the correct tags to run the movie.

{button See also,AL(`UsersGuide_215_help')}

Distributing movies on a local network

If you plan to distribute a movie on a local area network (LAN) such as a Novell network, 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.

{button See also,AL(`UsersGuide_216_help')}

Organizing movie files

In most cases, you should divide a larger 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 change if they are organized in discrete sections.

The best way to organize a larger disk-based 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 recreating the projector every time you change one part of a 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 avoids downloading time for parts of the movie that may not be used.

The size of your movie may be less of an issue if you properly use progressive play. See the **Setting streaming playback options** help topic.

{button See also,AL(`UsersGuide_217_help')}

Creating 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.

Certain Xtras must be included with a projector for it to be able to retrieve media from the internet during playback. You can include these Xtras automatically with the Include Network Xtras in the Projector Options dialog box. Shockwave movies do not require these Xtras.

Note: You can only include Director 6 movies in projectors. Use Update Movies to convert older movies to the latest version of Director. See the **Processing movies with Update Movies** help topic.

{button See also,AL(`UsersGuide_218_help')}

To create a projector:

1. **Choose Create Projector from the File menu.**

   The Create Projector dialog box appears.

2. **Double-click movies, external casts, or Xtras to include in the projector.**
   Director transfers the name of the selected items to the file list. You don't need to manually select every Xtra required by the movie. As explained later in this procedure, the Check Movie for Xtras option automatically adds most required Xtras to the projector.

   Click Add All to include all the movies in the open folder.

3. **Use the Move Up and Move Down buttons to arrange the movies in the proper order.**
   Be sure to place the starting movie at the top of the list. If the Play Every Movie option in the Projector Options dialog box is on, Movies play in the order they appear on the list. If the 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.

4. **Click Options.**
   The following projector options are most important. Director retains these settings once you define them, so you don't have to do this every time.

   - Select the type of computer the projector will run on.

Windows projector options	**Runs on**
Windows NT and 95	Windows NT and 95
Windows 3.1	Windows 3.1-also runs on Windows 95 and NT, but slower than native versions
Macintosh projector options	**Runs on**
Power Macintosh Native	Power Macintosh only
Standard Macintosh	Macintosh systems older than the Power Macintosh-also runs on the Power Macintosh, but is slower than native versions
All Macintosh Models	All Macintosh systems at optimal speed, but the projector file is much larger

   - If your movie uses any Xtras, choose Check Movie for Xtras. This option ensures that all Xtras listed in the Movie Xtras dialog box are included in the projector. See the **Managing Xtras for distributed movies** help topic.
   - If you want to compress the projector's movie data in the Shockwave format, click Compress.

     This makes the projector smaller but increases the load time.

5. **Click OK once all projector options are set.**

6. **Click Create in the Projector dialog box and then enter a name and location for the projector.**
   To avoid problems with linked media, it's best to create the new projector in its final folder location and not move it to a different folder after creation.

   Director turns the movies, casts, and Xtras you've selected into a single projector.

Use the Update Movies command on the Xtras menu to compress movies that are part of a production but are not in a projector. See the **Processing movies with Update Movies** help topic.

{button See also,AL(`UsersGuide_219a_help')}

Creating Shockwave movies

Save your work as a Shockwave movie to prepare it for playback in a web browser with Shockwave, or to make disk-based movies smaller. Shockwave movies also prevent users from editing the movie if they own Director.

Unlike projectors, a Shockwave movie does not have to be made for a particular platform. Any Shockwave movie plays on all compatible platforms, provided users have installed the correct version of Shockwave in their web browser.

If the Shockwave movie you're creating will be distributed on the internet and requires any Xtras, make sure the user can download the Xtras from your web site into the Support folder of their browser. See the **Managing Xtras for distributed movies** help topic.

Save as Shockwave Movie does not work like other Save As commands. The new Shockwave movie file is created on disk, but you continue to work on the original DIR file. You cannot edit a Shockwave movie file.

Tip: You can play a movie in a browser without creating an HTML page, or even saving the movie as a Shockwave movie. Just drag the movie from the system desktop to a web browser window.

{button See also,AL(`UsersGuide_219b_help')}

To create a Shockwave movie:

1. **Open the movie you want to save.**
2. **Choose File > Save as Shockwave Movie.**
3. **Enter a name and location for the new file.**
   To avoid problems with linked media, you should always save new Shockwave movies in the same folder as the original DIR file.

   Director automatically adds the DCR extension. Shockwave only plays movies with the DCR or DIR extensions.

Unlike projectors, a Shockwave movie does not have to be made for a particular platform. Any Shockwave movie plays on all compatible platforms, provided users have installed the correct version of Shockwave in their web browser.

Tip: Use Update Movies to convert several movies at once to the Shockwave format. See the **Processing movies with Update Movies** help topic.

{button See also,AL(`UsersGuide_220_help')}

Managing Xtras for distributed movies

If certain types of Xtras are used in a movie, the Xtras must be present when the movie runs. These include:

- Xtras that create and manage cast members (Button Editor, QuickDraw 3D, QuickTime VR, and so on)
- Shockwave Audio Xtras
- Transition Xtras
- Importing Xtras, if the movie uses any type of linked external cast members
- Network Xtras required for a projector to access the internet
- Lingo Xtras

Determining which Xtras are required

Choose Modify > Movie > Xtras to see which Xtras are required by your movie. The Movie Xtras dialog box shows any Xtras that control or import cast members referenced in the Score.

If you use Xtras that are referenced only in Lingo, they do not show up in Movie Xtras. Use the Add button in the Movie Xtras dialog box to add these Xtras to the list.

Any Xtra listed in Movie Xtras will be automatically included with a projector, as long as the Check Movie for Xtras option in the Projector Options is turned on.

Including importing Xtras

Director uses different Xtras to import media of each type. The Xtras that import the media types for all linked cast members must be present when a movie runs. The Shockwave players for Netscape Navigator and Microsoft Internet Explorer include the Xtras required to import the following media:

- GIF and JPEG graphics
- AIFF (compressed and uncompressed), Shockwave Audio, and WAV (uncompressed only) sounds.

Shockwave movies playing in web browsers can import these media types without first downloading importing Xtras. To import other types of media, the required Xtras must first be downloaded and installed.

For projectors, the Xtra for each type of file being imported must be included with the projector. In most cases, Director handles this automatically when you turn on Check Movie For Xtras option in the Projector Options dialog box.

{button See also,AL(`UsersGuide_221_help')}

Processing movies with Update Movies

Use the Update Movies command on the Xtras menu to:

- Update movies and casts from Director 4 and later to the latest file format.
- Compress movies for faster downloading from the internet.
- Remove redundant and fragmented data in movie and cast files. Save and Compact, and Save As 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 4.x and 5.x files to the latest file format. (See the **Converting existing movies** help topic for details on how movies are converted.)

At the end of a project, use this command to compress all your movies and casts at once.

Update Movies copies the old files to a new folder and creates the new updated or compress files in the original location. This avoids problems with linked media.

Note: A Shockwave movie can only be played by a web browser with the Shockwave player installed, from a projector, or as a movie in a window. To play a protected movie from a projector, you must use Lingo to go to or play the protected movie. Protected casts can only be opened by protected movies.

{button See also,AL(`UsersGuide_225_help')}

To update and compress movies and casts:

1. **Choose Update Movies from the Xtras menu.**

   The Update Movies dialog box appears.

2. **Choose one of the options for Action.**
   - Update converts movies from Director 4 or later to the latest file format. As it updates movies, Director consolidates and removes fragmented data, the same as when you use Save As. (To update movies from older versions, you must first convert them to the Director 4 file format.)
   - Protect removes all the data required to edit the movie, but does not compress it further.
   - Convert to Shockwave Movie(s) rewrites movies in the compressed Shockwave file format and adds the DCR extension. This options also prevents users from opening the movie or cast and making changes. Once a movie is compressed, there is no way to "decompress" it, so be sure to keep the original movie.

3. **Choose one of the options for Original Files.**
   - Back Up Into Folder specifies that the original files should be placed 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 original files should be overwritten by the newly updated files. Be very careful using this option. Once a file is converted, you cannot open it again in Director.

4. **Click OK.**
   A dialog box appears from which you select 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.

   Choose Add All Includes Folders before you click the Add All button to make it 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, recreating their original folder structure. If you didn't specify a folder for the original movies, Director prompts you to select one.

   Director adds a DCR extension to Shockwave movies, and CCT to external casts in the Shockwave format. Protected moves have the DXR extension and protected casts have the CXT extension.

{button See also,AL(`UsersGuide_226_help')}

Converting existing movies

Director 6 can convert movies from Director 4 and 5. You can update movies to Director 6 by simply opening and saving them, but Update Movies is faster for converting large projects. It also is more effective for preserving links to external media.

Note: The Director 6 Shockwave plug-in for Netscape and the controller for Internet Explorer can play Shockwave movies created with Director 4 and 5.

Director 5 movies

When you open a Director 5 movie in Director 6, or convert it to the new format with Update Movies, Director converts the old Score data to the new Score. It combines adjacent frames in the old Score containing the same cast members into single sprites in the new Score. The movie will run exactly as it did in Director 5, but you may want to split or join sprites to make working in the Score more convenient. For more information about using old movies in Director 6, see the **Using Director 5 Score options** help topic.

Converting Director 4 movies

You must use Update Movies to convert any Director 4 movies with a shared cast (Shared.dir) to Director 6. To update movies from older versions (pre-4.0), you must first convert them to the Director 4 file format.

Using Update Movies makes the following changes to movie files from Director 4:

- Converts the movie into a Director 6.0 format file.
- Converts a shared cast (Shared.dir) to a linked external cast named Shared.cst. It renumbers the cast members so they begin at 1 and updates all Score references to the new numbers. Make sure you select Shared.dir as one of the files to be updated while using Update Movies.
- Places transitions in the cast.

Note: Lingo from pre-4.0 versions that was allowed in Director 4 may not work in Director 6. If error messages inform you of script errors during Update Movies, this may be the problem. Update Movies converts movies with old Lingo to the new file format, but they will probably not run. To use these movies in Director 6, you have to find and change the old Lingo. For information on outdated Lingo, see "Using outdated Lingo" in the introduction to Learning Lingo.

{button See also,AL(`UsersGuide_227_help')}

Including required system elements

Depending on the type of project you are creating, you may have requirements for special system utilities that are not part of the project but must be present on the computer before your project can run correctly.

Fonts

Since text cast members are converted to bitmaps when a movie plays, you don't need to worry about what fonts are installed on other systems. For more about text cast members, see the **Using different types of text in Director** help topic.

If your project's field cast members require non-system fonts, you need to provide the font files and installation instructions for installing them. Your users will have to run an installer (either an installer that you create, or one that was provided by the font manufacturer) before they can run your projector.

Any installer you use will retrieve the fonts from your CD-ROM or floppy disks and install them in the appropriate location on the user's system. There is no requirement to store any of these font-related materials in the same folder as the projector.

For information on managing fonts for Shockwave movies, see the **Mapping fonts on the internet** help topic.

Note: If you distribute font software with your movie, you must honor all licensing agreements specified by the font manufacturer. You can also create your own fonts with Fontographer from Macromedia.

Video for Windows and QuickTime

If your project uses Video for Windows or QuickTime for Windows or Macintosh, you should inform the users to whom you distribute it that they need to install the current version of Video for Windows or QuickTime before they can run your project.

You may wish to simply leave the task of acquiring Video for Windows and QuickTime to users. However, you may decide to provide users with some type of installer.

To distribute Video for Windows Runtime, provide the entire folder that you received when you installed Director for Windows.

{button See also,AL(`UsersGuide_228_help')}

Video for Windows and QuickTime

If your project uses Video for Windows or QuickTime for Windows or Macintosh, you should inform the users to whom you distribute it that they need to install the current version of Video for Windows or QuickTime before they can run your project.

You may wish to simply leave the task of acquiring Video for Windows and QuickTime to users. However, you may decide to provide users with some type of installer.

To distribute Video for Windows Runtime, provide the entire folder that you received when you installed Director for Windows.

Naming files and folders

Macintosh and Windows have different file name and folder name conventions. Windows 95 and NT support and the Macintosh operating system support 32 character file names. For Windows 3.1, the file or folder name must be eight or fewer characters followed by an optional three-character extension (referred to as 8.3 format). Windows 95, NT, and 3.1 file names cannot include any of the following characters:

asterisk (*)	semicolon (;) (Windows 3.1 only)
less than (<)	double quotation mark (")
backslash (\)	slash (/)
period (.) (Windows 3.1 only)*	equal sign (=) (Windows 3.1 only)
brackets ([]) (Windows 3.1 only)	space ( ) (Windows 3.1 only)
plus sign (+) (Windows 3.1 only)	greater than (>)
colon (:)	vertical bar (|)
question mark (?)	at sign (@)
comma (,) (Windows 3.1 only)	characters with ASCII values less than 32

* Windows 3.1 interprets periods differently than the Macintosh does. In Windows, a period indicates the beginning of the file extension. This causes difficulty when names are converted for Windows.

If you are likely to distribute movies to both Windows 3.1 and Windows 95 users, it is best to adopt the Windows 3.1 naming conventions.

Tip: The best way to ensure that Director for Windows locates movies and external resources is to assign Macintosh files and folders names that are acceptable to Windows. That way, the file names and pathnames are consistent and usable when your movie plays back on Windows systems. Director for Windows does not convert file names.

{button See also,AL(`UsersGuide_230_help')}

Choosing file names and folder names for Windows 3.1

Follow these rules when assigning names to files and folders:

- Use eight or fewer characters, followed by an optional period and a three-character extension, for example, Mymovie.dir.
- Don't use a period within the Macintosh file name or folder name.
- Don't use any of the characters that are unacceptable in Windows 3.1 file or folder names. Unacceptable characters are listed in the previous section.

For example, these Macintosh file names can be used on Windows 3.1: Mymovie, Movie1, Movie2.

{button See also,AL(`UsersGuide_231a_help')}

Assigning pathnames

Director uses pathnames to locate files for external resources-such as linked cast members-and in Lingo statements that refer to external files.

On all systems, there is a maximum number of folder levels. Avoid using folders more than eight levels from the root. Also, hundreds of file in a single folder can slow down searches.

Pathname conventions are different in Windows and on the Macintosh:

- Windows 3.1 folder names must have eight or fewer characters and not use any of the characters that are unacceptable in Windows.
- Windows pathnames use back slashes (\) to separate folder names; Macintosh pathnames use colons (:). For example, the Macintosh pathname "Hard disk:Director:Projects:Movie 1:Sounds" is equivalent to the Windows pathname "C:\Director\Projects\Movie1\Sounds".
- Duplicate the folder structure between the movie and external resources on Windows systems. Folders do not have to be on drive C, but they should be on the same drive.

  Also, the folder structure does not need to duplicate the entire absolute pathname of the original system, but the structure must duplicate the relation between the files.

  {button See also,AL(`UsersGuide_231b_help')}

Exporting digital video

You can export all or part of a movie as a digital video and then import the video into a Director movie. The entire animation becomes a single cast member, which you can then move around the Stage as the movie plays. Exporting animation as a video captures not just the movement of the sprites on the Stage, but any tempos, palette effects, or transitions you've set. Any interactivity in the movie is lost when exported as digital video.

Export the Video for Windows (AVI) format using the Windows version of Director, or the QuickTime format on the Macintosh. Director cannot export to QuickTime for Windows.

Note: When you export to AVI, all sounds are lost. There is no way to preserve sounds in Director movies when exporting to AVI.

When Director exports animation as a video, it plays the animation, takes snapshots of the Stage moment by moment, and turns each snapshot into a frame in the video.

When Director exports animation as a video, it always uses the entire Stage. If you want to export a digital video that is smaller than your current Stage size, move the animation you want to export to the upper left corner of the Stage and reduce the size of the Stage to the size of the animation, as explained in the following procedure. If you don't need to change the Stage size, skip the procedure.

{button See also,AL(`UsersGuide_232_help')}

To prepare the animation for export:

1. **Select the animation in the Score, and then choose Modify > Sprite > Properties .**

   When you have more than one sprite selected, the size and location displayed in the Sprite Properties dialog box apply to a selection that surrounds all the frames and channels of the animation.

2. **Write down the width and height displayed in the dialog box.**
   The width and height tell you how big the Stage needs to be to accommodate the animation.

3. **Change the distance from both the top and left edges of the Stage to zero, and then click OK.**
   Director moves the animation to the upper left corner of the Stage.

4. **Choose Modify > Movie > Properties.**
   The size of the Stage needs to be as close as possible to the size of the area the animation occupies. On Macintosh systems, the width of the Stage must be divisible by 16.

   - If the area the animation occupies is smaller than 160 pixels wide by 120 pixels high, choose QuickTime 160 x 120 from the Stage Size pop-up.

   - If the area the animation occupies is larger than 160 pixels wide by 120 pixels high, round the width up to the nearest number divisible by 16 and type the number in the Width field.

Note: Turn off any screen savers before you export a movie as a digital video. Creating a digital video can take a long time, and if your screen saver comes on, Director will save it as part of the video.

{button See also,AL(`UsersGuide_233_help')}

To turn the animation into digital video:

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.**
3. **From the Format pop-up** in the Destination section at the bottom of the dialog box, choose Video for Windows (AVI) or QuickTime Movie.
4. **Click the Options button.**
   The Video for Windows or QuickTime Export Options dialog box appears.
5. **Select the options you want to use and then click OK.**
   The Export dialog box reappears when you click OK.
6. **Click Export.**
   A dialog box appears, prompting you to save the movie.
7. **Name the file** and then click Save.

Note: If you turn an animated sequence that includes a transition into a video, but the transition doesn't appear when you play the video, try increasing the Duration and Smoothness settings in the Frame Properties: Transition dialog box.

{button See also,AL(`UsersGuide_234_help')}

Shockwave Director basics

Shockwave for Director downloads and plays Director movies in Microsoft Internet Explorer and Netscape Navigator. Millions of web users have downloaded Shockwave from the Macromedia web site; others have received it with their operating system or web browser.

When Director creates a Shockwave movie, it compresses the movie's data to the smallest possible size for downloading. When a user views a web page that includes a Shockwave movie, the data from the movie begins downloading. Shockwave decompresses the data and plays the movie in the user's web browser. If the movie was created for streaming, it begins playing as soon as the data for the first frame has arrived.

For a demonstration and tutorial on Streaming Shockwave movies, see the **Streaming Shockwave** movie.

For a conceptual overview of developing content for Streaming Shockwave Movies, see Chapter 9, "Creating Shockwave Movies for the Web," in Using Director.

Note Director uses different Xtras to import media of each type. The Xtras that import the media types for all linked cast members must be present when a movie runs. The Shockwave players for Netscape Navigator and Microsoft Internet Explorer include the Xtras required to import GIF and JPEG graphics; AIFF (compressed and uncompressed), Shockwave Audio, and WAV (uncompressed only) sounds. Shockwave movies playing in web browsers can import these media types without downloading Xtras. To import other types of media, the required Xtras must first be downloaded and installed.

For projectors, the Xtra for each type of file being imported must be included with the projector. In most cases, Director handles this automatically. For more information, see the **Managing Xtras for distributed movies** help topic

{button See also,AL(`UsersGuide_246b_help')}

To set streaming options:

1. **Choose Modify > Movie > Playback to define streaming options.**

2. **Choose either Wait For All Media, Use Media as Available, or Show Placeholders.**
   - Wait For All Media makes the movie download completely before playing in the browser.
   - Use Media as Available turns on streaming playback. Unavailable cast members are ignored. Once they've downloaded, cast members appear in the movie and function as usual. This is the standard option to choose for distributing streaming movies.
   - Show Placeholders turns on streaming but makes the movie display placeholders for media that has not downloaded yet. This option can be very useful for testing.

3. **Set the Pre-Fetch box to the number of frames you want to download before the movie starts playing.**
   It's often a good idea to enter about 5 frames to make sure all the cast members for the first few frames of the movie are downloaded before the movie starts. This depends on how the movie is made.

With streaming playback, a movie begin playing as soon as a specified number of frames reaches the user's system. Without streaming playback, a movie downloads completely and then begin playing.

For a demonstration and tutorial on Streaming Shockwave movies, see the **Streaming Shockwave** movie.

Note If you want to test a movie streaming from a server before saving the movie as a Shockwave movie, first use File > Save and Compact to make sure the data in the movie is properly ordered and redundant data is removed.

{button See also,AL(`UsersGuide_251_help')}

Managing color palettes for browsers

When playing in a web browser, movies don't take over the color palette of a user's system the way stand-alone projectors do. Shockwave remaps the colors in Director movies to the most similar colors in the palette active in the user's system.

To get the best results in all types of browsers and systems, map all the images in your movie to the Netscape palette included with Director. Choose Xtras > Palettes to open the cast of included palettes.

Do not use any custom palettes in a movie or attempt to manipulate the user's palette in any way while the movie plays.

If you are not familiar with basic color palette concepts, see the **Understanding color palettes** help topic.

{button See also,AL(`UsersGuide_253_help')}

Total number of network operations

Most browsers support a limited number of concurrent network operations. A network operation is a distinct item that is being retrieved or sent over the internet. For example, downloading a movie counts as one operation, retrieving a linked media file counts as a second, and using getNetText would be a third. Try to keep your movies from performing more than four network operations at once.

{button See also,AL(`UsersGuide_254_help')}

Mapping fonts on the internet

If you use text cast members, font mapping between different platforms will not be an issue when distributing Shockwave movies on the internet. Shockwave treats text cast members exactly like bitmaps.

For field cast members, use standard system fonts that map well between platforms. If you use system fonts, you don't need to include a Fontmap.txt file. Shockwave maps the fonts between platforms using the default settings. Be sure to test the movie on all platforms to make sure this causes no problems.

If you use field cast members with non-system fonts, you must distribute the font itself and a customized Fontmap.txt file with the Shockwave movie. A typical Fontmap.txt is about 3K. For a description of font mapping, see the **Mapping fonts between platforms** help topic.

Linking to the Macromedia web site

To give potential viewers of your movie a quick way to download Shockwave, provide a link on your site to the Shockwave Central page in the Macromedia web site. The URL is:

http://www.macromedia.com/shockwave/index.html

You can also use the Shockwave logo graphic as part of your link.

Features disabled in browsers

The following features do not work for movies playing in a web browser.

- Movie-in-a-window
- Custom menus
- Looping playback specified with the Control Panel (use Lingo instead)

There are some Lingo commands that don't work in browsers. See "Shockwave, the internet, and Lingo" in Learning Lingo.

{button See also,AL(`UsersGuide_258_help')}

Downloading considerations

Multimedia delivered over the Internet is limited in size, primarily because the majority of users dial in at relatively slow speeds-14,400 or 28,800 bits per second. At 14,400 bps, a user receives only about 1400 data bytes per second. At 28,800 bps the rate increases to approximately 2800 bytes per second. At these speeds, it takes 30 seconds to one minute to download a 60K file.

Using streaming playback can help you avoid some of the problems caused by using large files, but it's important to be aware of downloading times.

If there is heavy traffic at the internet access point, or on the internet host, or if there's network congestion, the rate drops even lower-to as low as a few hundred bytes per second. For now, it is a good idea to assume your movies will download at a rate of about 1K per second.

The chart that follows 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).

Download times at common modem speeds

Content	14.4	28.8	64	1.5
Small graphics and animation, 30K	30 secs	10 secs	6 secs	1 sec
Small complete movie, 100-200K	100-200 secs.	50-100 secs	20-40 secs	1 sec
500K movie	500 secs	120-240 secs	90 secs	3 secs
1MB movie	NA	NA	180 secs	6 secs

{button See also,AL(`UsersGuide_260_help')}

Download times at common modem speeds

Content	14.4	28.8	64	1.5
Small graphics and animation, 30K	30 secs	10 secs	6 secs	1 sec
Small complete movie, 100-200K	100-200 secs.	50-100 secs	20-40 secs	1 sec
500K movie	500 secs	120-240 secs	90 secs	3 secs
1MB movie	NA	NA	180 secs	6 secs

OVERVIEW SCRIPTING HOW TO REFERENCE TROUBLE SHOW ME WEB LINKS

Shockwave browser compatibility

Shockwave works with Netscape Navigator as a plug-in, and with Microsoft Internet Explorer for Windows 95 and NT as an ActiveX control.

Shockwave can play Director movies in the following browsers:

Browser	Version	Platform
Netscape Navigator	2.0 or later	Windows and Macintosh
Microsoft Internet Explorer	3.0 or later	Windows and Macintosh

Shockwave also works with browsers that are compatible with the plug-in architecture of Netscape Navigator 3.0, including America Online.

The Director 6 Shockwave plug-in for Netscape and the ActiveX control for Internet Explorer can play Shockwave movies created with Director 4 and 5.

Note: When it first encounters an HTML page that references Shockwave, Internet Explorer for Windows asks the user for permission to download the Shockwave Active-X control. If the user approves, it downloads and installs the control.

{button See also,AL(`UsersGuide_261_help')}

Creating an HTML page to run a Shockwave movie

To make a movie play from a page on the web, you need to add the necessary tags to an HTML document.

To run a Director movie from an HTML document, use the EMBED or OBJECT tags. EMBED is the original tag defined by Netscape Navigator. OBJECT is the newer Microsoft Internet Explorer tag. All Shockwave compatible browsers support EMBED; newer browsers support the added functionality of OBJECT. To make sure a movie plays on as many compatible browsers as possible, it's best to use both tags.

The HTML statement that follows runs a movie from both Internet Explorer and Netscape Navigator. It is intended for use with Netscape Navigator versions 2.0 or later, and Internet Explorer version 3.0 or later. Because HTML is under constant revision, check the Director Developers Center web site for the latest information if you have any trouble.

For most Shockwave applications, you can enter the statement as shown and simply substitute your own values for the location and size of the movie. For more sophisticated applications, you may need to use additional parameters. For more information about OBJECT and EMBED parameters, see **Parameters for OBJECT and EMBED tags** help topic.

```
<OBJECT CLASSID="clsid:166B1BCA-3F9C-11CF-8075-444553540000"

CODEBASE="http://active.macromedia.com/director/cabs/
sw.cab#version=6,0,0,0"

WIDTH="512"

HEIGHT="480"

NAME="MovieName">

<PARAM NAME="SRC" VALUE="MYMOVIE.DCR">

<EMBED SRC="MYMOVIE.DCR" HEIGHT=480 WIDTH=512 NAME="MovieName">

</OBJECT>
```

For `WIDTH` and `HEIGHT`, enter the Stage size of your movie in pixels.

For `MYMOVIE.DCR`, enter the URL of your movie.

For `NAME`, enter a name to identify the movie for browser scripting (optional).

Tip You can play a movie in a browser without creating an HTML page, or even saving the movie as a Shockwave movie. Just drag the movie from the system desktop to a web browser window.

{button See also,AL(`UsersGuide_262_help')}

Parameters for OBJECT and EMBED tags

Both OBJECT and EMBED have a number of definable parameters. This table lists the syntax for using the parameter with both the OBJECT and EMBED tags.

Parameter	Definition	OBJECT syntax	EMBED syntax
CLASSID	The CLASSID parameter specifies the universal class identifier for the Shockwave ActiveX Control. Enter it exactly as shown in the next column.	CLASSID="clsid:166B1BCA-3F9C-11CF-8075-444553540000"	NA
CODEBASE	The CODEBASE parameter of the OBJECT tag specifies where the Director Shockwave ActiveX Control can be obtained if the user doesn't already have it installed in the browser, or if the user has a previous version installed. Enter it exactly as shown in the next column, except you may need to change the version number for sw.cab#version=.	CODEBASE="http://active.macromedia.com/director/cabs/sw.cab#version=6,0,0,0"	NA
WIDTH and HEIGHT	Use the WIDTH=width and HEIGHT=height parameters to specify the width and height of the image in pixels. The browser crops the image to the size you specify. In most cases, enter the exact Stage size of the movie.	WIDTH="356" HEIGHT="128"	WIDTH="356" HEIGHT="128"
ID	The ID parameter specifies a document-wide identifier. This identifier can be used for naming positions within documents to use as destinations of hypertext links. It can also be used by the browser or objects in the document to find and communicate with other objects embedded in the document. Replace document identifier with an identifier (SGML NAME token) that is unique to the HTML document.	ID="document identifier"	NA
NAME	The NAME parameter also provides a way for browsers that support FORMs to determine whether an object within a FORM block should participate in the "submit" process. The NAME parameter can be used by the browser or objects in the document to find and communicate with the	NAME="MovieName"	NAME="MovieName"

	movie. If NAME is specified, the browser should include the value of the NAME attribute and data obtained from the object along with the information derived from other form fields.		
SRC	Use the SRC parameter to specify the URL of the movie. The file's extension should be DCR. The file name is the name of your movie. The path is the path to the movie.	<PARAM NAME="SRC" VALUE="movie url">	SRC="movie url"
PLUGINSPAGE	If a Netscape Navigator user does not have Shockwave installed, the PLUGINSPAGE="url" parameter tells the browser to open a specified URL. Enter the parameter as shown at right to link to the right page in Macromedia's web site. If the PLUGINSPAGE parameter is not used, Netscape refers users to the page on its site that lists current plug-ins.	NA	PLUGINSPAGE="http://www.macromedia.com/shockwave"
PALETTE	PALETTE determines how the movie's palette affects the user's system when Shockwave plays the movie. PALETTE=background prevents the palette of the Director movie from loading and uses the system palette instead. This the correct setting for most movies. If this parameter is undefined, PALETTE=background is the default. When PALETTE=foreground, the palette of the movie takes over the user's system. This affects how other images and movies appear on the page and should not be used by those unfamiliar with these issues. PALETTE=background is the default when this attribute is not specified in the tag. Internet Explorer does not support PALETTE=foreground.	NA	PALETTE=background PALETTE=foreground
BGCOLOR	Defines the color of the movie rectangle before the movie itself appears. Use standard six character HTML RGB color descriptions to specify the color.	BGCOLOR=#FFFF00	BGCOLOR=#FFFF00

{button See also,AL(`UsersGuide_263_help')}

Parameters accessible from Lingo

There are several optional parameters for passing information from HTML to Lingo inside a movie. They are useful for a wide variety of functions in a movie. For an explanation of the Lingo that uses these parameters, see "External Parameter Access from Lingo" in Learning Lingo.

The parameters listed here work for both Navigator and Internet Explorer. Navigator allows you to name your own parameters, but to make sure a parameter is recognized on both browsers, use only these parameters.

Note: The maximum length of the strings passed to these parameters is specific to each browser. The parameter names have been prefixed with sw to avoid conflict with future HTML standards that may define additional tags such as URL or TEXT. Netscape Navigator 2.0 and later versions support user-defined parameter names on the EMBED tag. In future versions, the Shockwave ActiveX controller for Internet Explorer might support user-defined parameter names for the OBJECT tag. Currently, it supports only the parameters listed here.

Parameter	Suggested use	OBJECT syntax	EMBED syntax
swURL	Use this parameter to pass a URL to Lingo for use in a net Lingo command, such as gotoNetPage to include a dynamic linking movie, or an audio URL to alter the background music.	PARAM NAME="swURL" VALUE="parm value"	swURL="VALUE"
swText	Use HTML-specified variable text in the movie. Examples of use include creating text banners, showing a company name, phone number, name of a web page, and so on.	PARAM NAME="swText" VALUE="some text"	swText="VALUE"
swForeColor	Use colorCode to modify the foreground color of an object in the movie.	PARAM NAME="swForeColor" VALUE="colorCode"	swForeColor="VALUE"
swBackColor	Use colorCode to modify the background color of an object in the movie.	PARAM NAME="swBackColor" VALUE="colorCode"	swBackColor="VALUE"
swFrame	Use frameName to target a specific frame in a URL or to set a start frame of the movie.	PARAM NAME="swFrame" VALUE="frameName"	swFrame="VALUE"
swColor	Use objectColor to specify the color of a specific object in a movie.	PARAM NAME="swColor" VALUE="objectColor"	swColor="VALUE"
swName	Use userName to specify a name, such as a user name, to be displayed or used within the movie.	PARAM NAME="swName VALUE="userName"	swName="VALUE"
swPassword	Use userPassword to specify a password for some aspect of the movie.	PARAM NAME="swPassword" VALUE="userPassword"	swPassword="VALUE"
swBanner	Use theBanner to display specific text as a banner in the movie.	PARAM NAME="swBanner" VALUE="theBanner"	swBanner="VALUE"
swSound	Use theSound to name a	PARAM NAME="swSound"	swSound="VALUE"

	specific sound to be played. This allows the HTML author to determine the sound to be played. It can also be used as an indicator to turn on or off the sound.	VALUE="theSound"	
swVolume	Use theVolume to specify the sound volume for part or all of a movie. An integral value range between 0 and 10 is recommended, where 0 means that the sound is off and 10 means the sound is at maximum volume.	PARAM NAME="swVolume" VALUE="theVolume"	swVolume="VALUE"
swPreloadTime	Use theTime to specify with Shockwave Audio how many seconds of an audio file sound should be preloaded before playback starts.	PARAM NAME="swPreloadTime" VALUE="theTime"	swPreloadTime="VALUE"
swAudio	Use theAudio to specify the URL of an audio file to be played with a movie.	PARAM NAME="swAudio" VALUE="theAudio"	swAudio="VALUE"
swList	Put a comma-delimited list of items into theList that an author can parse and use in Lingo, for either key/value pairs or Boolean items, such as NoSound.	PARAM NAME="swList" VALUE="theList"	swList="VALUE"
sw1 through sw9	Additional parameters for authoring use. There are nine author-definable parameters, sw1 through sw9.	PARAM NAME="sw1" VALUE="Value"	sw1="VALUE"

Multiple movies in an HTML document

HTML documents can include more than one movie per page. The user can also scroll through the HTML page containing the movie while the movie is playing. Although technically there's no limit to how many movies you can incorporate into a web page, it's best to include no more than three. When a user leaves a page containing Shockwave movies, Shockwave frees the memory it was using to play the movies.

Browsers may have difficulty playing two movies with sound at once. Try to use sound in only one movie per HTML document, or program the movies so that the user activates the sound tracks by clicking the mouse.

{button See also,AL(`UsersGuide_265_help')}

Sound and multiple Shockwave movies

Browsers may have difficulty playing two movies with sound at once. Try to use sound in only one movie per HTML document, or program the movies so that the user activates the sound tracks by clicking the mouse.

{button See also,AL(`UsersGuide_266_help')}

Using Xtras

Xtras are software components that extend Director's functionality. They provide important capabilities such as importing filters and connecting to the internet. They also allow third-party developers to add specialized features to Director.

Xtras are also important for keeping movies small for distribution and downloading. There's no reason a movie should include code for importing media types it doesn't use, or networking code if it doesn't connect to the internet.

Although Xtras are written in C, you don't need to be a programmer to use them in a movie. Finished Xtras already exist; you just need to make them available to Director.

For more information about creating Xtras, see the Xtras Developer's Kit on the Director 6 CD.

{button See also,AL(`UsersGuide_268_help')}

Types of Xtras

- **Transition Xtras-**Transition Xtras supply transitions in addition to the predefined transitions available in the Frame Properties: Transition dialog box. After they are used in the Score's transition channel, Xtra transitions appear in the Cast window.

  An Xtra transition cast member can have its own custom properties, Properties dialog box, animated thumbnail, Cast window icon, and About box. When you use a transition Xtra, you must distribute it with your movie. In most cases this is handled automatically by turning on the Check Movie for Xtra option in the Projector Options dialog box.

- **Cast member Xtras-**Xtras can create a wide range of objects for use as cast members. These include databases, text managers, special types of graphics, and so on. Xtras that create cast members appear in the Insert menu. The Button Editor and QuickDraw 3D are examples of cast member Xtras. Cast member Xtras are also called asset Xtras.

  An Xtra cast member can have its own custom properties, Properties dialog box, media editor, animated thumbnail, Cast window icon, and About box. Open the dialog box that sets the Xtra's properties by opening the cast member's Properties dialog box and then clicking Options. Open the Xtra's media editor by double-clicking the cast member's thumbnail in the Cast window.

  When you use an Xtra cast member, you must distribute the Xtra that controls it with your movie. In most cases this is handled automatically by turning on the Check Movie for Xtras option in the Projector Options dialog box.

- **Scripting Xtras-**Scripting Xtras add Lingo elements to predefined Lingo. The Shockwave Audio Xtra, for example, provides special Lingo elements for controlling Shockwave Audio.

  Scripting Xtras must be distributed with any movie in which they are used. Since they do not appear in the Score, you must manually add them to the list of Xtras in the Movie Xtras dialog box. Choose Modify > Movie > Xtras and then click Add. Once they appear on the list in Movie Xtras, Scripting Xtras are automatically included with projectors as long as Check Movie for Xtras is turned on in Projector Options.

  For more information about using Xtras and Lingo, see the **Using Xtras** help topic.

- **Tool Xtras-**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.

  Bitmap filter Xtras (PhotoShop filters) are a good example of this type of Xtra. They create bitmapped cast members, but they don't do anything while the movie runs.

  Place any external cast in the Xtras folder to make it appear on the Xtras menu within Director. The Behavior Library is an example of this type of Xtra.

  Place any Director movie in the Xtras folder to make it appear on the Xtras menu and open as a movie in a window.

- **Importing Xtras-**Importing Xtras provide the code required for importing 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. In most cases this is handled automatically by turning on the Check Movie for Xtra option in the Projector Options dialog box. For more information, see the **Managing Xtras for distributed movies** help topic.

  The Shockwave players for Netscape Navigator and Microsoft Internet Explorer include the Xtras required to import GIF and JPEG graphics; AIFF (compressed and uncompressed), Shockwave Audio, and WAV (uncompressed only) sounds. Shockwave movies playing in web browsers can import these media types without downloading Xtras. To import other types of media, the required Xtras must first be downloaded and installed.

  {button See also,AL(`UsersGuide_269_help')}

Installing Xtras

When Director launches, it automatically recognizes Xtras in the Xtras folder in the same folder that contains the Director application.

To make an Xtra available, place it in this folder before you launch Director. (The Xtra can be in a folder within the Xtras folder up to five layers deep.) Director also automatically closes these Xtras when the application quits.

You can also open Scripting Xtras after Director is running by using the `openXlib` command. The Scripting Xtra can be in any folder if you open it this way. However, you must use the `closeXlib` command to close the Xtra after Director is finished with it.

Xtras can be packaged with projectors. For more information, see **Managing Xtras for distributed movies**.

If an Xtra that Director uses is missing, 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 for the missing Xtra.

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 if this occurs.

{button See also,AL(`UsersGuide_270_help')}

Checking which Xtras are available

Lingo can specify how many Scripting Xtras are available and the name of each.

The `the number of xtras` property indicates how many Scripting Xtras are available in the current movie.

The `the name of xtra` property determines the name of a specific Scripting Xtra. The `the name of xtra` property can be tested and set.

For example, the following repeat loop displays the name of each Xtra in the message window:

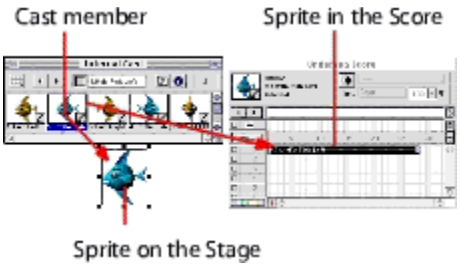
```
repeat with counter = 1 to (the number of xtras)
   put the name of xtra counter
```

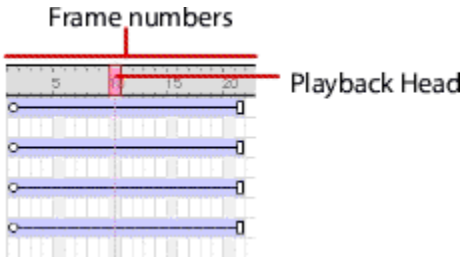
The showXLib command displays each Xtra file and its contents. For example, suppose that a Scripting Xtra Friends is in the folder c:\Xtra Reserve. If the Xtra file Friends contains the modules Fred and Joe, the showXlib command displays the following results:

```
c:\xtra reserve
   xtra Fred
   xtra Joe
```

Use `mMessageList` to display messages with information about the Xtra. For example, the statement `mMessageList(xtra "Fred")` displays information about the Xtra Fred.

{button See also,AL(`UsersGuide_271_help')}


Cast member
Sprite in the Score
Sprite on the Stage


Frame numbers
Playback Head
5
10
15
20

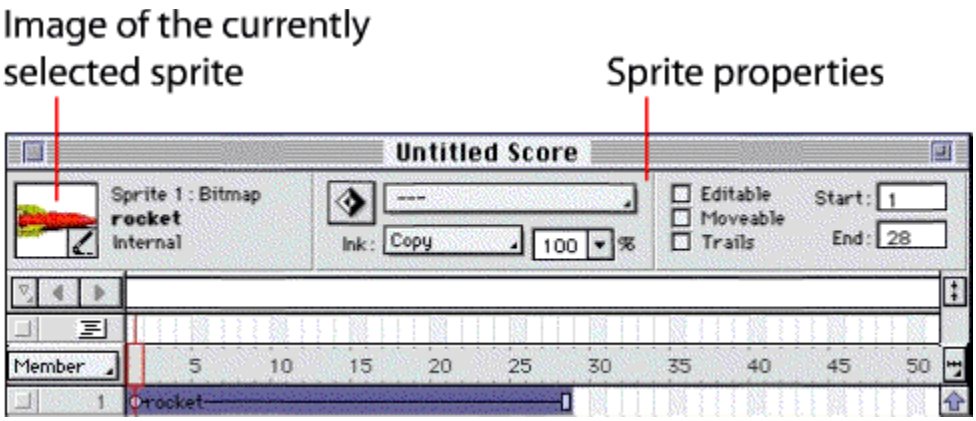
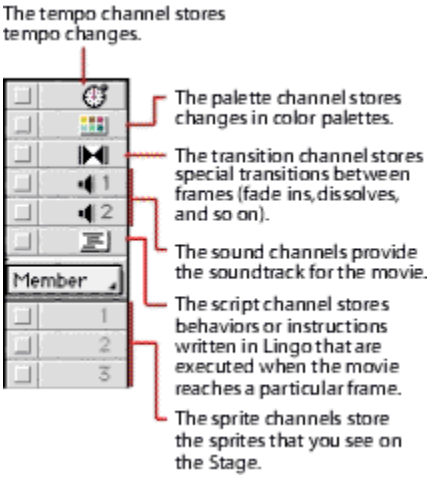
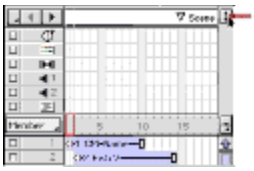
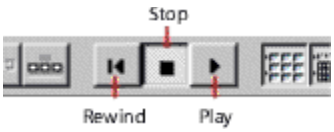


Image of the currently selected sprite
Sprite properties
Untitled Score
Sprite 1 : Bitmap
rocket
Internal

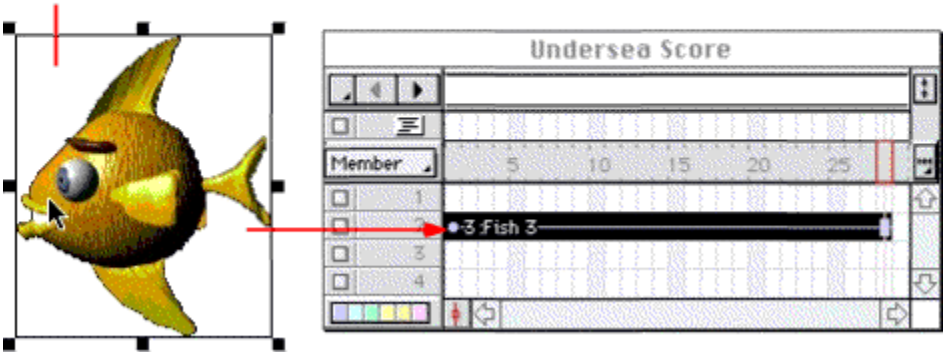
Ink :
Copy
100
%
Editable
Moveable
Trails
Start :
1
End :
28
Member
5
10
15
20
25
30
35
40
45
50
1
rocket

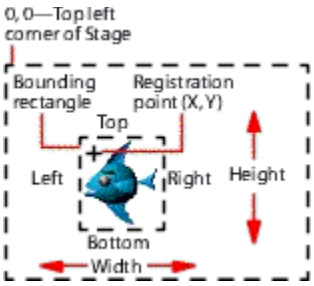

The tempo channel stores tempo changes.
The palette channel stores changes in color palettes.
The transition channel stores special transitions between frames (fade ins, dissolves, and so on).
The sound channels provide the soundtrack for the movie.
The script channel stores behaviors or instructions written in Lingo that are executed when the movie reaches a particular frame.
The sprite channels store the sprites that you see on the Stage.
1
2
Member
1
2
3


Score
Click here to show or hide the effects channels.

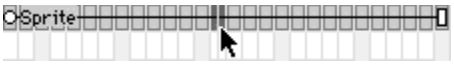

Stop
Rewind
Play

Clicking a sprite on the Stage selects the entire span of the sprite.

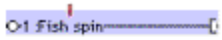



0, 0—Top left corner of Stage
Bounding rectangle
Registration point (X, Y)
Top
Left
Right
Height
Bottom
Width


Sprite


Sprite

Single sprite in the Score



Cast members in the sprite

Sprite animating

A single sprite can display several cast members


LOSE
NEXT
Library Cast
-- Marker
Go to
Named
-- Marker
Go to
Previous
-- Marker
Go to
Next
6


SynchMedia04.dir Score
Book
Build Menu
Member
...:Behavior Library Cast
Message All Sprites
15
Messaging
-- Message
Send to
Sprite
-- Message
Send to All
Sprites
-- Message
Movie in a
Window
17
18

Click here to expand the editing pane.