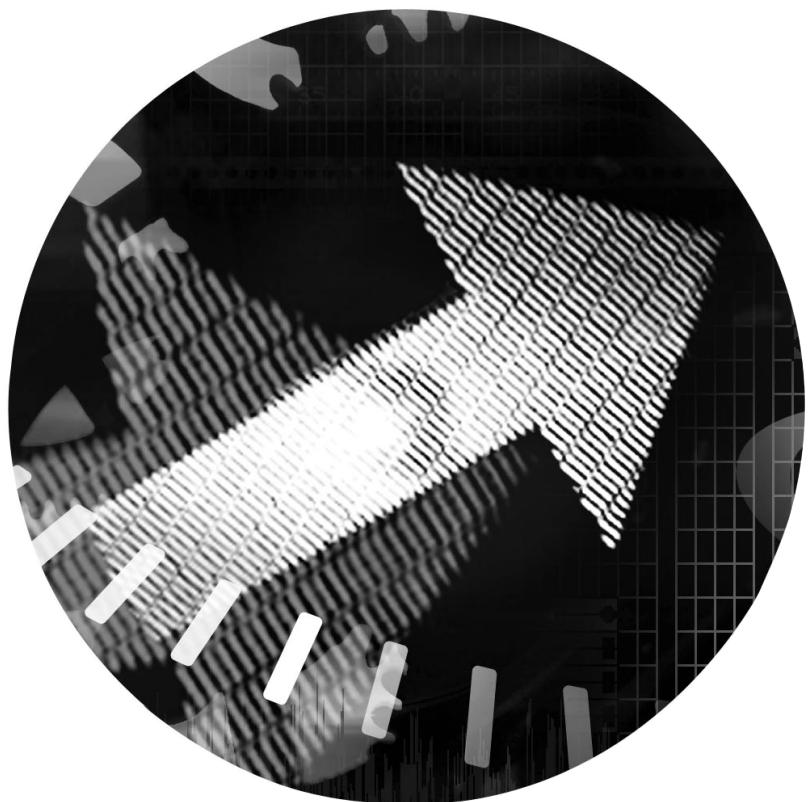


macromedia®
DIRECTOR® 8

Lingo™ Dictionary



Trademarks

Afterburner, AppletAce, Attain, Attain Enterprise Learning System, Attain Essentials, Attain Objects for Dreamweaver, Authorware, Authorware Attain, Authorware Interactive Studio, Authorware Star, Authorware Synergy, Backstage, Backstage Designer, Backstage Desktop Studio, Backstage Enterprise Studio, Backstage Internet Studio, Design in Motion, Director, Director Multimedia Studio, Doc Around the Clock, Dreamweaver, Dreamweaver Attain, Drumbeat, Drumbeat 2000, Extreme 3D, Fireworks, Flash, Fontographer, FreeHand, FreeHand Graphics Studio, Generator, Generator Developer's Studio, Generator Dynamic Graphics Server, Knowledge Objects, Knowledge Stream, Knowledge Track, Lingo, Live Effects, Macromedia, Macromedia M Logo & Design, Macromedia Flash, Macromedia Xres, Macromind, Macromind Action, MAGIC, Mediamaker, Object Authoring, Power Applets, Priority Access, Roundtrip HTML, Scriptlets, SoundEdit, ShockRave, Shockmachine, Shockwave, Shockwave Remote, Shockwave Internet Studio, Showcase, Tools to Power Your Ideas, Universal Media, Virtuoso, Web Design 101, Whirlwind and Xtra are trademarks of Macromedia, Inc. and may be registered in the United States or in other jurisdictions including internationally. Other product names, logos, designs, titles, words or phrases mentioned within this publication may be trademarks, servicemarks, or tradenames of Macromedia, Inc. or other entities and may be registered in certain jurisdictions including internationally.

This guide contains links to third-party Web sites that are not under the control of Macromedia, and Macromedia is not responsible for the content on any linked site. If you access a third-party Web site mentioned in this guide, then you do so at your own risk. Macromedia provides these links only as a convenience, and the inclusion of the link does not imply that Macromedia endorses or accepts any responsibility for the content on those third-party sites.

Apple Disclaimer

APPLE COMPUTER, INC. MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING THE ENCLOSED COMPUTER SOFTWARE PACKAGE, ITS MERCHANTABILITY OR ITS FITNESS FOR ANY PARTICULAR PURPOSE. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME STATES. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY PROVIDES YOU WITH SPECIFIC LEGAL RIGHTS. THERE MAY BE OTHER RIGHTS THAT YOU MAY HAVE WHICH VARY FROM STATE TO STATE.

Copyright © 2000 Macromedia, Inc. All rights reserved. This manual may not be copied, photocopied, reproduced, translated, or converted to any electronic or machine-readable form in whole or in part without prior written approval of Macromedia, Inc.

Part Number ZDR80M200

Acknowledgments

Writing: Jay Armstrong

Editing: Lisa Stanziano

Multimedia Design and Production: Larry Doyle, John Lehnus, and Noah Zilberberg

Print Production: Chris Basmajian

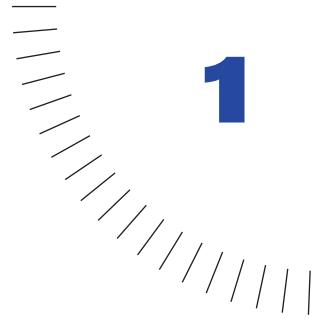
Sample Testing and Quality Assurance: Greg Barnett

Project Management: Joe Schmitz

Special Thanks: Peter DeCrescenzo, Buzz Kettles, and John "JT" Thompson

First Edition: February 2000

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



CHAPTER 1

Lingo by Feature

Lingo by feature: Overview

This appendix lists various Director features and the corresponding Lingo elements that you can use to implement those features.

Lists of Lingo for the Multiuser server and XML parsing are on Macromedia's web site. The entries for each of these features provides a link that takes you to the pages for these topics in the Director Developers Center at www.macromedia.com.

Animated GIFs

These terms are useful for working with animated GIFs:

directToStage	pause (movie playback)
frameRate	playBackMode
linked	resume sprite
moviePath	rewind sprite

Animation

These terms are useful for creating animation with Lingo:

blend	locV
ink	member (sprite property)
loc	regPoint
loch	tweened

Behaviors

These terms are useful for authoring behaviors and using behaviors while the movie plays.

Authoring behaviors

Use these terms to set up behaviors and the behavior's Parameters dialog box:

ancestor	on getBehaviorDescription
on runPropertyDialog	on getPropertyDescriptionList
on getBehaviorTooltip	property
on isOKToAttach	

Sending messages to behaviors

Use these commands to send messages to behaviors attached to sprites:

call	sendSprite
callAncestor	sendAllSprites

Identifying behaviors

Use these terms to identify the behaviors attached to sprites:

currentSpriteNum	scriptInstanceList
me	spriteNum

Bitmaps

These terms are useful for working with bitmaps.

Bitmap properties

Use these terms to check and set bitmap properties:

alphaThreshold	foreColor
backColor	palette
blend	picture (cast member property)
depth	pictureP()
dither	rect (member)
tripWhiteSpace	imageCompression
imageQuality	movielImageCompression
movielImageQuality	

Alpha channel

Use these terms to control alpha channel effects:

alphaThreshold	dither
depth	useAlpha
createMask()	createMatte()
extractAlpha()	setAlpha()

Image objects

Use these terms to create and control image objects:

copyPixels()	fill()
crop()	image
draw	image()
duplicate() (image function)	rect (image)
getPixel()	setPixel()

Cast members

These terms are useful for working with cast members.

Creating cast members

Use `importFileInto` and `new()` to create cast members.

Authoring

Use `duplicate member`, `erase member`, and `pasteClipBoardInto` to work with cast members during authoring.

Graphic cast members

Use these terms to check and set the images assigned to graphic cast members:

center	palette
crop (cast member property)	picture (cast member property)
depth	pictureP()
media	regPoint

General cast member properties

Use these terms to check and set cast member properties:

fileName (cast member property)	number (cast member property)
media	preLoadMode
modified	type (cast member property)
name (cast member property)	URL

Graphic cast member dimensions

Use `height`, `rect (member)`, and `width` to check and set dimensions for graphic cast members.

Casts

These terms are useful for working with casts.

Loading casts

Use preLoadMode to check and set when Director preloads a cast.

Cast properties

Use these terms to specify cast properties:

castLib	number (cast property)
fileName (cast property)	number (system property)
name (cast property)	

Cast management

Use these terms to manage casts:

activeCastLib	number of members
duplicate member	pasteClipBoardInto
erase member	save castLib
findEmpty()	selection (cast property)
move member	

Computer and operating system

Use these terms to check and control the computer:

beep	freeBlock()
beepOn	freeBytes()
cpuHogTicks	maxInteger
emulateMultiButtonMouse	multiSound
floatPrecision	romanLingo

Operating system control

Use restart and shutDown to control the operating system.

Data types

These terms are useful for specifying data types:

# (symbol)	string()
float()	stringP()
floatP()	symbol()
integer()	symbolP()
integerP()	VOID
objectP()	voidP()

Digital video

These terms are useful for working with AVI and QuickTime digital video:

controller	trackNextSampleTime
digitalVideoTimeScale	trackPreviousKeyTime
digitalVideoType	trackPreviousSampleTime
directToStage	trackStartTime (sprite property)
duration	trackStartTime (cast member property)

frameRate	trackStopTime (sprite property)
loop (cast member property)	trackStopTime (cast member property)
movieRate	trackText
movieTime	trackType (cast member property)
pausedAtStart	trackType (sprite property)
quickTimeVersion()	trackCount (cast member property)
timeScale	trackCount (sprite property)
trackEnabled	video
trackNextKeyTime	videoForWindowsPresent

QuickTime

Use these terms to work with QuickTime:

enableHotSpot	nodeType
fieldOfView	nudge
getHotSpotRect()	pan (QTVR property)
hotSpotExitCallback	ptToHotSpotID()
hotSpotEnterCallback	quickTimeVersion()
invertMask	rotation
isVRMovie	scale
loopBounds	swing()
mask	staticQuality
motionQuality	tilt
mouseLevel	translation
node	triggerCallback
nodeEnterCallback	warpMode
nodeExitCallback	

Events

Use these event handlers for Lingo that runs when a specific event occurs:

on activateWindow	on moveWindow
close window	on mouseWithin
on cuePassed	open window
on deactivateWindow	on prepareFrame
on enterFrame	on prepareMovie
on EvalScript	on resizeWindow
on exitFrame	on mouseUpOutside
on idle	on rightMouseDown (event handler)
on keyDown	on rightMouseUp (event handler)
on keyUp	on startMovie
on mouseDown (event handler)	on stepFrame
on mouseEnter	on streamStatus
on mouseLeave	on timeOut
on mouseUp (event handler)	on zoomWindow
on stopMovie	on beginSprite
on endSprite	on hyperLinkClicked

Use the `pass` and `stopEvent` commands to override the way that Director passes messages along the message hierarchy.

External files

These terms are useful for working with external files.

Path and file names

Use these terms to check and set path and file names:

@ (pathname)	getNthFileNameInFolder()
applicationPath	moviePath
fileName (cast property)	searchCurrentFolder
fileName (cast member property)	URL

Obtaining external media

Use these terms to obtain external media:

downloadNetThing	preloadNetThing()
importFileInto	

Managing external files

Use these terms to manage external files:

closeXlib	showXlib
open	sound playFile
openXlib	

Flash

These terms are useful for working with Flash cast members:

actionsEnabled	originMode
broadcastProps	originPoint
bufferSize	originV
buttonsEnabled	pathName (movie property)
bytesStreamed	pausedAtStart
centerRegPoint	percentStreamed
clearError	play
clickMode	playBackMode
defaultRect	playing
defaultRectMode	posterFrame
directToStage	quality
eventPassMode	rewind sprite
findLabel()	rotation
fixedRate	scale
flashRect	scaleMode
flashToStage()	showProps()
frame() (function)	sound
frame (sprite property)	stageToFlash()
frameCount	state
frameRate	static
frameReady()	stop (Flash)
getError()	stream
getFrameLabel() (function)	streamMode
goToFrame	streamSize
hitTest()	URL
hold	viewH

imageEnabled	viewPoint
linked	viewScale
loop (keyword)	viewV
mouseOverButton	volume (cast member property)
obeyScoreRotation	getVariable()
originH	setVariable()
getFlashProperty()	setFlashProperty()
soundMixMedia	

Frames

These Lingo terms let you work with frames.

Frame events

Use the on enterFrame, on exitFrame, and on prepareFrame event handlers to contain Lingo that runs at specific events during the course of a frame.

Frame properties

Use these Lingo terms to check and set frame properties:

frameLabel	frameTempo
framePalette	frameTransition
frameScript	label()
frameSound1	labelList
frameSound2	marker()
markerList	

Interface elements

These terms are useful for working with interface elements.

Menus

Use these terms to create menus:

enabled	name (menu item property)
installMenu	number (menu items)
menu	number (menus)
name (menu property)	script

Buttons and check boxes

Use these terms to specify buttons and check boxes:

alert	checkButtonType
buttonStyle	checkMark
buttonType	hilite (cast member property)
checkBoxAccess	

Keys

These terms are useful for Lingo related to using the keyboard.

Identifying keys

Use these terms to identify keys:

charToNum()	keyPressed()
commandDown	mouseChar
controlDown	numToChar()
key()	optionDown
keyCode()	shiftDown

Keyboard interaction

Use keyPressed(), lastEvent(), and lastKey to detect what the user types at the keyboard.

Keyboard events

Use these terms to set up handlers that respond to pressing keys:

on keyDown	keyDownScript
on keyUp	keyUpScript
flushInputEvents	

Lingo

These terms are important language elements that you use to construct scripts.

Booleans

Use these terms to test whether a condition exists:

- FALSE (0 is the numerical equivalent of FALSE.)
- TRUE (1 is the numerical equivalent of TRUE.)
- not
- or

Script control

Use these terms to control how a script executes:

abort	pass
do	result
exit	scriptsEnabled
halt	scriptText
nothing	stopEvent

Code structures

Use if to create if..then statements.

Use case, end case, and otherwise in case statements.

Use these terms for repeat loops:

end repeat	repeat with
exit repeat	repeat with...down to
next repeat	repeat with...in list
repeat while	

Syntax elements

Use these terms as part of Lingo's syntax:

# (symbol)	member (keyword)
" (string)	of
¬ (continuation)	or
-- (comment)	property
() (parentheses)	sprite
castLib	the
end	window
global	

Lists

These terms are useful for working with lists.

Creating lists

Use [] (list), duplicate() (list function), or list() to create a list.

Adding list items

Use these terms to add items to a list:

[] (bracket access)	addProp
add	append
addAt	

Deleting list items

Use these terms to delete items from a list:

deleteAll	deleteOne
deleteAt	deleteProp

Retrieving values from a list

Use these terms to retrieve values from a list:

[] (bracket access)	getOne()
deleteProp	getPos()
deleteProp	getProp()
getLast()	getPropAt()

Getting information about lists

Use these terms to get information about lists:

count()	max()
findPos	min
findPosNear	param()
ilk()	paramCount()
listP()	

Setting values in a list

Use these terms to set values in a list:

[] (bracket access)	setAt
setAProp	setProp

Media synchronization

Use these terms to synchronize animation and sound:

cuePointNames	on cuePassed
cuePointTimes	isPastCuePoint()
mostRecentCuePoint	

Memory management

These terms are useful for determining memory requirements and controlling when the movie loads and unloads cast members.

Use the on idle event handler for Lingo that runs when the movie is idle.

Idle loading

Use these terms to control idle loading:

cancelIdleLoad	idleLoadPeriod
finishIdleLoad	idleLoadTag
idleHandlerPeriod	idleReadChunkSize
idleLoadDone()	netThrottleTicks

Preloading and querying media

Use these terms to load media into memory and check whether media is available:

frameReady()	preloadNetThing()
loaded	preLoadMember
mediaReady	preLoadMovie
preLoad (command)	preLoadRAM
preLoad (cast member property)	purgePriority
preLoadBuffer member	unLoad
preLoadEventAbort	unLoadMember
preLoadMode	unloadMovie

Available memory

Use these terms to check how much memory is available:

freeBlock()	movieFileFreeSize
freeBytes()	movieFileSize
memorySize	

Memory requirements

Use `ramNeeded()` and `size` to determine how much memory required for a cast member or a range of frames.

Message window

Use these terms to work in the Message window:

put	traceLoad
showXlib	traceLogFile
trace	appMinimize

Monitor

Use `colorDepth`, `deskTopRectList`, and `switchColorDepth` to check and control the monitor.

Mouse interaction

These terms are useful for Lingo related to using the mouse.

Mouse clicks

Use these terms to detect what the user does with the mouse:

clickOn	mouseLine
doubleClick	mouseLoc
emulateMultiButtonMouse	mouseMember
lastClick()	mouseOverButton
lastEvent()	on mouseUp (event handler)
lastRoll	mouseV
mouseChar	mouseWord
on mouseDown (event handler)	on rightMouseDown (event handler)
mouseH	on rightMouseUp (event handler)
mouseItem	rollOver()
mouseLevel	stillDown

Mouse events

Use these terms to set up handlers that respond to mouse events:

mouseDownScript	on mouseUp (event handler)
mouseUpScript	on mouseUpOutside
on mouseDown (event handler)	on mouseWithin
on mouseEnter	on rightMouseDown (event handler)
on mouseLeave	on rightMouseUp (event handler)

Cursor control

Use cursor (command), cursor (sprite property), and cursorSize to control the cursor.

Movie in a window

These terms are useful for working with movies in a window.

Movie in a window events

Use these event handlers to contain Lingo that you want to run in response to events in a movie in a window:

on activateWindow	on openWindow
on closeWindow	on resizeWindow
on moveWindow	zoomWindow

Opening and closing movies in a window

Use these terms for opening and closing windows:

close window	open window
forget window	windowList

Window appearance

Use these terms to check and set the appearance of a movie's window:

drawRect	sourceRect
fileName (window property)	tell
frontWindow	title
modal	titleVisible
moveToBack	visible (window property)
moveToFront	windowPresent()
name (window property)	windowType
rect (window)	appMinimize

Communication between movies

Use the tell command to send messages between movies.

Movies

These terms are useful for managing movies.

Stopping movies

Use these terms to stop or quit the movie or projector:

exitLock	quit
halt	restart
pauseState	shutDown

Movie information

Use these terms to obtain information about the movie and the movie's environment:

environment	moviePath
lastFrame	number (system property)
movie	runMode
movieFileFreeSize	safePlayer
movieFileSize	version
movieName	movieFileVersion

Source control

Use these terms to manage Director projects with more than one person working on them:

comments	creationdate
modifiedBy	modifiedDate
linkAs()	seconds

Saving movies

Use `saveMovie` and `updateMovieEnabled` to save changes to a movie.

Error checking

Use the alertHook event to post alerts that describe errors in a projector.

Movie events

Use the on prepareMovie, on startMovie, and on stopMovie event handlers for Lingo that responds to movie events.

Multiuser server

For information about individual entries, see “Using the Multiuser Server Xtra,” in the Director Developers Center.

Navigation

Use these terms to jump to different locations:

delay	goToFrame
go	gotoNetMovie
go loop	gotoNetPage
go next	play
go previous	play done

Network Lingo

These terms are useful for working with the network.

Downloading and streaming media

Use these terms to obtain or stream media from the network:

downloadNetThing (For projectors and authoring only)	gotoNetPage
getNetText()	postNetText
gotoNetMovie	preloadNetThing()

Checking availability

Use frameReady() and mediaReady to check whether specific media is completely downloaded.

Using network operations

Use these terms to check the progress of a network operation or get information regarding network media:

getStreamStatus	netLastModDate
getLatestNetID	netMIME
netAbort	netTextResult
netDone	on streamStatus
netError	proxyServer
netPresent	tellStreamStatus

Working with the local computer

Use these terms to work with the user's computer:

browserName	clearCache (For projectors and authoring only)
cacheDocVerify (For projectors and authoring only)	getPref
cacheSize (For projectors and authoring only)	setPref

Browsers

Use onEvalScript, externalEvent, and netStatus to interact with browsers. For additional information about browser scripting using languages such as JavaScript, see “Additional browser scripting information,” in the Director Developers Center.

Accessing EMBED and OBJECT tag parameters

Use externalParamCount(), externalParamName(), and externalParamValue() to access EMBED and OBJECT parameter tags:

Operators

These terms are operators available in Lingo.

Math operators

Use these terms for math statements:

* (multiplication)	<> (not equal)
/ (division)	> (greater than)
+ (addition)	>= (greater than or equal to)
- (minus)	< (less than)
= (equals)	<= (less than or equal to)

Comparison operators

Use and, not, and or to compare expressions.

Palettes and color

Use these terms to check and set palettes for movies and for cast members:

color()	paletteMapping
depth	puppetPalette
palette	RGB

Parent scripts

Use these terms to work with parent scripts and child objects:

actorList	property
ancestor	on stepFrame
new()	handler
handlers()	rawNew()

Points and rects

These terms are useful for checking and setting points and rectangles.

inflate	quad
inside()	rect()
intersect()	rect (sprite)
map()	sourceRect
offset() (rectangle function)	union()
point()	

For Lingo that controls a sprite's bounding rectangle, see [Sprite dimensions](#) in this appendix.

Projectors

These terms are useful for working with projectors:

alertHook	platform
environment	runMode
editShortCutsEnabled	

Puppets

Use this Lingo to control the puppet of sprites and effects channels:

puppet	puppetTempo
puppetPalette	puppetTransition
puppetSound	updateStage
puppetSprite	

Score

The following terms let you work with the Score.

Score properties

Use lastFrame, score, and scoreSelection to work with the movie's Score.

Score generation

Use these terms to generate score from Lingo:

beginRecording	scoreSelection
clearFrame	scriptNum
deleteFrame	scriptType
duplicateFrame	tweened
endRecording	updateFrame
insertFrame	updateLock
scoreColor	

Shapes

Use these Lingo terms to work with shapes:

filled	pattern
lineDirection	shapeType
lineSize	

Shockwave audio

Use these terms to check, stream, and play Shockwave audio sounds:

bitRate	play member
bitsPerSample	preLoadBuffer member
copyrightInfo	preLoadTime
duration	sampleRate
getError()	soundChannel
getErrorString()	state
numChannels	stop member
pause (movie playback)	streamName
percentPlayed	URL
percentStreamed	volume (cast member property)

Sound

These terms are useful for playing sounds.

Sound information

Use these terms for information about a sound:

channelCount	soundEnabled
sound	volume (sprite property)
soundBusy()	isBusy()
sampleCount	status

Playing sound

Use these terms to control how sound plays:

puppetSound	sound fadeOut
sound close	sound playFile
sound fadeIn	sound stop
breakLoop	elapsedTime
endTime	fadeIn
fadeOut	fadeTo
getPlayList	setPlayList()
loopCount	loopEndTime
loopStartTime	loopsRemaining
member (sound property)	pan
pause	playNext()
queue()	rewind()
stop()	

Sprites

These Lingo terms are for sprites.

Sprite events

Use the on beginSprite and on endSprite event handlers to contain Lingo that you want to run when a sprite begins or ends.

Assigning cast members to sprites

Use castLibNum, member (sprite property), or memberNum to specify a sprite's cast member.

Sprite rotation

Use the rotation sprite property to rotate sprites.

Dragging sprites

Use these terms to set how the user can drag sprites:

constrainH()	moveableSprite
constrainV()	sprite...intersects
constraint	sprite...within

Sprites and Lingo

Use these terms to manage how Lingo controls sprites:

puppetSprite	spriteNum
puppet	sendSprite
scriptNum	sendAllSprites
scriptInstanceList	

Drawing sprites on the Stage

Use these terms to control how Director draws a sprite on the Stage:

blend	skew
flipH	trails
flipV	tweened
ink	updateStage
quad	visible (sprite property)
rotation	

Sprite dimensions

Use these terms to check and set the size of a sprite's bounding rectangle:

bottom	right
height	top
left	width
quad	zoomBox

You can also manipulate a sprite's bounding rectangle with Lingo for rectangles. See Points and rects.

Sprite locations

Use the loc, locH, and locV sprite properties to check and set sprite locations.

Sprite color

Use these terms to check and set a sprite's color:

backColor	color (sprite property)
bgColor	foreColor

Stage

These terms are useful for controlling the Stage and determining its size and location:

centerStage	stageColor
fixStageSize	stageLeft
picture (window property)	stageRight
stage	stageTop
stageBottom	updateStage

Tempo

Use the `puppetTempo` command to control a movie's tempo.

Text

These terms are useful for working with text, strings, and fields.

Manipulating strings

Use these terms to manipulate strings:

& (concatenator)	put...before
&& (concatenator)	put...into
delete	string()
hilite (cast member property)	stringP()
put...after	text

Chunk expressions

Use these terms to identify chunks of text:

char...of	number (words)
chars()	offset() (string function)
contains	paragraph
EMPTY	ref
item...of	selection (text cast member property)
itemDelimiter	selectedText
last()	selEnd (fields only)
length()	selStart (fields only)
line...of	string()
number (characters)	stringP()
number of items in	value()
number of lines in	word...of
number of paragraphs	

Editable text

Use the `editable` property to specify whether text is editable.

Shocked fonts

Use these terms to include Shocked fonts with downloaded text:

recordFont	bitMapSizes
originalFont	characterSet

Character formatting

Use these terms to format text:

backColor	fontf
bgColor	fontSize
charSpacing	fontStyle
color()	foreColor
dropShadow	

Paragraph formatting

Use these terms to format paragraphs:

alignment	rightIndent
bottomSpacing	tabCount
firstIndent	tabs
fixedLineSpace	topSpacing
leftIndent	wordWrap
margin	

Text cast member properties

Use these terms to work with the entire text content of a text cast member:

antiAlias	kerning
antiAliasThreshold	kerningThreshold
autoTab	picture (cast member property)
HTML	RTF

Lingo that applies to chunk expressions is also available to the text within a text cast member.

Mouse pointer position in text

Use these terms to detect where the mouse pointer is within text:

pointInHyperlink()	pointToParagraph()
pointToChar()	pointToWord()
pointToItem()	

Text boxes for field cast members

Use these terms to set up the box for a field cast member:

border	lineHeight() (function)
boxType	lineHeight (cast member property)
lineCount	pageHeight

Scrolling text

Use these terms to work with scrolling text:

linePosToLocV()	scrollByLine
locToCharPos()	scrollByPage
locVToLinePos()	scrollTop

Constants

Use these terms to work with constants:

BACKSPACE	RETURN (constant)
EMPTY	VOID
ENTER	

Time

These terms are useful for working with time.

Current date and time

Use these terms to determine the current date and time:

abbr, abbrev, abbreviated	short
date() (system clock)	systemDate
long	

Measuring a length of time

Use these terms to measure time in a movie:

framesToHMS()	ticks
HMSToFrames()	time()
milliseconds	timer
startTimer	

Timeouts

Use these terms to handle timeouts:

timeoutKeyDown	timeoutMouse
timeoutLapsed	timeoutPlay
timeoutLength	timeoutScript
name (timeout property)	period
persistent	target
time	timeout
timeoutHandler	timeoutList

Transitions

Use these terms to work with transitions:

changeArea	puppetTransition
chunkSize	transitionType
duration	

Variables

These terms are useful for creating and changing variables:

Creating variables

Use these terms to create variables:

= (equals)	property
global	

Testing and changing variables

Use these terms to check and change the values assigned to variables:

= (equals)	put
clearGlobals	set...to, set...=
globals	showGlobals
ilk()	showLocals

Vector Shapes

Use these Lingo terms to work with vector shapes:

addVertex	gradientType
antiAlias	imageEnabled
backgroundColor	moveVertex()
broadcastProps	moveVertexHandle()
centerRegPoint	originH
closed	originMode
defaultRect	originPoint
defaultRectMode	originV
deleteVertex()	rotation
directToStage	scale
endColor	scaleMode
fillColor	showProps()
fillCycles	skew
fillDirection	static
fillMode	strokeColor
fillOffset	strokeWidth
fillScale	vertexList
flashRect	viewPoint
flipH	viewScale
flipV	viewV
curve	newCurve()
regPointVertex	

XML parsing

The following Lingo is useful for XML parsing within Director. For information about individual entries, see “XML Parsing,” in the Director Developers Center.

getAttributeName	getAttributeValue
getCharacterData	getElementsName
getNumberOfAttributes	getNumberOfChildren
isCurrentAnElement	makeList
parseString	visitChild
visitParent	

Xtras

Use these terms to work with Xtras:

movieXtraList	xtra
name (system property)	xtraList
number of xtras	xtras

Miscellaneous

Use random() and randomSeed to generate random numbers.



CHAPTER 2

Lingo Dictionary

Overview

This dictionary describes the syntax and use of Lingo elements in Director 8. Nonalphabetical operators are presented first, followed by all other operators in alphabetical order. For information about Lingo used for the Multiuser server Xtra, see “Using the Multiuser server Xtra” in the Director Support Center. For information about Lingo used for XML parsing, see “XML Parsing” in the Director Support Center.

The entries in this dictionary are the same as those in Director Help. To use examples in a script, copy the example text from Director Help and paste it in the Script window.

(symbol)

Syntax `#symbolName`

Description Symbol operator; defines a symbol, a self-contained unit that can be used to represent a condition or flag. The value *symbolName* begins with an alphabetical character and may be followed by any number of alphabetical or numerical characters.

A symbol can:

- Assign a value to a variable
- Compare strings, integers, rectangles, and points
- Pass a parameter to a handler or method
- Return a value from a handler or method

A symbol takes up less space than a string and can be manipulated, but unlike a string it does not consist of individual characters. You can convert a symbol to a string for display purposes by using the `string` function.

The following are some important points about symbol syntax:

- Symbols are not case sensitive.
- Symbols can't start with a number.
- Spaces may not be used, but you can use underscore characters to simulate them.
- Symbols use the 128 ASCII characters, and letters with diacritical or accent marks are treated as their base letter.
- Periods may not be used in symbols.

All symbols, global variables, and names of parameters passed to global variables are stored in a common lookup table.

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

```
state = #Playing
```

See also `ilk()`, `string()`, `symbol()`, `symbolP()`

. (dot operator)

Syntax *objectReference.objectProperty*
textExpression.objectProperty
object.commandOrFunction()

Description Operator; used to test or set properties of objects, or to issue a command or execute a function of the object. The object may be a cast member, a sprite, a property list, a child object of a parent script, or a behavior.

Example This statement displays the current member contained by the sprite in channel 10:

```
put sprite(10).member
```

Example To use the alternate syntax and call a function, you can use the form:

```
myColorObject = color(#rgb, 124, 22, 233)  
put myColorObject.ilk()  
-- #color
```

- (minus)

Syntax (Negation): *-expression*

Description Math operator; reverses the sign of the value of *expression*.

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

Syntax (Subtraction): *expression1 - expression2*

Description Math operator; performs an arithmetic subtraction on two numerical expressions, subtracting *expression2* from *expression1*. When both expressions are integers, the difference is an integer. When either or both expressions are floating-point numbers, the difference is a floating-point number.

This is an arithmetic operator with a precedence level of 3.

Example (Negation): This statement reverses the sign of the expression 2 + 3:

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

The result is -5.

Example (Subtraction): This statement subtracts the integer 2 from the integer 5 and displays the result in the Message window:

```
put 5 - 2
```

The result is 3, which is an integer.

Example (Subtraction): This statement subtracts the floating-point number 1.5 from the floating-point number 3.25 and displays the result in the Message window:

```
put 3.25 - 1.5
```

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

-- (comment)

Syntax

-- *comment*

Description

Comment delimiter; indicates the beginning of a script comment. On any line, anything that appears between the comment delimiter (double hyphen) and the end-of-line return character is interpreted as a comment rather than a Lingo statement.

The Director player for Java accepts Lingo that uses this delimiter, but comments do not appear in the final Java code.

Example

This handler uses a double hyphen to make the second, fourth, and sixth lines comments:

```
on resetColors
    -- This handler resets the sprite's colors.
    sprite(1).forecolor = 35
    -- bright red
    sprite(1).backcolor = 36
    -- light blue
end
```

& (concatenator)

Syntax

expression1 & *expression2*

Description

String operator; performs a string concatenation of two expressions. If either *expression1* or *expression2* is a number, it is first converted to a string. The resulting expression is a string.

This is a string operator with a precedence level of 2.

Be aware that Lingo allows you to use some commands and functions that take only one argument without parentheses surrounding the argument. When an argument phrase includes an operator, Lingo interprets only the first argument as part of the function, which may confuse Lingo.

For example, the open window command allows one argument that specifies which window to open. If you use the & operator to define a pathname and file name, Director interprets only the string before the & operator as the file name. For example, Lingo interprets the statement `open window the applicationPath & "theMovie"` as `(open window the applicationPath) & ("theMovie")`. Avoid this problem by placing parentheses around the entire phrase that includes an operator, as follows:

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

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

Example This statement concatenates the strings “abra” and “cadabra” and displays the resulting string in the Message window:

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

The result is the string “abracadabra”.

Example This statement concatenates the strings “\$” and the content of the price variable and then assigns the concatenated string to the Price field cast member:

```
member("Price").text = "$" & price
```

&& (concatenator)

Syntax *expression1 && expression2*

Description String operator; concatenates two expressions, inserting a space character between the original string expressions. If either *expression1* or *expression2* is a number, it is first converted to a string. The resulting expression is a string.

This is a string operator with a precedence level of 2.

Example This statement concatenates the strings “abra” and “cadabra” and inserts a space between the two:

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

The result is the string “abra cadabra”.

Example This statement concatenates the strings “Today is” and today’s date in the long format and inserts a space between the two:

```
put "Today is" && the long date
```

If today’s date is Tuesday, March 15, 1999, the result is the string “Today is Tuesday, March 15, 1999”.

() (parentheses)

Syntax *(expression)*

Description Grouping operator; performs a grouping operation on an expression to control the order of execution of the operators in an expression. This operator overrides the automatic precedence order so that the expression within the parentheses is evaluated first. When parentheses are nested, the contents of the inner parentheses are evaluated before the contents of the outer ones.

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

Be aware that Lingo allows you to use some commands and functions that take only one argument without parentheses surrounding the argument. When an argument phrase includes an operator, Lingo interprets only the first argument as part of the function, which may confuse Lingo.

For example, the open window command allows one argument that specifies which window to open. If you use the & operator to define a pathname and file name, Director interprets only the string before the & operator as the file name. For example, Lingo interprets the statement `open window the applicationPath & "theMovie"` as `(open window the applicationPath) & ("theMovie")`. Avoid this problem by placing parentheses around the entire phrase that includes an operator, as follows:

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

Example These statements use the grouping operator to change the order in which operations occur (the result appears below each statement):

```
put (2 + 3) * (4 + 5)  
-- 45  
put 2 + (3 * (4 + 5))  
-- 29  
put 2 + 3 * 4 + 5  
-- 19
```

* (multiplication)

Syntax $expression1 * expression2$

Description Math operator; performs an arithmetic multiplication on two numerical expressions. If both expressions are integers, the product is an integer. If either or both expressions are floating-point numbers, the product is a floating-point number.

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

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

```
put 2 * 3
```

The result is 6, which is an integer.

Example This statement multiplies the floating-point numbers 2.0 and 3.1414 and displays the result in the Message window:

```
put 2.0 * 3.1414
```

The result is 6.2832, which is a floating-point number.

+ (addition)

Syntax $expression1 + expression2$

Description Math operator; performs an arithmetic sum on two numerical expressions. If both expressions are integers, the sum is an integer. If either or both expressions are floating-point numbers, the sum is a floating-point number.

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

Example This statement adds the integers 2 and 3 and then displays the result, 5, an integer, in the Message window:

```
put 2 + 3
```

Example This statement adds the floating-point numbers 2.5 and 3.25 and displays the result, 5.7500, a floating-point number, in the Message window:

```
put 2.5 + 3.25
```

/ (division)

Syntax *expression1 / expression2*

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

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

Example This statement divides the integer 22 by 7 and then displays the result in the Message window:

```
put 22 / 7
```

The result is 3. Because both numbers in the division are integers, Lingo rounds the answer down to the nearest integer.

Example This statement divides the floating-point number 22.0 by 7.0 and then displays the result in the Message window:

```
put 22.0 / 7.0
```

The result is 3.1429, which is a floating-point number.

< (less than)

Syntax *expression1 < expression2*

Description Comparison operator; compares two expressions and determines whether *expression1* is less than *expression2* (TRUE), or whether *expression1* is greater than or equal to *expression2* (FALSE).

This operator can compare strings, integers, floating-point numbers, rects, and points. Be aware that comparisons performed on rects or points are handled as if the terms were lists, with each element of the first list compared to the corresponding element of the second list.

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

<= (less than or equal to)

Syntax $expression1 <= expression2$

Description Comparison operator; compares two expressions and determines whether $expression1$ is less than or equal to $expression2$ (TRUE), or whether $expression1$ is greater than $expression2$ (FALSE).

This operator can compare strings, integers, floating-point numbers, rects, and points. Be aware that comparisons performed on rects or points are handled as if the terms were lists, with each element of the first list compared to the corresponding element of the second list.

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

<> (not equal)

Syntax $expression1 <> expression2$

Description Comparison operator; compares two expressions, symbols, or operators and determines whether $expression1$ is not equal to $expression2$ (TRUE), or whether $expression1$ is equal to $expression2$ (FALSE).

This operator can compare strings, integers, floating-point numbers, rects, and points. Be aware that comparisons performed on rects or points are handled as if the terms were lists, with each element of the first list compared to the corresponding element of the second list.

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

= (equals)

Syntax $expression1 = expression2$

Description Comparison operator; compares two expressions, symbols, or objects and determines whether $expression1$ is equal to $expression2$ (TRUE), or whether $expression1$ is not equal to $expression2$ (FALSE).

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

Lists are compared based on the number of elements in the list. The list with more elements is considered larger than the than the list with fewer elements.

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

> (greater than)

Syntax $expression1 > expression2$

Description Comparison operator; compares two expressions and determines whether $expression1$ is greater than $expression2$ (TRUE), or whether $expression1$ is less than or equal to $expression2$ (FALSE).

This operator can compare strings, integers, floating-point numbers, rects, and points. Be aware that comparisons performed on rects or points are handled as if the terms were lists, with each element of the first list compared to the corresponding element of the second list.

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

>= (greater than or equal to)

Syntax $expression1 >= expression2$

Description Comparison operator; compares two expressions and determines whether $expression1$ is greater than or equal to $expression2$ (TRUE), or whether $expression1$ is less than $expression2$ (FALSE).

This operator can compare strings, integers, floating-point numbers, rects, and points. Be aware that comparisons performed on rectangles or points are handled as if the terms were lists, with each element of the first list compared to the corresponding element of the second list.

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

[] (bracket access)

Syntax $textExpression[chunkNumberBeingAddressed]$
 $textExpression[firstChunk..lastChunk]$

Description Operator; allows a chunk expression to be addressed by number. Useful for finding the n th chunk in the expression. The chunk can be a word, line, character, paragraph, or other Text cast member chunk.

Example This outputs the first word of the third line in the text cast member First Names:
`put member("First Names").text.line[3].word[1]`

[] (list)

Syntax	[entry1, entry2, entry3, ...]
Description	List operator; specifies that the entries within the brackets are one of four types of lists: <ul style="list-style-type: none">• Unsorted linear lists• Sorted linear lists• Unsorted property lists• Sorted property lists
	Each entry in a linear list is a single value that has no other property associated with it. Each entry in a property list consists of a property and a value. The property appears before the value and is separated from the value by a colon. You cannot store a property in a linear list. When using strings as entries in a list, enclose the string in quotation marks.
	For example, [6, 3, 8] is a linear list. The numbers have no properties associated with them. However, [#gears:6, #balls:3, #ramps:8] is a property list. Each number has a property—in this case, a type of machinery—associated with it. This property list could be useful for tracking the number of each type of machinery currently on the Stage in a mechanical simulation. Properties can appear more than once in a property list.
	Lists can be sorted in alphanumeric order. A sorted linear list is ordered by the values in the list. A sorted property list is ordered by the properties in the list. You sort a list by using the appropriate command for a linear list or property list. <ul style="list-style-type: none">• In linear lists, symbols and strings are case sensitive.• In property lists, symbols aren't case sensitive, but strings are case sensitive.
	A linear list or property list can contain no values at all. An empty list consists of two square brackets ([]). To create or clear a linear list, set the list to []. To create or clear a property list, set the list to [:].
	You can modify, test, or read items in a list.
	Lingo treats an instance of a list as a reference to the list. This means each instance is the same piece of data, and changing it will change the original. Use the duplicate command to create copies of lists.
	Lists are automatically disposed when they are no longer referred to by any variable. When a list is held within a global variable, it persists from movie to movie.
	You can initialize a list in the on prepareMovie handler or write the list as a field cast member, assign the list to a variable, and then handle the list by handling the variable.

Not all PC keyboards have square brackets. If square brackets aren't available, use the list function to create a linear list.

For a property list, create the list pieces as a string before converting them into a useful list.

```
myListString = numToChar(91) & ":" & numToChar(93)
put myListString
-- "[:]"
myList = myListString.value
put myList
-- [:]
put myList.listP
-- 1
myList[#name] = "Brynn"
put myList
-- [#name: "Brynn"]
```

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

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

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

```
on sortList aList
    alist.sort()
    put aList
end sortList
```

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

Example This statement creates an empty linear list:

```
set x = [ ]
```

Example This statement creates an empty property list:

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

See also add, addAt, append, count(), deleteAt, duplicate() (list function), findPos, findPosNear, getProp(), getAt, getLast(), getPos(), ilk(), list(), max(), min, setAt, setaProp, sort

" (string)

Syntax

"

Description String constant; when used before and after a string, quotation marks indicate that the string is a literal—not a variable, numerical value, or Lingo element. Quotation marks must always surround literal names of cast members, casts, windows, and external files.

Example This statement uses quotation marks to indicate that the string “San Francisco” is a literal string, the name of a cast member:

```
put member("San Francisco").loaded
```

See also QUOTE

¬ (continuation)

Description This symbol is obsolete. Use the \ character instead. See \ (continuation).

\ (continuation)

Syntax

*first part of a statement on this line *

*second part of the statement *

third part of the statement

Description Continuation symbol; when used as the last character in a line, indicates that the statement continues on the next line. Lingo then interprets the lines as one continuous statement.

Example This statement uses the \ character to wrap the statement onto two lines:

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

@ (pathname)

Syntax

@pathReference

Description Pathname operator; defines the path to the current movie’s folder and is valid on both Windows and Macintosh computers.

Identify the current movie’s folder by using the @ symbol followed by one of these pathname separators:

- / (forward slash)
- \ (backslash)
- : (colon)

When a movie is queried to determine its location, the string returned will include the @ symbol.

Be sure to use only the @ symbol when navigating between Director movies or changing the source of a linked media cast member. The @ symbol does not work when the FileIO Xtra or other functions are used outside those available within Director.

You can build on this pathname to specify folders that are one or more levels above or below the current movie's folder. Keep in mind that the @ portion represents the current movie's location, not necessarily the location of the projector.

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

You can use relative pathnames in Lingo to indicate the location of a linked file in a folder different than the movie's folder.

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

```
@/bigFolder  
@:bigFolder  
@|bigFolder
```

Example These are equivalent expressions that specify the file linkedFile, in the subfolder bigFolder, which is in the current movie's folder:

```
@:bigFolder:linkedFile  
@|bigFolder\linkedFile  
@/bigFolder/linkedFile
```

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

```
@//linkedFile
```

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

```
@::linkedFile
```

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

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

See also searchPath, fileName (cast property), fileName (cast member property), fileName (window property)

abbr, abbrev, abbreviated

These elements are used by the date and time functions.

See also

`date()` (system clock)

abort

Syntax

`abort`

Description

Command; tells Lingo to exit the current handler and any handler that called it without executing any of the remaining statements in the handler. This differs from the `exit` keyword, which returns to the handler from which the current handler was called.

The `abort` command does not quit Director.

Example

This statement instructs Lingo to exit the handler and any handler that called it when the amount of free memory is less than 50K:

```
if the freeBytes < 50*1024 then abort
```

See also

`exit`, `halt`, `quit`

abs()

Syntax

`abs (numericExpression)`

Description

Math function; calculates the absolute value of a numerical expression. If *numericExpression* is an integer, its absolute value is also an integer. If *numericExpression* is a floating-point number, its absolute value is also a floating-point number.

The `abs` function has several uses. It can simplify the tracking of mouse and sprite movement by converting coordinate differences (which can be either positive or negative numbers) into distances (which are always positive numbers). The `abs` function is also useful for handling mathematical functions, such as `sqr` and `log`.

Example

This statement determines whether the absolute value of the difference between the current mouse position and the value of the variable `startV` is greater than 30 (since you wouldn't want to use a negative number for distance). If it is, the foreground color of sprite 6 is changed.

```
if (the mouseV - startV).abs > 30 then sprite(6).forecolor = 95
```

actionsEnabled

Syntax	the actionsEnabled of sprite <i>whichFlashSprite</i> the actionsEnabled of member <i>whichFlashMember</i> sprite <i>whichFlashSprite</i> .actionenabled member <i>whichFlashMember</i> .actionenabled
Description	Cast member property and sprite property; controls whether the actions in a Flash movie are enabled (TRUE, default) or disabled (FALSE). This property can be tested and set.
Example	This handler accepts a sprite reference as a parameter, and then toggles the sprite's actionsEnabled property on or off. Dot syntax: <pre>on ToggleActions whichSprite sprite (whichSprite).actionsEnabled = not sprite (whichSprite).actionsEnabled end</pre> Verbose syntax: <pre>on ToggleActions whichSprite set the actionsEnabled of sprite whichSprite = not the actionsEnabled of sprite whichSprite end</pre>

activateApplication

Syntax	on activateApplication
Description	Built-in handler; runs when the projector is brought to the foreground. This handler is useful when a projector runs in a window and the user can send it to the background to work with other applications. When the projector is brought back to the foreground, this handler runs. Any MIAWs running in the projector can also make use of this handler. During authoring, this handler is called only if Animate in Background is turned on in General Preferences. On Windows, this handler is not called if the projector is merely minimized and no other application is brought to the foreground.
Example	This handler plays a sound each time the user brings the projector back to the foreground: <pre>on activateApplication sound(1).queue(member("openSound")) sound(1).play() end</pre>
See also	deactivateApplication, activateWindow, deactivateWindow

on activateWindow

Syntax

```
on activateWindow  
    statement(s)  
end
```

Description System message and event handler; contains statements that run in a movie when the user clicks the inactive window and the window comes to the foreground.

You can use an on activateWindow handler in a script that you want executed every time the movie becomes active.

Clicking the main movie (the main Stage) does not generate an on activateWindow handler.

Example This handler plays the sound Hurray when the window that the movie is playing in becomes active:

```
on activateWindow  
    puppetSound 2, "Hurray"  
end
```

See also activeWindow, close window, on deactivateWindow, frontWindow, on moveWindow, open

activeCastLib

Syntax the activeCastLib

Description System property; indicates which cast was most recently activated. The activeCastLib property's value is the cast's number.

The activeCastLib property is useful when working with the selection castLib property. Use it to determine which cast the selection refers to.

This property can be tested but not set.

Example These statements assign the selected cast members in the most recently selected cast to the variable selectedMembers:

```
castLibOfInterest = the activeCastLib  
selectedMembers = castLib(castLibOfInterest).selection
```

An equivalent way to write this is:

```
selectedMembers = castLib(the activeCastLib).selection
```

activeWindow

Syntax the activeWindow

Description Movie property; indicates which movie window is currently active. For the main movie, activeWindow is the Stage. For a movie in a window, activeWindow is the movie in the window.

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

```
on activateWindow
    set clickedWindow = (the windowlist).getPos(the activeWindow)
    set windowCount = (the windowlist).count
    repeat with x = 1 to windowCount
        if x = clickedWindow then
            (the activeWindow).title = "Active"
        else
            (the windowlist[x]).title = "Inactive"
        end if
    end repeat
end
```

See also on activateWindow, windowList

actorList

Syntax the actorList

Description Movie property; a list of child objects that have been explicitly added to this list. Objects in actorList receive a stepFrame message each time the playback head enters a frame.

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

To clear objects from the actorList, set actorList to [], which is an empty list.

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

Unlike previous versions of Director, actorList is now supported in the Director player for Java.

Example	This statement adds a child object created from the parent script Moving Ball. All three values are parameters that the script requires.
	add the actorList, new(script "MovingBall", 1, 200,200)
Example	This statement displays the contents of actorList in the Message window:
	put the actorList

add

Syntax `linearList.add(value)`
`add linearList, value`

Description List command; for linear lists only, adds the value specified by *value* to the linear list specified by *linearList*. For a sorted list, the value is placed in its proper order. For an unsorted list, the value is added to the end of the list.
This command returns an error when used on a property list.

Note: Don't confuse the add command with the + operator used for addition or the & operator used to concatenate strings.

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

```
bids = [3, 4, 1]  
bids.add(2)
```

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

```
bids.add(2)
```

See also sort

addAt

Syntax	list.AddAt(<i>position</i> , <i>value</i>) addAt <i>list</i> , <i>position</i> , <i>value</i>
Description	List command; for linear lists only, adds a value specified by <i>value</i> to a list at the position specified by <i>position</i> . This command returns an error when used with a property list.
Example	This statement adds the value 8 to the fourth position in the list named bids, which is [3, 2, 4, 5, 6, 7]: bids = [3, 2, 4, 5, 6, 7] bids.addAt(4,8) The resulting value of bids is [3, 2, 4, 8, 5, 6, 7].

addProp

Syntax	list.addProp(<i>property</i> , <i>value</i>) addProp <i>list</i> , <i>property</i> , <i>value</i>
Description	Property list command; for property lists only, adds the property specified by <i>property</i> and its value specified by <i>value</i> to the property list specified by <i>list</i> . For an unsorted list, the value is added to the end of the list. For a sorted list, the value is placed in its proper order. If the property already exists in the list, Lingo creates a duplicate property. You can avoid duplicate properties by using the setaProp command to change the new entry's property. This command returns an error when used with a linear list.
Example	This statement adds the property named kayne and its assigned value 3 to the property list named bids, which contains [#gee: 4, #ohasi: 1]. Because the list is sorted, the new entry is placed in alphabetical order: bids.addProp(#kayne, 3) The result is the list [#gee: 4, #kayne: 3, #ohasi: 1].
Example	This statement adds the entry kayne: 7 to the list named bids, which now contains [#gee: 4, #kayne: 3, #ohasi: 1]. Because the list already contains the property kayne, Lingo creates a duplicate property: bids.addProp(#kayne, 7) The result is the list [#gee: 4, #kayne: 3, #kayne: 7, #ohasi: 1].

addVertex

Syntax	<pre>member(memberRef).AddVertex(indexToAddAt, pointToAddVertex \ {,[controlLocH, controlLocV], [controlLocH, controlLocV]}) addVertex(member memberRef, indexToAddAt, pointToAddVertex \ {,[controlLocH, controlLocV], [controlLocH,controlLocV]})</pre>
Description	<p>Vector shape command; adds a new vertex to a vector shape cast member in the position specified.</p> <p>The horizontal and vertical positions are relative to the origin of the vertex shape cast member.</p> <p>When using the final two optional parameters, you can specify the location of the control handles for the vertex. The control handle location is offset relative to the vertex, so if no location is specified, it will be located at 0 horizontal offset and 0 vertical offset.</p>
Example	<p>This line adds a vertex point in the vector shape Archie between the two existing vertex points, at the position 25 horizontal and 15 vertical:</p> <pre>member("Archie").addVertex(2, point(25, 15))</pre>
See also	<code>vertexList</code> , <code>moveVertex()</code> , <code>addVertex</code> , <code>deleteVertex()</code> , <code>originMode</code>

after

See	put...after command
------------	---------------------

alert

Syntax	<code>alert message</code>
Description	<p>Command; causes a system beep and displays an alert dialog box containing the string specified by <i>message</i> and an OK button. This command is useful for providing error messages of up to 255 characters in your movie.</p> <p>The message must be a string. If you want to include a number variable in an alert, use the string function to convert the variable to a string.</p>
Example	<p>The following statement produces an alert stating that there is no CD-ROM drive connected:</p> <pre>alert "There is no CD-ROM drive connected."</pre>
Example	<p>This statement produces an alert stating that a file was not found:</p> <pre>alert "The file" && QUOTE & filename & QUOTE && "was not found."</pre>
See also	<code>string()</code> , <code>alertHook</code>

alertHook

Syntax the alertHook

Description System property; specifies a parent script that contains the on alertHook handler. Use alertHook to control the display of alerts about file errors or Lingo script errors. When an error occurs and a parent script is assigned to alertHook, Director runs the on alertHook handler in the parent script.

Although it is possible to place on alertHook handlers in movie scripts, it is strongly recommended that you place an on alertHook handler in a behavior or parent script to avoid unintentionally calling the handler from a wide variety of locations and creating confusion about where the error occurred.

Because the on alertHook handler runs when an error occurs, avoid using the on alertHook handler for Lingo that isn't involved in handling an error. For example, the on alertHook handler is a bad location for a go to movie statement.

The on alertHook handler is passed an instance argument and two string arguments that describe the error. Depending on the Lingo within it, the on alertHook handler can ignore the error or report it in another way.

Example The following statement specifies that the parent script Alert is the script that determines whether to display alerts when an error occurs. If an error occurs, Lingo assigns the error and message strings to the field cast member Output and returns the value 1.

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

-- parent script "Alert"
on alertHook me, err, msg
    member("Output").text = err && msg
    return 1
end
```

alignment

Syntax

```
member(whichCastMember).alignment
```

the alignment of member *whichCastMember*

Description

Cast member property; determines the alignment used to display characters within the specified cast member. This property appears only to field and text cast members containing characters, if only a space.

For field cast members, the value of the property is a string consisting of one of the following: left, center, or right.

For text cast members, the value of the property is a symbol consisting of one of the following: #left, #center, #right, or #full.

The parameter *whichCastMember* can be either a cast name or a cast number.

This property can be tested and set. For text cast members, the property can be set on a per-paragraph basis.

Example

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

Dot syntax:

```
characterAlign = member("Rokujo Speaks").alignment
```

Verbose syntax:

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

Example

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

Dot syntax:

```
repeat with i = 1 to 3  
    member("Rove").alignment = ("left center right").word[i]  
end repeat
```

Verbose syntax:

```
repeat with i = 1 to 3  
    set the alignment of member "Rove" to word i of "left center right"  
end repeat
```

See also

text, font, lineHeight (cast member property), fontSize, fontStyle, & (concatenator), && (concatenator)

allowCustomCaching

Syntax	the allowCustomCaching
Description	Movie property; will contain information regarding a private cache in future versions of Director.
	This property defaults to TRUE, and can be tested and set.
See also	allowGraphicMenu, allowSaveLocal, allowTransportControl, allowVolumeControl, allowZooming

allowGraphicMenu

Syntax	the allowGraphicMenu
Description	Movie property; sets the availability of the graphic controls in the context menu when playing the movie in a Shockwave environment.
	Set this property to FALSE if you would rather have a text menu displayed than the graphic context menu.
	This property defaults to TRUE, and can be tested and set.
Example	the allowGraphicMenu = 0
See also	allowSaveLocal, allowTransportControl, allowVolumeControl, allowZooming

allowSaveLocal

Syntax	the allowSaveLocal
Description	Movie property; sets the availability of the Save control in the context menu when playing the movie in a Shockwave environment.
	This property is provided to allow for enhancements in future versions of Shockwave.
	This property defaults to TRUE, and can be tested and set.
See also	allowGraphicMenu, allowTransportControl, allowVolumeControl, allowZooming

allowTransportControl

Syntax	the allowTransportControl
Description	Movie property; This property is provided to allow for enhancements in future versions of Shockwave.
	This property defaults to TRUE, and can be tested and set.
See also	allowGraphicMenu, allowSaveLocal, allowTransportControl, allowVolumeControl, allowZooming

allowVolumeControl

Syntax the allowVolumeControl

Description Movie property; sets the availability of the volume control in the context menu when playing the movie in a Shockwave environment.

When set to TRUE one or the other volume control is active, and is disabled when the property is set to FALSE.

This property defaults to TRUE, and can be tested and set.

See also allowGraphicMenu, allowSaveLocal, allowTransportControl, allowZooming

allowZooming

Syntax the allowZooming

Description Movie property; determines whether the movie may be stretched or zoomed by the user when playing back in Shockwave and ShockMachine. Defaults to TRUE. This property can be tested and set. Set this property to FALSE to prevent users from changing the size of the movie in browsers and ShockMachine.

See also allowGraphicMenu, allowSaveLocal, allowTransportControl, allowVolumeControl

alphaThreshold

Syntax member(*whichMember*).alphaThreshold

the alphaThreshold of member *whichMember*

Description Bitmap cast member property; governs how the bitmap's alpha channel affects hit detection. This property is a value from 0 to 255, that exactly matches alpha values in the alpha channel for a 32-bit bitmap image.

For a given alphaThreshold setting, Director detects a mouse click if the pixel value of the alpha map at that point is equal to or greater than the threshold. Setting the alphaThreshold to 0 makes all pixels opaque to hit detection regardless of the contents of the alpha channel.

See also useAlpha

ancestor

Syntax

```
property {optionalProperties} ancestor
```

Description

Object property; allows child objects and behaviors to use handlers that are not contained within the parent script or behavior.

The ancestor property is typically used with two or more parent scripts. You can use this property when you want child objects and behaviors to share certain behaviors that are inherited from an ancestor, while differing in other behaviors that are inherited from the parents.

For child objects, the ancestor property is usually assigned in the on new handler within the parent script. Sending a message to a child object that does not have a defined handler forwards that message to the script defined by the ancestor property.

If a behavior has an ancestor, the ancestor receives mouse events such as mouseDown and mouseWithin.

The ancestor property lets you change behaviors and properties for a large group of objects with a single command.

The ancestor script can contain independent property variables that can be obtained by child objects. To refer to property variables within the ancestor script, you must use this syntax:

```
me.propertyVariable = value
```

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

```
me.legCount = 4
```

Use the syntax the *variableName* of *scriptName* to access property variables that are not contained within the current object. This statement allows the variable myLegCount within the child object to access the property variable legCount within the ancestor script:

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

Example

Each of the following scripts is a cast member. The ancestor script Animal and the parent scripts Dog and Man interact with one another to define objects.

The first script, Dog, sets the property variable breed to Mutt, sets the ancestor of Dog to the Animal script, and sets the legCount variable that is stored in the ancestor script to 4:

```
property breed, ancestor  
on new me  
    set breed = "Mutt"  
    set the ancestor of me to new(script "Animal")  
    set the legCount of me to 4  
    return me  
end
```

The second script, Man, sets the property variable race to Caucasian, sets the ancestor of Man to the Animal script, and sets the legCount variable that is stored in the ancestor script to 2:

```
property race, ancestor  
on new me  
    set race to "Caucasian"  
    set the ancestor of me to new(script "Animal")  
    set the legCount of me to 2  
    return me  
end
```

See also [new\(\); me, property](#)

and

Syntax *logicalExpression1 and logicalExpression2*

Description Logical operator; determines whether both *logicalExpression1* and *logicalExpression2* are TRUE (1), or whether either or both expressions are FALSE (0).

The precedence level of this logical operator is 4.

Example This statement determines whether both logical expressions are TRUE and displays the result in the Message window:

```
put 1 < 2 and 2 < 3
```

The result is 1, which is the numerical equivalent of TRUE.

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

```
put 1 < 2 and 2 < 1  
-- 0
```

See also [not and or](#)

antiAlias

Syntax

```
member(whichMember).antiAlias  
sprite(whichVectorSprite).antiAlias
```

Description

Cast member property; controls whether a text, Vector shape, or Flash cast member is rendered using anti-aliasing to produce high-quality rendering, but possibly slower playback of the movie. The antiAlias property is TRUE by default.

For vector shapes, TRUE is the equivalent of the #high quality setting for a Flash asset, and FALSE is the equivalent of #low.

The antiAlias property may also be used as a sprite property only for Vector shape sprites.

This property can be tested and set.

Example

This behavior checks the color depth of the computer on which the movie is playing. If the color depth is set to 8 bits or less (256 colors), the script sets the antiAlias of the sprite to FALSE.

```
property spriteNum  
on beginsprite me  
    if the colorDepth <= 8 then  
        sprite(spriteNum).antiAlias = FALSE  
    end if  
end
```

See also

antiAliasThreshold, quality

antiAliasThreshold

Syntax

```
member(whichTextMember).antiAliasThreshold
```

Description

Text cast member property; this setting controls the point size at which automatic anti-aliasing takes place in a text cast member. This has an effect only when the antiAlias property of the text cast member is set to TRUE.

The setting itself is an integer indicating the font point size at which the anti-alias takes place.

This property defaults to 14 points.

See also

antiAlias

append

Syntax `list.append(value)`

`append list, value`

Description List command; for linear lists only, adds the specified value to the end of a linear list. This differs from the add command, which adds a value to a sorted list according to the list's order.

This command returns a script error when used with a property list.

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

```
set bids = [1, 3, 4]  
bids.append(2)
```

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

See also `add`, `sort`

applicationPath

Syntax `the applicationPath`

Description System property; determines the path or location of the folder containing the running copy of the Director application during authoring, or the folder containing the projector during run time. The property value is a string.

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

The Director player for Java doesn't support this property, nor does Shockwave.

This property can be tested but not set.

Example This statement displays the pathname for the folder that contains the Director application.

```
put the applicationPath  
--"Z:\Program Files\Macromedia\Director"
```

Example This statement opens the movie Sunset Boulevard in a window (on a Windows machine):

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

See also `@(pathname)`, `moviePath`

appMinimize

Syntax appMinimize

Description Command; on Windows, appMinimize causes a projector to minimize to the Windows Task Bar. On the Macintosh, appMinimize causes a projector to be hidden. Once hidden, the projector may be re-opened from the Macintosh application menu.

This is useful for projectors and MIAW's that play back without a title bar.

See also windowType

atan()

Syntax *(number).atan*

atan (number)

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

Example This statement displays the arctangent of 1:

(1).atan

The result, to four decimal places, is 0.7854, or approximately $\pi/4$.

Note that most trigonometric functions use radians, so you may want to convert from degrees to radians.

Example This handler lets you convert between degrees and radians:

```
on DegreesToRads degreeValue  
    return degreeValue * PI/180  
end
```

The handler displays the conversion of 30 degrees to radians in the Message window:

```
put DegreesToRads(30)  
-- 0.5236
```

See also cos(), PI, and sin()

autoMask

Syntax

```
member(whichCursorCastMember).autoMask
```

the autoMask of member *whichCastMember*

Description

Cast member property; specifies whether the white pixels in the animated color cursor cast member *whichCursorCastMember* are transparent, allowing the background to show through (TRUE, default), or opaque (FALSE).

Example

In this script, when the custom animated cursor stored in cast member 5 enters the sprite, the automask is turned on so that the background of the sprite will show through the white pixels. When the cursor leaves the sprite, the automask is turned off.

Using dot syntax, the script is written as:

```
on mouseEnter  
    member 5.autoMask = TRUE  
end
```

```
on mouseLeave  
    member 5.autoMask = FALSE  
end
```

Using traditional Lingo syntax, the script is written as:

```
on mouseEnter  
    set the autoMask of member 5 = TRUE  
end  
  
on mouseLeave  
    set the autoMask of member 5 = FALSE  
end
```

autoTab

Syntax

`member(whichCastMember).autoTab`

the autoTab of member *whichCastMember*

Description

Cast member property; determines the effect that pressing the Tab key has on the editable field or text cast member specified by *whichCastMember*. The property can be made active (TRUE) or inactive (FALSE). Tabbing order depends on sprite number order, not position on the Stage.

This property is always TRUE in an applet created with the Save as Java feature of Director.

Example

This statement causes the cast member Comments to automatically advance the insertion point to the next editable field or text sprite after the user presses Tab.

Dot syntax:

`member ("Comments").autotab = TRUE`

Verbose Lingo syntax:

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

backColor

Syntax

`member(whichCastMember).backColor = colorNumber`

set the backColor of member *whichCastMember* to *colorNumber*

`sprite(whichSprite).backColor`

the backColor of sprite *whichSprite*

Description

Cast member and sprite property; sets the background color of the specified cast member or sprite according to the color value assigned.

- For cast members: it affects field or button cast member displays.
- For sprites: setting the backColor of a sprite is the same as choosing the background color from the tool palette when the sprite is selected on the Stage. For the value that Lingo sets to last beyond the current sprite, the sprite must be a puppet. The background color applies only to 1-bit bitmap and shape cast members.

The backColor value ranges from 0 to 255 for 8-bit color and from 0 to 15 for 4-bit color. The numbers correspond to the index number of the background color in the current palette. (A color's index number appears in the color palette's lower left corner when you click the color.)

You should not apply this property to bitmap cast members deeper than 1-bit, as the results are difficult to predict.

For a movie that plays back as an applet created with the Save as Java feature of Director, specify colors for backColor using the decimal equivalent of the 24-bit hexadecimal values used in an HTML document.

For example, the hexadecimal value for pure red, FF0000, is equivalent to 16711680 in decimal numbers. This statement specifies pure red as a cast member's background color:

```
set the backColor of member = 16711680
```

This property can be tested and set.

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

Dot syntax:

```
member(1).backColor = 250
```

Verbose Lingo syntax:

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

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

```
oldColor = sprite (5).backColor
```

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

```
sprite(10 + random(3)).backColor = 36
```

See also bgColor, color (sprite property)

backgroundColor

Syntax `member(whichVectorMember).backgroundColor`
the backgroundColor of member *whichVectorMember*

Description Vector shape cast member property; sets the background color of the specified cast member or sprite to the RGB color value assigned.

This property can be both tested and set.

Example `member("Archie").backgroundColor= rgb(255,255,255)`
See also bgColor

BACKSPACE

Syntax BACKSPACE

Description Constant; represents the Backspace key. This key is labeled Backspace in Windows and Delete on the Macintosh.

Example This on keyDown handler checks whether the Backspace key was pressed and, if it was, calls the handler clearEntry:

```
on keyDown
    if the key = BACKSPACE then clearEntry
        stopEvent
    end keyDown
```

beep

Syntax beep {*numberOfTimes*}

Description Command; causes the computer's speaker to beep the number of times specified by *numberOfTimes*. If *numberOfTimes* is missing, the beep occurs once.

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

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

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

beepOn

Syntax the beepOn

Description Movie property; determines whether the computer automatically beeps when the user clicks on anything except an active sprite (TRUE), or not (FALSE, default).

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

This property can be tested and set.

Example This statement sets beepOn to TRUE:

```
the beepOn = TRUE
```

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

```
the beepOn = not the beepOn
```

before

See put...before

beginRecording

Syntax beginRecording

Description Keyword; starts a Score generation session. Only one update session in a movie can be active at a time.

Every beginRecording keyword must be matched by an endRecording keyword that ends the Score generation session.

Example When used in the following handler, the beginRecording keyword begins a Score generation session that animates the cast member Ball by assigning the cast member to sprite channel 20 and then moving the sprite horizontally and vertically over a series of frames. The number of frames is determined by the argument `numberOfFrames`.

```
on animBall numberOfFrames
    beginRecording
        horizontal = 0
        vertical = 100
        repeat with i = 1 to numberOfFrames
            go to frame i
            sprite(20).member = member "Ball"
            sprite(20).locH = horizontal
            sprite(20).locV = vertical
            sprite(20).type = 1
            sprite(20).foreColor = 255
            horizontal = horizontal + 3
            vertical = vertical + 2
            updateFrame
        end repeat
    endRecording
end
```

See also endRecording, updateFrame, scriptNum, tweened

on beginSprite

Syntax

```
on beginSprite  
    statement(s)  
end
```

Description

System message and event handler; contains statements that run when the playback head moves to a frame that contains a sprite that was not previously encountered. Like endSprite, this event is generated only one time, even if the playback head loops on a frame, since the trigger is a sprite not previously encountered by the playback head. The event is generated before prepareFrame. Director creates instances of any behavior scripts attached to the sprite when the beginSprite message is sent.

The object reference me is passed to this event if it is used in a behavior. The message is sent to behaviors and frame scripts.

If a sprite begins in the first frame that plays in the movie, the beginSprite message is sent after the prepareMovie message but before the prepareFrame and startMovie messages.

Note: Be aware that some sprite properties, such as the rect sprite property, may not be accessible in a beginSprite handler. This is because the property needs to be calculated, which is not done until the sprite is drawn.

The go, play, and updateStage commands are disabled in an on beginSprite handler.

Example

This handler plays the sound cast member Stevie Wonder when the sprite begins:

```
on beginSprite me  
    puppetSound "Stevie Wonder"  
end
```

See also

on endSprite, on prepareFrame, scriptInstanceList

bgColor

Syntax

```
sprite(whichSpriteNumber).bgColor  
the bgColor of sprite whichSpriteNumber  
the bgColor of the stage  
(the stage).bgColor
```

Description

Sprite property and system property; determines the background color of the sprite specified by *whichSprite* or the color of the Stage. Setting the bgColor sprite property is equivalent to choosing the background color from the Tools window when the sprite is selected on the Stage. Setting the bgColor property for the Stage is equivalent to setting the color in the Movie Properties dialog box.

The sprite property has the equivalent functionality of the backColor sprite property, but the color value returned is a color object of whatever type has been set for that sprite.

This property can be tested and set.

Example This example sets the color of the Stage to an RGB value.

Dot syntax:

```
(the stage).bgColor = rgb(255, 153, 0)
```

Verbose Lingo syntax:

```
set the bgColor of the stage = rgb(255, 153, 0)
```

See also [color\(\)](#), [backColor](#), [backgroundColor](#), [stageColor](#)

bitAnd()

Syntax `bitAnd(integer1, integer2)`

Description Function; converts the two specified integers to 32-bit binary numbers and returns a binary number whose digits are 1's in the positions where both numbers had a 1, and 0's in every other position. The result is the new binary number, which Lingo displays as a base 10 integer.

Integer	Binary number (abbreviated)
6	00110
7	00111
Result	
6	00110

Example This statement compares the binary versions of the integers 6 and 7 and returns the result as an integer:

```
put bitAnd(6, 7)  
-- 6
```

See also [bitNot\(\)](#), [bitOr\(\)](#), [bitXor\(\)](#)

bitmapSizes

Syntax member(*whichFontMember*).bitmapSizes

the bitmapSizes of member *whichFontMember*

Description Font cast member property; returns a list of the bitmap point sizes that were included when the font cast member was created.

Example This statement displays the bitmap point sizes that were included when cast member 11 was created:

```
put member(11).bitmapSizes  
-- [12, 14, 18]
```

See also recordFont, characterSet, originalFont

bitNot()

Syntax (*integer*).bitNot

bitNot(*integer*)

Description Function; converts the specified integer to a 32-bit binary number and reverses the value of each binary digit, replacing 1's with 0's and 0's with 1's. The result is the new binary number, which Lingo displays as a base 10 integer.

Integer	Binary number
1	00000000000000000000000000000001
Result	
-2	11111111111111111111111111111110

Example This statement inverts the binary representation of the integer 1 and returns a new number.

```
put (1).bitNot  
-- -2
```

See also bitAnd(), bitOr(), bitXor()

bitOr()

Syntax

bitOr(*integer1*, *integer2*)

Description

Function; converts the two specified integers to 32-bit binary numbers and returns a binary number whose digits are 1's in the positions where either number had a 1, and 0's in every other position. The result is the new binary number, which Lingo displays as a base 10 integer.

Integer	Binary number (abbreviated)
5	0101
6	0110
Result	
7	0111

Example

This statement compares the 32-bit binary versions of 5 and 6 and returns the result as an integer:

```
put bitOr(5, 6)  
-- 7
```

See also

[bitNot\(\)](#), [bitAnd\(\)](#), [bitXor\(\)](#)

bitRate

Syntax

member(*whichCastMember*).bitRate

the bitRate of member *whichCastMember*

Description

Shockwave Audio (SWA) cast member property; returns the bit rate, in kilobits per second (Kbps), of the specified SWA cast member that has been preloaded from the server.

The bitRate member property returns 0 until streaming begins.

Example This behavior outputs the bit rate of an SWA cast member when the sprite is first encountered.

Dot syntax:

```
property spriteNum
```

```
on beginSprite me
    memName = sprite(spriteNum).member.name
    put "The bitRate of member"&&memName&&"is"&&member(memName).bitRate
end
```

Verbose syntax:

```
property spriteNum
```

```
on beginSprite me
    memName = sprite(spriteNum).member.name
    put "The bitRate of member"&&memName&&"is"&&member(memName).bitRate
end
```

bitsPerSample

Syntax

```
member(whichCastMember).bitsPerSample
```

the bitsPerSample of member *whichCastMember*

Description Shockwave Audio (SWA) cast member property; indicates the bit depth of the original file that has been encoded for Shockwave Audio (SWA). This property is available only after the SWA sound begins playing or after the file has been preloaded using the preLoadBuffer command.

This property can be tested but not set.

Example

This statement assigns the original bit rate of the file used in SWA streaming cast member Paul Robeson to the field cast member How Deep.

Dot syntax:

```
put member "Paul Robeson".bitsPerSample into member "How Deep"
```

Verbose syntax:

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

bitXor()

Syntax

bitXor(*integer1*, *integer2*)

Description

Function; converts the two specified integers to 32-bit binary numbers and returns a binary number whose digits are 1's in the positions where the given numbers' digits do not match, and 0's in the postions where the digits are the same. The result is the new binary number, which Lingo displays as a base 10 integer.

Integer	Binary number (abbreviated)
5	0101
6	0110
Result	
3	0011

Example

This statement compares the 32-bit binary versions of 5 and 6 and returns the result as an integer:

```
put bitXor(5, 6)  
-- 3
```

See also

[bitNot\(\)](#), [bitOr\(\)](#), [bitAnd\(\)](#)

blend

Syntax

sprite(*whichSprite*).blend

the blend of sprite *whichSprite*

Description

Sprite property; sets or determines a sprite's blend value, from 0 to 100, corresponding to the blend values in the Sprite Properties dialog box.

The possible colors depend on the colors available in the palette, regardless of the monitor's color depth.

The Director player for Java supports the blend sprite property for bitmap sprites only.

For best results, use the blend ink with images that have a color depth greater than 8-bit.

Example	This statement sets the blend value of sprite 3 to 40 percent.
	Dot syntax:
	<code>sprite(3).blend = 40</code>
	Verbose syntax:
	set the blend of sprite 3 to 40
Example	This statement displays the blend value of sprite 3 in the Message window:
	put the blend of sprite 3
See also	blendLevel

blendLevel

Syntax	<code>sprite(<i>whichSpriteNumber</i>).blendLevel</code>
	the blendLevel of sprite <i>whichSpriteNumber</i>
Description	Sprite property; allows the current blending value of a sprite to be set or accessed. The possible range of values is from 0 to 255. This differs from the Sprite Inspector, which shows values in the range 0 to 100. The results are the same, the scales simply differ.
	This property is the equivalent of the <code>blend</code> sprite property.
Example	<code>sprite(3).blendlevel = 99</code>
See also	blend

border

Syntax	<code>member(<i>whichFieldCastmember</i>).border</code>
	the border of member <i>whichFieldCastmember</i>
Description	Field cast member property; indicates the width, in pixels, of the border around the specified field cast member.
Example	This statement makes the border around the field cast member Title 10 pixels wide.
	Dot syntax:
	<code>member("Title").border = 10</code>
	Verbose syntax:
	set the border of member "Title" to 10

bottom

Syntax

`sprite(whichSprite).bottom`

the bottom of sprite *whichSprite*

Description

Sprite property; specifies the bottom vertical coordinate of the bounding rectangle of the sprite specified by *whichSprite*.

When a movie plays back as an applet, this property's value is measured from the upper left corner of the applet.

This property can be tested and set.

Example

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

Dot syntax:

`set lowest = sprite (i + 1).bottom`

Verbose syntax:

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

See also

height, left, locH, locV, right, top, and width

bottomSpacing

Syntax

`chunkExpression.bottomSpacing`

Description

Text cast member property; enables you to specify additional spacing applied to the bottom of each paragraph in the *chunkExpression* portion of the text cast member.

The value itself is an integer, where less than 0 indicates less spacing between paragraphs and greater than 0 indicates more spacing between paragraphs.

The default value is 0, which results in default spacing between paragraphs.

Note: This property, like all text cast member properties, supports only dot syntax.

Example

This example adds spacing after the first paragraph in cast member News Items.
`member("News Items").paragraph[1].bottomSpacing=20`

See also

topSpacing

boxDropShadow

Syntax	member(<i>whichCastMember</i>).boxDropShadow the boxDropShadow of member <i>whichCastMember</i>
Description	Cast member property; determines the size, in pixels, of the drop shadow for the box of the field cast member specified by <i>whichCastMember</i> .
Example	This statement makes the drop shadow of field cast member Title 10 pixels wide. Dot syntax: member("Title").boxDropShadow = 10 Verbose syntax: set the boxDropShadow of member "Title" to 10

boxType

Syntax	member(<i>whichCastMember</i>).boxType the boxType of member <i>whichCastMember</i>
Description	Cast member property; determines the type of text box used for the specified cast member. The possible values are #adjust, #scroll, #fixed, and #limit.
Example	This statement makes the box for field cast member Editorial a scrolling field. Dot syntax: member("Editorial").boxType = #scroll Verbose syntax: set the boxType of member "Editorial" to #scroll

breakLoop()

Syntax	sound(<i>channelNum</i>).breakLoop()
Description	This function causes the currently looping sound in channel <i>channelNum</i> to stop looping and play through to its <i>endTime</i> . If there is no current loop, this function has no effect.
Example	This handler causes the background music looping in sound channel 2 to stop looping and play through to its end: on continueBackgroundMusic sound(2).breakLoop() end
See also	end , loopCount , loopEndTime , loopsRemaining , loopStartTime

broadcastProps

Syntax

```
member(whichVectorOrFlashMember).broadcastProps
```

the broadcastProps of member *whichVectorOrFlashMember*

Description

Cast member property; controls whether changes made to a Flash or Vector shape cast member are immediately broadcast to all of its sprites currently on the Stage (TRUE) or not (FALSE).

When this property is set to FALSE, changes made to the cast member are used only as defaults for new sprites and don't affect sprites on the Stage.

The default value for this property is TRUE, and it can be both tested and set.

Example

This frame script assumes that a Flash movie cast member named Navigation Movie has been set up with its broadcastProps property set to FALSE. The script momentarily allows changes to a Flash movie cast member to be broadcast to its sprites currently on the Stage. It then sets the viewScale property of the Flash movie cast member, and that change is broadcast to its sprite. The script then prevents the Flash movie from broadcasting changes to its sprites.

Dot syntax:

```
on enterFrame
    member("Navigation Movie").broadcastProps = TRUE
    member("Navigation Movie").viewScale = 200
    member("Navigation Movie").broadcastProps = FALSE
end
```

Verbose syntax:

```
on enterFrame
    set the broadcastProps of member "Navigation Movie" = TRUE
    set the viewScale of member "Navigation Movie" = 200
    set the broadcastProps of member "Navigation Movie" = FALSE
end
```

browserName()

Syntax

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

Description

System property, command, and function; specifies the path or location of the browser. You can use the FileIO Xtra to display a dialog box that allows the user to search for a browser. The displayOpen() method of the FileIO Xtra is useful for displaying an Open dialog box.

The form browserName() returns the name of the currently specified browser. Placing a pathname, like one found using the FileIO Xtra, as an argument in the form browserName(*fullPathToApplication*) allows the property to be set. The form browserName(#enabled, *trueOrFalse*) determines whether the specified browser launches automatically when the goToNetPage command is issued.

This command is only useful playing back in a projector or in Director, and has no effect when playing back in a browser.

This property can be tested and set.

Example

This statement refers to the location of the Netscape browser:

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

Example

This statement displays the browser name in a Message window:

```
put browserName()
```

bufferSize

Syntax

```
member(whichFlashMember).bufferSize  
the bufferSize of member whichFlashMember
```

Description

Flash cast member property; controls how many bytes of a linked Flash movie are streamed into memory at one time. The bufferSize member property can have only integer values. This property has an effect only when the cast member's preload property is set to FALSE.

This property can be tested and set. The default value is 32,768 bytes.

Example This startMovie handler sets up a Flash movie cast member for streaming and then sets its bufferSize property.

Dot syntax:

```
on startMovie
    member("Flash Demo").preload = FALSE
    member("Flash Demo").bufferSize = 65536
end
```

Verbose syntax:

```
on startMovie
    set the preload of member "Flash Demo" = FALSE
    set the bufferSize of member "Flash Demo" = 65536
end
```

See also bytesStreamed, preLoadRAM, stream, streamMode

buttonsEnabled

Syntax sprite(*whichFlashSprite*).buttonsEnabled

the buttonsEnabled of sprite *whichFlashSprite*

member(*whichFlashMember*).buttonsEnabled

the buttonsEnabled of member *whichFlashMember*

Description Flash cast member property and sprite property; controls whether the buttons in a Flash movie are active (TRUE, default) or inactive (FALSE). Button actions are triggered only when the actionsEnabled property is set to TRUE.

This property can be tested and set.

Example This handler accepts a sprite reference and toggles the sprite's buttonsEnabled property on or off.

Dot syntax:

```
on ToggleButtons whichSprite
    sprite(whichSprite).buttonsEnabled = not sprite(whichSprite).buttonsEnabled
end
```

Verbose syntax:

```
on ToggleButtons whichSprite
    set the buttonsEnabled of sprite whichSprite = not the buttonsEnabled of \
        sprite whichSprite
end
```

See also actionsEnabled

buttonStyle

Syntax the buttonStyle

Description Movie property; determines the visual response of buttons when the user rolls the pointer off them. This property applies only to buttons created with the Button tool in the Tool palette.

The buttonStyle property can have these values:

- 0 (list style: default)—Subsequent buttons are highlighted when the pointer passes over them. Releasing the mouse button activates the script associated with that button.
- 1 (dialog style)—Only the first button clicked is highlighted. Subsequent buttons are not highlighted. Releasing the mouse button while the pointer is over a button other than the original button clicked does not activate the script associated with that button.

This property can be tested and set in any type of script.

Example The following statement sets the buttonStyle property to 1:

```
the buttonStyle = 1
```

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

```
buttonStyleValue = the buttonStyle
```

See also checkBoxAccess, checkBoxType

buttonType

Syntax member(*whichCastMember*).buttonType

the buttonType of member *whichCastMember*

Description Button cast member property; indicates the specified button cast member's type. Possible values are #pushButton, #checkbox, and #radioButton. This property applies only to buttons created with the button tool in the Tool palette.

Example This statement makes the button cast member Editorial a check box.

Dot syntax:

```
member("Editorial").buttonType = #checkbox
```

Verbose syntax:

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

bytesStreamed

Syntax

```
member(whichFlashOrSWAMember).bytesStreamed
```

the bytesStreamed of member *whichFlashOrSWAMember*

Description

Flash and Shockwave Audio cast member property; indicates the number of bytes of the specified cast member that have been loaded into memory. The bytesStreamed property returns a value only when the Director movie is playing. It returns an integer value.

This property can be tested but not set.

Example

This handler accepts a cast member reference as a parameter, and it then uses the stream command to load the cast member into memory. Every time it streams part of the cast member into memory, it uses the bytesStreamed property to report in the Message window how many bytes have been streamed.

Dot syntax:

```
on fetchMovie whichFlashMovie
    repeat while member(whichFlashMovie).percentStreamed < 100
        stream(member whichFlashMovie)
        put "Number of bytes streamed:" && member(whichFlashMovie).bytesStreamed
    end repeat
end
```

Verbose syntax:

```
on fetchMovie whichFlashMovie
    repeat while the percentStreamed of member whichFlashMovie < 100
        stream(member whichFlashMovie)
        put "Number of bytes streamed:" && the bytesStreamed of member \
            whichFlashMovie
    end repeat
end
```

See also

bufferSize, bytesStreamed, percentStreamed, stream

cacheDocVerify()

Syntax

```
cacheDocVerify #setting  
cacheDocVerify()
```

Description

Function; sets how often the contents of a page on the Internet are refreshed with information from the projector's cache. Possible values are #once (default) and #always.

Specifying #once tells a movie to get a file from the Internet once and then use the file from the cache without looking for an updated version on the Internet.

Specifying #always tells a movie to try to get an updated version of the file each time the movie calls a URL.

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

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

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

See also

cacheSize(), clearCache

cacheSize()

Syntax

```
cacheSize Size  
cacheSize()
```

Description

Function and command; sets Director's cache size. The value is in kilobytes.

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

Example

This handler checks whether the browser's cache setting is less than 1 MB. If it is, the handler displays an alert and sets the cache size to 1 MB:

```
on checkCache if  
    cacheSize()<1000 then  
        alert "increasing cache to 1MB"  
        cacheSize 1000  
    end if  
end
```

See also

cacheDocVerify(), clearCache

call

Syntax

```
call #handlerName, script, {args...}  
call (#handlerName, scriptInstance, {args...})
```

Description

Command; sends a message that invokes a handler in specified scripts where *handlerName* is the name of the handler to be activated, *script* references the script or a list of scripts, and *args* are any optional parameters to be passed to the handler.

If *script* is a single script instance, an error alert occurs if the handler is not defined in the script's ancestor script.

If *script* is a list of script instances, the message is sent to each item in the list in turn; if the handler is not defined in the ancestor script, no alert is generated.

The call command can use a variable as the name of the handler. Messages passed using call are not passed to other scripts attached to the sprite, cast member scripts, frame scripts, or movie scripts.

Example This handler sends the message bumpCounter to the first behavior script attached to sprite 1:

```
on mouseDown me  
    -- get the reference to the first behavior of sprite 1  
    set xref = getAt (the scriptInstanceList of sprite 1,1)  
    -- run the bumpCounter handler in the referenced script,  
    -- with a parameter  
    call (#bumpCounter, xref, 2)  
end
```

Example This example shows how a call statement can call handlers in a behavior or parent script and its ancestor.

- This is the parent script:

```
-- script Man  
property ancestor  
on new me  
    set ancestor = new(script "Animal", 2)  
    return me  
end  
on run me, newTool  
    put "Man running with "&the legCount of me&" legs"  
end
```

- This is the ancestor script:

```
-- script Animal
property legCount
on new me, newLegCount
    set legCount = newLegCount
    return me
end
on run me
    put "Animal running with "& legCount &" legs"
end
on walk me
    put "Animal walking with "& legCount &" legs"
end
```

- The following statements use the parent script and ancestor script.

This statement creates an instance of the parent script:

```
set m = new(script "man")
```

This statement makes the man walk:

```
call #walk, m
-- "Animal walking with 2 legs"
```

This statement makes the man run:

```
set msg = #run
call msg, m
-- "Man running with 2 legs and rock"
```

This statement creates a second instance of the parent script:

```
set m2 = new(script "man")
```

This statement sends a message to both instances of the parent script:

```
call msg, [m, m2]
-- "Man running with 2 legs "
-- "Man running with 2 legs "
```

callAncestor

Syntax

```
callAncestor handlerName, script, {args...}
```

Description

Command; sends a message to a child object's ancestor script, where *handlerName* is the name of the handler to be activated, *script* references the script instance or a list of script instances, and *args* are any optional parameters to be passed to the handler.

If *script* is a single script instance, an error alert occurs if the handler is not defined in the ancestor of the script.

If *script* is a list of script instances, the message is sent to each item in the list in turn. In this case, if the handler is not defined in an ancestor script, no alert is generated.

Ancestors can, in turn, have their own ancestors.

When you use callAncestor, the name of the handler can be a variable, and you can explicitly bypass the handlers in the primary script and go directly to the ancestor script.

Example

This example shows how a callAncestor statement can call handlers in the ancestor of a behavior or parent script.

- This is the parent script:

```
-- script "man"
property ancestor
on new me, newTool
    set ancestor = new(script "Animal", 2)
    return me
end
on run me
    put "Man running with "&the legCount of me&"legs"
end
```

- This is the ancestor script:

```
-- script "animal"
property legCount
on new me, newLegCount
    set legCount = newLegCount
    return me
end
on run me
    put "Animal running with "& legCount &" legs"
end
on walk me
    put "Animal walking with "& legCount &" legs"
end
```

- The following statements use the parent script and ancestor script.

This statement creates an instance of the parent script:

```
set m = new(script "man")
```

This statement makes the man walk:

```
call #walk, m
-- "Animal walking with 2 legs"
```

This statement makes the man run:

```
set msg = #run
callAncestor msg, m
-- "Animal running with 2 legs"
```

This statement creates a second instance of the parent script:

```
set m2 = new(script "man")
```

This statement sends a message to the ancestor script for both men:

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

See also [ancestor](#), [new\(\)](#)

cancelIdleLoad

Syntax `cancelIdleLoad loadTag`

Description Command; cancels the loading of all cast members that have the specified load tag.

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

```
cancelIdleLoad 20
```

See also [idleLoadTag](#)

case

Syntax	<pre>case <i>expression</i> of <i>expression1</i> : <i>Statement</i> <i>expression2</i> : <i>multipleStatements</i> . . . <i>expression3, expression4</i> : <i>Statement</i> {otherwise: <i>statement(s)</i>} end case</pre>
Description	<p>Keyword; starts a multiple branching logic structure that is easier to write than repeated if...then statements.</p> <p>Lingo compares the value in <i>case expression</i> to the expressions in the lines beneath it, starting at the beginning and continuing through each line in order, until Lingo encounters an expression that matches <i>case expression</i>.</p> <p>When Lingo finds a matching expression, it executes the corresponding statement or statements that follow the colon after the matching expression. When only one statement follows the matching expression, the matching expression and its corresponding statement may appear on the same line. Multiple statements must appear on indented lines immediately below the matching expression.</p> <p>When more than one possible match could cause Lingo to execute the same statements, the expressions must be separated by commas. (The syntax line containing <i>expression3</i> and <i>expression4</i> is an example of such a situation.)</p> <p>After Lingo encounters the first match, it stops testing for additional matches.</p> <p>If the optional otherwise statement is included at the end of the case structure, the statements following otherwise are executed if there are no matches.</p> <p>If a case statement tests cases that aren't all integer constants, the Export Xtra for Java converts the case statement to an if...then statement.</p>

Example The following handler tests which key the user pressed most recently and responds accordingly.

- If the user pressed A, the movie goes to the frame labeled Apple.
- If the user pressed B or C, the movie performs the specified transition and then goes to the frame labeled Oranges.
- If the user pressed any other key, the computer beeps.

```
on keyDown
  case (the key) of
    "A": go to frame "Apple"
    "B", "C":
      puppetTransition 99
      go to frame "Oranges"
    otherwise beep
  end case
end keyDown
```

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

```
case the rollOver of
  1: puppetSound "Horn"
  2: puppetSound "Drum"
  3: puppetSound "Bongos"
end case
```

castLib

Syntax castLib *whichCast*

Description Keyword; indicates that the cast specified in *whichCast* is a cast.

The default cast is cast number 1. To specify a cast member in a cast other than cast 1, set castLib to specify the alternative cast.

Example This statement displays the number of the Buttons cast in the Message window.

Dot syntax:

```
put castLib("Buttons").number
```

Verbose syntax:

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

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

```
sprite(10).member = member(5, 4)
```

castLibNum

Syntax

member(*whichCastMember*).castLibNum
the castLibNum of member *whichCastMember*
sprite(*whichSprite*).castLibNum
the castLibNum of sprite *whichSprite*

Description

Cast member and sprite property; determines the number of the cast that contains the specified cast member, or which castLib is currently associated with the specified sprite.

If you change the castLibNum sprite property without changing the memberNum sprite property, Director uses the cast member that has the same cast member number in the new cast. This is useful for movies that you use as templates and update by supplying new casts. If you organize the cast contents so that each cast member has a cast member number that corresponds to its role in the movie, Director automatically inserts the new cast members correctly. To change the cast member assigned to a sprite regardless of its cast, set the member sprite property.

For a castmember, this property can be tested but not set. It can be both tested and set for a sprite.

Example

This statement determines the number of the cast to which cast member Jazz is assigned.

Dot syntax:

```
put member("Jazz").castLibNum
```

Verbose syntax:

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

Example

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

Dot syntax:

```
sprite(5).castLibNum = castLib("Wednesday Schedule").number
```

Verbose syntax:

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

castMemberList

Syntax

`member(whichCursorCastMember).castmemberList`

the castmemberList of member *whichCursorCastMember*

Description

Cursor cast member property; specifies a list of cast members that make up the frames of a cursor. For *whichCursorCastMember*, substitute a cast member name (within quotation marks) or a cast member number. You can also specify cast members from different casts.

The first cast member in the list is the first frame of the cursor, the second cast member is the second frame, and so on.

If you specify cast members that are invalid for use in a cursor, they will be ignored, and the remaining cast members will be used.

This property can be tested and set.

Example

This command sets a series of four cast members for the animated color cursor cast member named *myCursor*.

Dot syntax:

```
member("myCursor").castmemberList = \  
[member 1, member 2, member 1 of castlib 2, member 2 of castlib 2]
```

Verbose syntax:

```
set the castmemberList of member "myCursor" = \  
[member 1, member 2, member 1 of castlib 2, member 2 of castlib 2]
```

center

Syntax

`member(whichCastMember).center`

the center of member *whichCastMember*

Description

Cast member property; interacts with the crop cast member property.

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

This property can be tested and set.

Example	This statement causes the digital video cast member Interview to be displayed in the top left corner of the sprite.
	Dot syntax:
	<pre>member("Interview").center = FALSE</pre>

Verbose syntax:

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

See also

[crop](#) (cast member property), [centerRegPoint](#), [regPoint](#), [scale](#)

centerRegPoint

Syntax	<pre>member(<i>whichCastMember</i>).centerRegPoint</pre> the centerRegPoint of member <i>whichCastMember</i>
Description	Flash, vector shape, and bitmap cast member property; automatically centers the registration point of the cast member when you resize the sprite (TRUE, default); or repositions the registration point at its current point value when you resize the sprite, set the defaultRect property, or set the regPoint property (FALSE). This property can be tested and set.
Example	This script checks to see if a Flash movie's centerRegPoint property is set to TRUE. If it is, the script uses the regPoint property to reposition the sprite's registration point to its upper left corner. By checking the centerRegPoint property, the script ensures that it does not reposition a registration point that had been previously set using the regPoint property. Dot syntax: <pre>on beginSprite me if sprite(the spriteNum of me).member.centerRegPoint = TRUE then sprite(the spriteNum of me).member.regPoint = point(0,0) end if end"</pre> Verbose syntax: <pre>on beginSprite me if the centerRegPoint of member the memberNum of me = TRUE then set the regPoint of member the memberNum of me = point(0,0) end if end</pre>

See also

[regPoint](#)

centerStage

Syntax	the centerStage
Description	Movie property; determines whether the Stage is centered on the monitor when the movie is loaded (TRUE, default) or not centered (FALSE). Place the statement that includes this property in the movie that precedes the movie you want it to affect. This property is useful for checking the Stage location before a movie plays from a projector.
	This property can be tested and set.
	Note: Be aware that behavior while playing back in a projector differs between Windows and Macintosh systems. Settings selected during creation of the projector may override this property.
Example	This statement sends the movie to a specific frame if the Stage is not centered: if the centerStage = FALSE then go to frame "Off Center"
Example	This statement changes the centerStage property to the opposite of its current value: set the centerStage to (not the centerStage)
See also	fixStageSize

changeArea

Syntax	member(<i>whichCastMember</i>).changeArea the changeArea of member <i>whichCastMember</i>
Description	Transition cast member property; determines whether a transition applies only to the changing area on the Stage (TRUE) or to the entire Stage (FALSE). Its effect is similar to selecting the Changing Area Only option in the Frame Properties Transition dialog box.
	This property can be tested and set.
Example	This statement makes the transition cast member Wave apply only to the changing area on the Stage. Dot syntax: member("Wave").changeArea = TRUE
	Verbose syntax: set the changeArea of member "Wave" to TRUE

channelCount

Syntax

`member(whichCastMember).channelCount`

the channelCount of member *whichCastMember*

`sound(channelNum).channelCount`

Description

Sound channel and cast member property; for sound channels, determines the number of channels in the currently playing or paused sound in the given sound channel. For sound cast members, determines the number of channels in the specified cast member.

This is useful for determining whether a sound is in monaural or in stereo. This property can be tested but not set.

Example

This statement determines the number of channels in the sound cast member, Jazz.

Dot syntax:

```
put member("Jazz").channelCount
```

Verbose syntax:

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

Example

This statement determines the number of channels in the sound member currently playing in sound channel 2:

```
put sound(2).channelCount
```

char...of

Syntax

`textMemberExpression.char[whichCharacter]`

char *whichCharacter* of *fieldOrStringVariable*

`textMemberExpression.char[firstCharacter..lastCharacter]`

char *firstCharacter* to *lastCharacter* of *fieldOrStringVariable*

Description

Keyword; identifies a character or a range of characters in a chunk expression. A chunk expression is any character, word, item, or line in any source of text (such as field cast members and variables) that holds a string.

- An expression using *whichCharacter* identifies a specific character.
- An expression using *firstCharacter* and *lastCharacter* identifies a range of characters.

The expressions must be integers that specify a character or range of characters in the chunk. Characters include letters, numbers, punctuation marks, spaces, and control characters such as Tab and Return.

You can test but not set the char...of keyword. Use the put...into command to modify the characters in a string.

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

```
put ("$9.00").char[1..1]
-- "$"
```

Example This statement displays the entire string \$9.00:

```
put ("$9.00").char[1..5]
-- "$9.00"
```

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

```
member("quiz").line[3].word[2].char[1..5] = "?????"
```

See also [mouseMember](#), [mouseItem](#), [mouseLine](#), [mouseWord](#)

characterSet

Syntax `member(whichFontMember).characterSet`

the characterSet of member *whichFontMember*

Description Font cast member property; returns a string containing the characters included for import when the cast member was created. If all characters in the original font were included, the result is an empty string.

Example This statement displays the characters included when cast member 11 was created. The characters included during import were numerals and Roman characters.

```
put member(11).characterSet
-- "1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
```

See also [recordFont](#), [bitmapSizes](#), [originalFont](#)

charPosToLoc()

Syntax `member(whichCastMember).charPosToLoc(nthCharacter)`

`charPosToLoc(member whichCastMember, nthCharacter)`

Description Field function; returns the point in the entire field cast member (not just the part that appears on the Stage) that is closest to the character specified by *nthCharacter*. This is useful for determining the location of individual characters.

Values for `charPosToLoc` are in pixels from the top left corner of the field cast member. The *nthCharacter* parameter is 1 for the first character in the field, 2 for the second character, and so on.

Example The following statement determines the point where the fiftieth character in the field cast member Headline appears and assigns the result to the variable location:

```
location = charPosToLoc(member "Headline", 50)
```

chars()

Syntax

`chars(stringExpression, firstCharacter, lastCharacter)`

Description

Function; identifies a substring of characters in *stringExpression*. The substring starts at *firstCharacter* and ends at *lastCharacter*. The expressions *firstCharacter* and *lastCharacter* must specify a position in the string.

If *firstCharacter* and *lastCharacter* are equal, then a single character is returned from the string. If *lastCharacter* is greater than the string length, only a substring up to the length of the string is identified. If *lastCharacter* is before *firstCharacter*, the function returns the value EMPTY.

To see an example of `chars()` used in a completed movie, see the Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement identifies the sixth character in the word *Macromedia*:

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

Example

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

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

Example

This statement tries to identify the sixth through twentieth characters of the word *Macromedia*. Because the word has only 10 characters, the result includes only the sixth through tenth characters.

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

See also

`char...of`; `length()` and `offset()` (string function); `number (characters)`

charSpacing

Syntax

`chunkExpression.charSpacing`

Description

Text cast member property; enables specifying any additional spacing applied to each letter in the *chunkExpression* portion of the text cast member.

A value less than 0 indicates less spacing between letters. A value greater than 0 indicates more spacing between letters.

The default value is 0, which results in default spacing between letters.

Example

The following handler increases the current character spacing of the third through fifth words within the text cast member *myCaption* by a value of 2:

```
on myCharSpacer  
    mySpaceValue = member("myCaption").word[3..5].charSpacing  
    member("myCaption").word[3..5].charSpacing = (mySpaceValue + 2)  
end
```

charToNum()

Syntax `(stringExpression).charToNum`

`charToNum(stringExpression)`

Description Function; returns the ASCII code that corresponds to the first character of *stringExpression*.

The `charToNum()` function is especially useful for testing the ASCII value of characters created by combining keys, such as the Control key and another alphanumeric key.

Director treats uppercase and lowercase letters the same if you compare them using the equal sign (=) operator; for example, the statement `put ("M" = "m")` returns the result 1 or TRUE.

Avoid problems by using `charToNum()` to return the ASCII code for a character and then use the ASCII code to refer to the character.

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

```
put ("A").charToNum  
-- 65
```

Example The following comparison determines whether the letter entered is a capital A, and then navigates to either a correct sequence or incorrect sequence in the Score:

```
on CheckKeyHit theKey  
    if (theKey).charToNum = 65 then  
        go "Correct Answer"  
    else  
        go "Wrong Answer"  
    end if  
end
```

See also [numToChar\(\)](#)

checkBoxAccess

Syntax	the checkBoxAccess
Description	Movie property; specifies one of three possible results when the user clicks a check box or radio button created with button tools in the Tools window:
	<ul style="list-style-type: none">• 0 (default)—Lets the user set check boxes and radio buttons on and off.• 1—Lets the user set check boxes and radio buttons on but not off.• 2—Prevents the user from setting check boxes and radio buttons at all; the buttons can be set only by scripts.
	This property can be tested and set.
Example	This statement sets the checkBoxAccess property to 1, which lets the user click check boxes and radio buttons on but not off: the checkBoxAccess to 1
Example	This statement records the current setting of the checkBoxAccess property by putting the value in the variable oldAccess: oldAccess to the checkBoxAccess
See also	hilite (cast member property), checkBoxType

checkBoxType

Syntax	the checkBoxType
Description	Movie property; specifies one of three ways to indicate whether a check box is selected:
	<ul style="list-style-type: none">• 0 (default)—Creates a standard check box that contains an X when the check box is selected.• 1—Creates a check box that contains a black rectangle when the check box is selected.• 2—Creates a check box that contains a filled black rectangle when the check box is selected.
	This property can be tested and set.
Example	This statement sets the checkBoxType property to 1, which creates a black rectangle in check boxes when the user clicks them: the checkBoxType to 1
See also	hilite (cast member property), checkBoxAccess

checkMark

Syntax

the checkMark of menuItem *whichItem* of menu *whichMenu*

Description

Menu item property; determines whether a check mark appears next to the custom menu item (TRUE) or not (FALSE, default).

The *whichItem* value can be either a menu item name or a menu item number. The *whichMenu* value can be either a menu name or a menu number.

This property can be tested and set.

Note: Menus are not available in Shockwave

Example

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

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

See also

`installMenu`; `enabled`, `name` (menu item property), `number` (menu items), `and` script; `name` (menu item property) and `number` (menu items); `menu`

chunkSize

Syntax

`member(whichCastMember).chunkSize`

the `chunkSize` of member *whichCastMember*

Description

Transition cast member property; determines the transition's chunk size in pixels from 1 to 128 and is equivalent to setting the smoothness slider in the Frame Properties: Transition dialog box. The smaller the chunk size, the smoother the transition appears.

This property can be tested and set.

Example

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

Dot syntax:

```
member("Fog").chunkSize = 4
```

Verbose syntax:

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

clearCache

Syntax clearCache

Description Command; clears Director's network cache.

The clearCache command clears only Director's cache, which is separate from the browser's cache.

If a file is in use, it remains in the cache until it is no longer in use.

Example This handler clears the cache when the movie starts:

```
on startMovie  
    clearCache  
end
```

See also cacheDocVerify(), cacheSize()

clearError

Syntax member(*whichFlashMember*).clearError()

clearError (*member whichFlashMember*)

Description Flash command; resets the error state of a streaming Flash cast member to 0.

When an error occurs while a cast member is streaming into memory, Director sets the cast member's state property to -1 to indicate that an error occurred.

When this happens, you can use the getError function to determine what type of error occurred and then use the clearError command to reset the cast member's error state to 0. After you clear the member's error state, Director tries to open the cast member if it is needed again in the Director movie. Setting a cast member's pathName, linked, and preload properties also automatically clears the error condition.

Example This handler checks to see if an out-of-memory error occurred for a Flash cast member named Dali, which was streaming into memory. If a memory error occurred, the script uses the unloadCast command to try to free some memory; it then branches the playback head to a frame in the Director movie named Artists, where the Flash movie sprite first appears, so Director can again try to play the Flash movie. If something other than an out-of-memory error occurred, the script goes to a frame named Sorry, which explains that the requested Flash movie can't be played.

```
on CheckFlashStatus
    if member("Dali").getError() = #memory then
        member("Dali").clearError()
        unloadCast
        go to frame "Artists"
    else
        go to frame "Sorry"
    end if
end
```

See also state, getError()

clearFrame

Syntax clearFrame

Description Command; erases everything in the current frame's sprite and affects channels during Score recording only.

Example The following handler clears the content of each frame before it edits that frame during Score generation:

```
on newScore
    beginRecording
    repeat with counter = 1 to 50
        clearFrame
        the frameScript to 25
        updateFrame
    end repeat
    endRecording
end
```

See also beginRecording and endRecording; updateFrame

clearGlobals

Syntax clearGlobals

Description Command; sets all global variables to VOID.

This command can be useful when initializing global variables or when opening a new movie that requires a new set of global variables.

Example This statement sets all global variables to VOID:

```
clearGlobals
```

clickLoc

Syntax the clickLoc

Description Function; identifies as a point the last place on the screen where the mouse was clicked.

Example The following on mouseDown handler displays the last mouse click location:

```
on mouseDown  
    put the clickLoc  
end mouseDown
```

If the click were 50 pixels from the left end of the Stage and 100 pixels from the top of the Stage, the Message window would display the following:

```
-- point(50, 100)
```

clickMode

Syntax sprite(*whichFlashSprite*).clickMode

the clickMode of sprite *whichFlashSprite*

member(*whichFlashMember*).clickMode

the clickMode of member *whichFlashMember*

Description Flash cast member and sprite property; controls when the Flash movie sprite detects mouse click events (mouseUp and mouseDown) and when it detects rollovers (mouseEnter, mouseWithin, and mouseLeave). The clickMode property can have these values:

- #boundingBox—Detects mouse click events anywhere within the sprite's bounding rectangle and detects rollovers at the sprite's boundaries.
- #opaque (default)—Detects mouse click events only when the pointer is over an opaque portion of the sprite and detects rollovers at the boundaries of the opaque portions of the sprite if the sprite's ink effect is set to Background Transparent. If the sprite's ink effect is not set to Background Transparent, this setting has the same effect as #boundingBox.
- #object—Detects mouse click events when the mouse pointer is over any filled (nonbackground) area of the sprite and detects rollovers at the boundaries of any filled area. This setting works regardless of the sprite's ink effect.

This property can be tested and set.

Example This script checks to see if the sprite, which is specified with an ink effect of Background Transparent, is currently set to be rendered direct to Stage. If the sprite is not rendered direct to Stage, the sprite's clickMode is set to #opaque. Otherwise (because ink effects are ignored for Flash movie sprites that are rendered direct to Stage), the sprite's clickMode is set to #boundingBox.

Dot syntax:

```
on beginSprite me
    if sprite(the spriteNum of me).directToStage = FALSE then
        sprite(the spriteNum of me).clickMode = #opaque
    else
        sprite(the spriteNum of me).clickMode = #boundingBox
    end if
end
```

Verbose syntax:

```
on beginSprite me
    if the directToStage of sprite the spriteNum of me = FALSE then
        set the clickMode of sprite the spriteNum of me = #opaque
    else
        set the clickMode of sprite the spriteNum of me = #boundingBox
    end if
end
```

clickOn

Syntax the clickOn

Description Function; returns the last active sprite clicked by the user. An active sprite is a sprite that has a sprite or cast member script associated with it.

When the user clicks the Stage, clickOn returns 0. To detect whether the user clicks a sprite with no script, you must assign a placeholder script to it ("--," for example) so that it can be detected by the clickOn function.

The clickOn can be checked within a repeat loop. However, neither clickOn nor clickLoc functions change value when the handler is running. The value that you obtain is the value from before the handler started.

Example This statement checks whether sprite 7 was the last active sprite clicked:

```
if the clickOn = 7 then alert "Sorry, try again."
```

Example This statement sets the foreColor property of the last active sprite that was clicked to a random color:

```
sprite(the clickOn).foreColor = random(255)-1
```

See also doubleClick, the mouseDown (system property), mouseMember, the mouseUp (system property)

closed

Syntax member(*whichCastMember*).closed

Description Vector shape cast member property; indicates whether the end points of a path are closed or open.

Vector shapes must be closed in order to contain a fill.

The value can be:

- TRUE—the end points are closed.
- FALSE—the end points are open.

close window

Syntax window(*windowIdentifier*).close()

close window *windowIdentifier*

Description Window command; closes the window specified by *windowIdentifier*.

- To specify a window by name, use the syntax close window *name*, where you replace *name* with the name of a window. Use the complete pathname.
- To specify a window by its number in *windowList*, use the syntax close window *number*, where you replace *number* with the window's number in *windowList*.

Closing a window that is already closed has no effect.

Be aware that closing a window does not stop the movie in the window nor clear it from memory. This command simply closes the window in which the movie is playing. You can reopen it quickly by using the open window command. This allows rapid access to windows that you want to keep available.

If you want to completely dispose of a window and clear it from memory, use the forget window command. Make sure that nothing refers to the movie in that window if you use the forget window command, or you will generate errors when scripts try to communicate or interact with the forgotten window.

Example This statement closes the window named Panel, which is in the subfolder MIAW Sources within the current movie's folder:

```
window("@/MIAW Sources/Panel").close()
```

Example This statement closes the window that is number 5 in *windowList*:

```
window(5).close()
```

See also forget window and open window; *windowList*

on closeWindow

Syntax

```
on closeWindow  
    statement(s)  
end
```

Description System message and event handler; contains statements that run when the user closes the window for a movie by clicking the window's close box.

The on closeWindow handler is a good place to put Lingo commands that you want executed every time the movie's window closes.

Example This handler tells Director to forget the current window when the user closes the window that the movie is playing in:

```
on closeWindow  
    -- perform general housekeeping here  
    forget the activeWindow  
end
```

closeXlib

Syntax

```
closeXlib whichFile
```

Description Command; closes the Xlibrary file specified by the string *whichFile*. If the Xlibrary file is in a folder other than that for the current movie, *whichFile* must specify a pathname. If no file is specified, all open Xlibraries are closed.

Xtras are stored in Xlibrary files. Xlibrary files are resource files that contain Xtras. HyperCard XCMDs and XFCNs can also be stored in Xlibrary files.

The closeXlib command doesn't work for URLs.

In Windows, using the DLL extension for Xtras is optional.

It is good practice to close any file you have opened as soon as you have finished using it.

Note: This command is not supported in Shockwave.

Example This statement closes all open Xlibrary files:

```
closeXlib
```

Example This statement closes the Xlibrary Video Disc Xlibrary when it is in the same folder as the movie:

```
closeXlib "Video Disc Xlibrary"
```

Example This statement closes the Xlibrary Transporter Xtras in the folder New Xtras, which is in the same folder as the movie. The disk is identified by the variable *currentDrive*:

```
closeXlib "@:New Xtras:Transporter Xtras"
```

See also [interface\(\)](#), [openXlib](#), [showXlib](#)

color()

Syntax

```
color(#rgb, redValue, greenValue, blueValue)  
color(#paletteIndex, paletteIndexNumber)  
rgb(rgbHexString)  
rgb(redValue, greenValue, blueValue)  
paletteIndex(paletteIndexNumber)
```

Description

Function and data type; determines an object's color as either RGB or 8-bit palette index values. These are the same values as those used in the color member and color sprite properties, the bgColor member and bgColor sprite properties, and the bgColor Stage property.

The color function allows for either 24-bit or 8-bit color values to be manipulated as well as applied to cast members, sprites, and the Stage.

For RGB values, each color component has a range from 0 to 255, and all other values are truncated. For paletteIndex types, an integer from 0 to 255 is used to indicate the index number in the current palette, and all other values are truncated.

Example

This statement performs a math operation:

```
palColorObj = paletteIndex(20)  
put palColorObj  
-- paletteIndex(20)  
put palColorObj / 2  
-- paletteIndex(10)
```

Example

This statement converts one color type to another type:

```
newColorObj = color(#rgb, 155, 0, 75)  
put newColorObj  
-- rgb(155, 0, 75)  
newColorObj.colorType = #paletteIndex  
put newColorObj  
-- paletteIndex(106)
```

Example

This statement obtains the hexadecimal representation of a color regardless of its type:

```
someColorObj = color(#paletteIndex, 32)  
put someColorObj.hexString()  
-- "#FF0099"
```

Example This statement determines individual RGB components and the paletteIndex value of a color regardless of its type:

```
newColorObj = color(#rgb, 155, 0, 75)
put newColorObj.green
-- 0
put newColorObj.paletteIndex
-- 106
newColorObj.green = 100
put newColorObj.paletteIndex
-- 94
put newColorObj
-- rgb(155, 100, 75)
newColorObj.paletteIndex = 45
put newColorObj
-- paletteIndex(45)
```

Example This statement changes the color of the fourth through the seventh characters of text member myQuotes:

```
member("myQuotes").char[4..7].color = rgb(200, 150, 75)
```

Example This Lingo displays the color of sprite 6 in the Message window, and then sets the color of sprite 6 to a new RGB value:

```
"put sprite(6).color
-- rgb( 255, 204, 102 )
sprite(6).color = rgb(122, 98, 210)"
```

Note: Setting the paletteIndex value of an RGB color type changes colorType to paletteIndex. Setting the RGB color type of a paletteIndex color sets its colorType value to RGB.

See also [color\(\)](#), [bgColor](#)

color (sprite property)

Syntax `sprite(whichSpriteNumber).color`

the color of sprite *whichSpriteNumber*

Description Sprite property; determines the foreground color of the sprite specified by *whichSprite*. Setting the foreColor sprite property is equivalent to choosing the foreground color from the Tools window when the sprite is selected on the Stage.

This property has the equivalent functionality of the foreColor sprite property, but the color value returned is a color object of whatever type has been set for that sprite.

This property can be tested and set.

See also [color\(\)](#), [bgColor](#), [foreColor](#)

colorDepth

Syntax the colorDepth

Description System property; determines the color depth of the computer's monitor.

- In Windows, using this property lets you check and set the monitor's color depth. Some video card and driver combinations may not enable you to set the colorDepth property. Always verify that the color depth has actually changed after you attempt to set it.
- On the Macintosh, this property lets you check the color depth of different monitors and change it when appropriate.

Possible values are the following:

1	Black and white
2	4 colors
4	16 colors
8	256 colors
16	32,768 or 65,536 colors
32	16,777,216 colors

If you try to set a monitor's color depth to a value that monitor does not support, the monitor's color depth doesn't change.

On computers with more than one monitor, the colorDepth property refers to the monitor displaying the Stage. If the Stage spans more than one monitor, the colorDepth property indicates the greatest depth of those monitors; colorDepth tries to set all those monitors to the specified depth.

This property can be tested and set.

Example This statement tells Director to play the segment Full color only if the monitor color depth is set to 256 colors:

```
if the colorDepth = 8 then play movie "Full color"
```

Example The following handler tries to change the color depth, and if it can't, it displays an alert:

```
on TryToSetColorDepth desiredDepth
    the colorDepth = desiredDepth
    if the colorDepth = desiredDepth then
        return true
    else
        alert "Please change your system to" && desiredDepth &&"color depth and reboot."
        return false
    end if
end
```

When changing the user's monitor color depth settings, it is good practice to restore the original depth when the movie has finished. In Windows, the command set the colorDepth = 0 restores the user's preferred settings from the control panel.

See also [switchColorDepth](#)

commandDown

Syntax the commandDown

Description Function; determines whether the Control key (Windows) or the Command key (Macintosh) is being pressed (TRUE) or not (FALSE).

You can use commandDown together with the element the key to determine when the Control or Command key is pressed in combination with another key. This lets you create handlers that are executed when the user presses specified Control or Command key combinations.

Control or Command key equivalents for Director's authoring menus take precedence while the movie is playing, unless you have installed custom Lingo menus or are playing a projector version of the movie.

For a movie playing back with the Director player for Java, this function returns TRUE only if a second key is pressed simultaneously with the Control or Command key. If the Control or Command key is pressed by itself, commandDown returns FALSE. This is because the browser receives the keys before the movie and thus responds to and intercepts any key combinations that are also browser keyboard shortcuts. For example, if the user presses Control+R or Command+R, the browser reloads the current page; the movie never receives the key combination.

For a demonstration of modifier keys in Lingo, see the sample movie "Keyboard Lingo" in Director Help.

Example These statements pause a projector whenever the user presses Control+ or Command+A. By setting the keyDownScript property to doCommandKey, the on prepareMovie handler makes the doCommandKey handler the first event handler executed when a key is pressed. The doCommandKey handler checks whether the Control+A or Command+A keys are pressed at the same time and pauses the movie if they are.

```
on prepareMovie
    the keyDownScript = "doCommandKey"
end

on doCommandKey
    if (the commandDown) and (the key = "a") then go to the frame
end
```

See also [controlDown](#), [key\(\)](#), [keyCode\(\)](#), [optionDown](#), [shiftDown](#)

comments

Syntax	<i>member.comments</i>
	the comments of <i>member</i>
Description	This cast member property provides a place to store any comments you want to maintain about the given cast member, or any other strings you want to associate with the member. This property can be tested and set. It can also be set in the Property Inspector's Member tab.
Example	This statement sets the comments of the member Backdrop to the string "Still need to license this artwork": <code>member("Backdrop").comments = "Still need to license this artwork"</code>
See also	<code>creationDate</code> , <code>modifiedBy</code> , <code>modifiedDate</code>

constrainH()

Syntax	<code>constrainH (whichSprite, integerExpression)</code>
Description	Function; evaluates <i>integerExpression</i> and then returns a value that depends on the horizontal coordinates of the left and right edges of <i>whichSprite</i> , as follows: <ul style="list-style-type: none">• When the value is between the left and right coordinates, the value doesn't change.• When the value is less than the left horizontal coordinate, the value changes to the value of the left coordinate.• When the value is greater than the right horizontal coordinate, the value changes to the value of the right coordinate. The constrainH and constrainV functions constrain only one axis each; the constraint sprite property limits both. Note that this function does not change the sprite's properties.
Example	These statements check the constrainH function for sprite 1 when it has left and right coordinates of 40 and 60: <code>put constrainH(1, 20) -- 40</code> <code>put constrainH(1, 55) -- 55</code> <code>put constrainH(1, 100) -- 60</code>
Example	This statement constrains a moveable slider (sprite 1) to the edges of a gauge (sprite 2) when the mouse pointer goes past the edge of the gauge: <code>set the locH of sprite 1 to constrainH(2, the mouseH)</code>
See also	<code>constrainV()</code> , <code>constraint</code> , <code>left</code> , <code>right</code>

constraint

Syntax `sprite(whichSprite).constraint`

the constraint of sprite *whichSprite*

Description Sprite property; determines whether the registration point of the sprite specified by *whichSprite* is constrained to the bounding rectangle of another sprite (1 or TRUE) or not (0 or FALSE, default).

The constraint sprite property is useful for constraining a moveable sprite to the bounding rectangle of another sprite to simulate a track for a slider control or to restrict where on the screen a user can drag an object in a game.

The constraint sprite property affects moveable sprites and the `locH` and `locV` sprite properties. The constraint point of a moveable sprite cannot be moved outside the bounding rectangle of the constraining sprite. (The constraint point for a bitmap sprite is the registration point. The constraint point for a shape sprite is its top left corner.) When a sprite has a constraint set, the constraint limits override any `locH` and `locV` sprite property settings.

This property can be tested and set.

Example This statement removes a constraint sprite property:

Dot syntax:

```
sprite(whichSprite).constraint = 0
```

Verbose syntax:

```
set the constraint of sprite whichSprite to 0
```

Example This statement constrains sprite (*i* + 1) to the boundary of sprite 14:

```
sprite(i + 1).constraint = 14
```

Example This statement checks whether sprite 3 is constrained and activates the handler `showConstraint` if it is: (The operator `<>` performs a not-equal-to operation.)

```
if sprite(3).constraint <> 0 then showConstraint
```

See also `constrainH()`, `constrainV()`; `locH`, `locV`

constrainV()

Syntax constrainV (*whichSprite*, *integerExpression*)

Description Function; evaluates *integerExpression* and then returns a value that depends on the vertical coordinates of the top and bottom edges of the sprite specified by *whichSprite*, as follows:

- When the value is between the top and bottom coordinates, the value doesn't change.
- When the value is less than the top coordinate, the value changes to the value of the top coordinate.
- When the value is greater than the bottom coordinate, the value changes to the value of the bottom coordinate.

This function does not change the sprite properties.

Example These statements check the constrainV function for sprite 1 when it has top and bottom coordinates of 40 and 60:

```
put constrainV(1, 20)
-- 40

put constrainV(1, 55)
-- 55

put constrainV(1, 100)
-- 60
```

Example This statement constrains a moveable slider (sprite 1) to the edges of a gauge (sprite 2) when the mouse pointer moves past the edge of the gauge:

```
set the locV of sprite 1 to constrainV(2, the mouseV)
```

See also bottom, constraint, top; constrainH()

contains

Syntax *stringExpression1* contains *stringExpression2*

Description Operator; compares two strings and determines whether *stringExpression1* contains *stringExpression2* (TRUE) or not (FALSE).

The contains comparison operator has a precedence level of 1.

The contains comparison operator is useful for checking whether the user types a specific character or string of characters. You can also use the contains operator to search one or more fields for specific strings of characters.

Example This example determines whether a character passed to it is a digit:

```
on isNumber aLetter
  digits = "1234567890"
  if digits contains aLetter then
    return TRUE
  else
    return FALSE
  end if
end
```

Note: The string comparison is not sensitive to case or diacritical marks; "a" and Å are treated the same.

See also offset() (string function), starts

continue

This is obsolete. Use go to the frame +1.

controlDown

Syntax the controlDown

Description Function; determines whether the Control key is being pressed (TRUE) or not (FALSE).

You can use the controlDown function together with the key to check for combinations of the Control key and another key.

For a movie playing back with the Director player for Java, this function returns TRUE only if a second key is pressed simultaneously with the Control key. If the Control key is pressed by itself, controlDown returns FALSE. The Director player for Java supports key combinations with the Control key. However, the browser receives the keys before the movie and thus responds to and intercepts any key combinations that are also browser keyboard shortcuts.

For a demonstration of modifier keys and Lingo, see the sample movie Keyboard Lingo in Director Help.

Example This on keyDown handler checks whether the pressed key is the Control key, and if it is, the handler activates the on doControlKey handler. The argument (the key) identifies which key was pressed in addition to the Control key.

```
on keyDown
  if (the controlDown)then doControlKey (the key)
end
```

See also charToNum(), commandDown, key(), keyCode(), optionDown, shiftDown

controller

Syntax `member(whichCastMember).controller`

the controller of member *whichCastMember*

Description Digital video cast member property; determines whether a digital video movie cast member shows or hides its controller. Setting this property to 1 shows the controller; setting it to 0 hides the controller.

The controller member property applies to a QuickTime digital video only.

- Setting the controller member property for a Video for Windows digital video performs no operation and generates no error message.
- Checking the controller member property for a Video for Windows digital video always returns FALSE.

The digital video must be in direct-to-stage playback mode to display the controller.

Example This statement causes the QuickTime cast member Demo to display its controller.

Dot syntax:

```
member("Demo").controller = 1
```

Verbose syntax:

```
set the controller of member "Demo" to 1
```

See also `directToStage`

copyPixels()

Syntax `imageObject.copyPixels(sourceImageObject, destinationRect, sourceRect {, parameterList})`

`imageObject.copyPixels(sourceImageObject, destinationQuad, sourceRect {, parameterList })`

Description This function copies the contents of the *sourceRect* from the *sourceImageObject* into the given *imageObject*. The pixels are copied from the *sourceRect* in the *sourceImageObject* and placed into the *destinationRect* or *destinationQuad* in the given *imageObject*. See quad for information on using quads.

You can include an optional property list of parameters in order to manipulate the pixels being copied before they are placed into the *destinationRect*. The property list may contain any or all of the following parameters:

Property	Use and Effect
#color	The foreground color to apply for colorization effects. The default color is black.
#bgColor	The background color to apply for colorization effects or background transparency. The default background color is white.
#ink	The type of ink to apply to the copied pixels. This can be an ink symbol or the corresponding numeric ink value. The default ink is #copy. See ink for the list of possible values.
#blendLevel	The degree of blend (transparency) to apply to the copied pixels. The range of values is from 0 to 255. The default value is 255 (opaque). Using a value less than 255 forces the #ink setting to be #blend, or #blendTransparent if it was originally #backgroundTransparent. You may also substitute #blend as the property name and use a value range of 0 to 100. See blend for more information.
#dither	A TRUE or FALSE value that determines whether the copied pixels will be dithered when placed into the <i>destinationRect</i> in 8- and 16-bit images. The default value is FALSE, which maps the copied pixels directly into the <i>imageObject</i> 's color palette.
#useFastQuads	A TRUE or FALSE value that determines whether quad calculations are made using Director's faster but less precise method when copying pixels into a <i>destinationQuad</i> . Set this to TRUE if you are using quads for simple rotation and skew operations. Leave it FALSE (the default value) for arbitrary quads, such as those used for perspective transformations. See useFastQuads.
#maskImage	Used to specified a mask or matte object created with the createMask() or createMatte() functions to be used as a mask for the pixels being copied. This allows you to duplicate the effects of mask and matte sprite inks. Note that if the source image has an alpha channel and its useAlpha property is TRUE, the alpha channel is used and the specified mask or matte is ignored. The default is no mask.
#maskOffset	A point indicating the amount of x and y offset to apply to the mask specified by #maskImage. The offset is relative to the upper left corner of the <i>sourceImage</i> . The default offset is (0, 0).

When copying pixels from one area of a cast member to another area of the same member, it is best to copy the pixels first into a duplicate image object before copying them back into the original member. Copying directly from one area to another in the same image is not recommended. See [duplicate\(\)](#).

To simulate matte ink with [copyPixels\(\)](#), create a matte object with [createMatte\(\)](#) and then pass that object as the #maskImage parameter with [copyPixels\(\)](#).

Copying pixels from an image object into itself is not recommended. Use separate image objects instead.

To see an example of quad used in a completed movie, see the Quad movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement copies the entire image of member Happy into the rectangle of member flower. If the members are different sizes, the image of member Happy will be resized to fit the rectangle of member flower.

```
member("flower").image.copyPixels(member("Happy").image, \
member("flower").rect, member("Happy").rect)
```

Example This statement copies part of the image of member Happy into part of member flower. The part of the image copied from Happy is within rectangle(0, 0, 200, 90). It is pasted into rectangle(20, 20, 100, 40) within the image of member flower. The copied portion of Happy is resized to fit the rectangle into which it is pasted.

```
member("flower").image.copyPixels(member("Happy").image,\ 
rect(20, 20, 100, 40), rect(0, 0, 200, 90))
```

Example This statement copies the entire image of member Happy into a rectangle within the image of member flower. The rectangle into which the copied image of member Happy is pasted is the same size as the rectangle of member Happy, so the copied image is not resized. The blend level of the copied image is 50, so it is semi-transparent, revealing the part of member flower it is pasted over.

```
member("flower").image.copyPixels(member("Happy").image,\ 
member("Happy").rect, member("Happy").rect, [#blendLevel: 50])
```

See also ink, color()

copyrightInfo

Syntax member(*whichCastMember*).copyrightInfo

copyrightInfo of member *whichCastMember*

Description Shockwave Audio (SWA) cast member property; displays the copyright text in a SWA file. This property is available only after the SWA sound begins playing or after the file has been preloaded using the preLoadBuffer command.

This property can be tested and set.

Example This statement tells Director to display the copyright information for the Shockwave Audio file SWAfile in a field cast member named Info Display.

Dot syntax:

```
set whatState = the state of member "SWAfile"
if whatState > 1 AND whatState < 9 then
    put member("SWAfile").copyrightInfo into member("Info Display")
end if
```

Verbose syntax:

```
set whatState = the state of member "SWAfile"  
if whatState > 1 AND whatState < 9 then  
    put the copyrightInfo of member "SWAfile" into member "Info Display"  
end if
```

copyToClipBoard

Syntax `member(whichCastMember).copyToClipBoard()`

`copyToClipBoard member whichCastMember`

Description Command; copies the specified cast member to the Clipboard without requiring that the cast window is active. You can use this command to copy cast members between movies or applications.

Example This statement copies the cast member named chair to the Clipboard:

`member("chair").copyToClipboard()`

Example This statement copies cast member number 5 to the Clipboard:

`member(5).copyToClipboard()`

See also [pasteClipBoardInto](#)

cos()

Syntax `(angle).cos`

`cos (angle)`

Description Function; calculates the cosine of the specified angle, which must be expressed in radians.

Example The following statement calculates the cosine of PI divided by 2 and displays it in the Message window:

`put (PI/2).cos`

See also [atan\(\)](#), [PI](#), [sin\(\)](#)

count()

Syntax

list.count
count (*list*)
count(*theObject*)
object.count
textExpression.count

Description

Function; returns the number of entries in a linear or property list, the number of properties in a parent script without counting the properties in an ancestor script, or the chunks of a text expression such as characters, lines, or words.

The count command works with linear and property lists, objects created with parent scripts, and the globals property.

To see an example of count() used in a completed movie, see the Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement displays the number 3, the number of entries:

```
put [10,20,30].count  
-- 3
```

See also

globals

cpuHogTicks

Syntax

the cpuHogTicks

Description

System property; determines how often Director releases control of the CPU to let the computer process background events, such as events in other applications, network events, clock updates, and other keyboard events.

The default value is 20 ticks. To give more time to Director before releasing the CPU to background events or to control how the computer responds to network operations, set cpuHogTicks to a higher value.

To create faster auto-repeating key performance but slower animation, set cpuHogTicks to a lower value. In a movie, when a user holds down a key to generate a rapid sequence of auto-repeating key presses, Director typically checks for auto-repeating key presses less frequently than the rate set in the computer's control panel.

The cpuHogTicks property works only on the Macintosh.

Example

This statement tells Director to release control of the CPU every 6 ticks, or every 0.10 of a second:

```
the cpuHogTicks = 6
```

See also

ticks

createMask()

Syntax

imageObject.createMask()

Description

This function creates and returns a mask object for use with the `copyPixels()` function.

Mask objects aren't image objects; they're useful only with the `copyPixels()` function for duplicating the effect of mask sprite ink. To save time, if you plan to use the same image as a mask more than once, it's best to create the mask object and save it in a variable for reuse.

Example

This statement copies the entire image of member Happy into a rectangle within the image of member brown square. Member gradient2 is used as a mask with the copied image. The mask is offset by 10 pixels up and to the left of the the rectangle into which the image of member Happy is pasted.

```
member("brown square").image.copyPixels(member("Happy").image, \
rect(20, 20, 150, 108), member("Happy").rect, \
[#maskImage:member("gradient2").image.createMask(), maskOffset:point(-10, -10)])
```

See also

`copyPixels()`, `createMatte()`, `ink`

createMatte()

Syntax

imageObject.createMatte({alphaThreshold})

Description

This function creates and returns a matte object that you can use with `copyPixels()` to duplicate the effect of the matte sprite ink. The matte object is created from the specified image object's alpha layer. The optional parameter `alphaThreshold` excludes from the matte all pixels whose alpha channel value is below that threshold. It is used only with 32-bit images that have an alpha channel. The `alphaThreshold` must be a value between 0 and 255.

Matte objects aren't image objects; they are useful only with the `copyPixels()` function. To save time, if you plan to use the same image as a matte more than once, it's best to create the matte and save it in a variable for reuse.

Example

This statement creates a new matte object from the alpha layer of the image object testImage and ignores pixels with alpha values below 50%:

```
newMatte = testImage.createMatte(128)
```

See also

`copyPixels()`, `createMask()`

creationDate

Syntax *member.creationDate*

the creationDate of *member*

Description This cast member property records the date and time that the cast member was first created, using the system time on the computer. You can use this property to schedule a project; Director doesn't use it for anything.

This property can be tested and set.

Example Although you typically inspect the creationDate property using the Property Inspector or the Cast window list view, you can check it in the Message window:

```
put member(1).creationDate  
-- date( 1999, 12, 8 )
```

See also comments, modifiedBy, modifiedDate

crop() (image object command)

Syntax *imageObject.crop(rectToCropTo)*

Description When used with an image object, returns a new image object that contains a copy of the given image object, cropped to the given rect. The original image object is unchanged. The new image object does not belong to any cast member and has no association with the Stage. If you wish to assign it to a cast member you can do so by setting the image property of that cast member.

This is different from using crop with a cast member, which crops the cast member itself, altering the original.

Example This Lingo takes a snapshot of the Stage and crops it to the rect of sprite 10, capturing the current appearance of that sprite on the Stage:

```
set stagelImage = (the stage).image  
set spritelImage = stagelImage.crop(sprite(10).rect)  
member("sprite snapshot").image = spritelImage
```

Example This statement uses the rectangle of cast member Happy to crop the image of cast member Flower, then sets the image of cast member Happy to the result.

```
member("Happy").image = member("Flower").image.crop(member("Happy").rect)
```

crop() (member command)

Syntax	<code>member(<i>whichMember</i>).crop(<i>rectToCropTo</i>)</code> crop member <i>whichMember</i> , <i>rectToCropTo</i>
Description	Bitmap command; allows a bitmap cast member to be cropped to a specific size. You can use crop to trim existing cast members, or in conjunction with the picture of the Stage to grab a snapshot and then crop it to size for display. The registration point is kept in the same location so the bitmap does not move in relation to the original position.
Example	This statement sets an existing bitmap member to a snapshot of the Stage then crops the resulting image to a rectangle equal to sprite 10: <code>member("stage image").picture = (the stage).picture member("stage image").crop(sprite(10).rect)</code>
See also	picture (cast member property)

crop (cast member property)

Syntax	<code>member(<i>whichCastMember</i>).crop</code> the crop of member <i>whichCastMember</i>
Description	Cast member property; scales a digital video cast member to fit exactly inside the sprite rectangle in which it appears (FALSE), or it crops but doesn't scale the cast member to fit inside the sprite rectangle (TRUE). This property can be tested and set.
Example	This statement instructs Lingo to crop any sprite that refers to the digital video cast member Interview. Dot syntax: <code>member("Interview").crop = TRUE</code> Verbose syntax: <code>set the crop of member "Interview" to TRUE</code>
See also	center

on cuePassed

Syntax on cuePassed(*channelID*, *cuePointNumber*,*cuePointName*)

statement(s)

 end

 on cuePassed(*me*,*channelID*, *cuePointNumber*,*cuePointName*)

statement(s)

 end

Description System message and event handler; contains statements that run each time a sound or sprite passes a cue point in its media.

- *me*—The optional *me* parameter is the *scriptInstanceRef* value of the script being invoked. You must include this parameter when using the message in a behavior. If this parameter is omitted, the other arguments will not be processed correctly.
- *channelID*—The number of the sound or sprite channel for the file where the cue point occurred.
- *cuePointNumber*—The ordinal number of the cue point that triggers the event in the list of the cast member's cue points.
- *cuePointName*—The name of the cue point that was encountered.

The message is passed—in order—to sprite, cast member, frame, and movie scripts. For the sprite to receive the event, it must be the source of the sound, like a QuickTime movie or SWA cast member. Use the *isPastCuePoint* property to check cues in behaviors on sprites that don't generate sounds.

Example This handler placed in a Movie or Frame script reports any cue points in sound channel 1 to the Message window:

```
on cuePassed channel, number, name
    if (channel = #Sound1) then
        put "CuePoint" && number && "named" && name && "occurred in sound 1"
    end if
end
```

See also *scriptInstanceList*, *cuePointNames*, *cuePointTimes*, *isPastCuePoint()*

cuePointNames

Syntax	member(<i>whichCastMember</i>).cuePointNames the cuePointNames of member <i>whichCastMember</i>
Description	Cast member property; creates list of cue point names, or if a cue point is not named, inserts an empty string ("") as a placeholder in the list. Cue point names are useful for synchronizing sound, QuickTime, and animation. This property is supported by SoundEdit cast members, QuickTime digital video cast members, and Xtras cast members that contain cue points. Xtras that generate cue points at run time may not be able to list cue point names.
Example	This statement obtains the name of the third cue point of a cast member. Dot syntax: <pre>put member("symphony").cuePointNames[3]</pre> Verbose syntax: <pre>put (getAt(the cuePointNames of member "symphony",3))</pre>
See also	cuePointTimes , mostRecentCuePoint

cuePointTimes

Syntax	member(<i>whichCastMember</i>).cuePointTimes the cuePointTimes of member <i>whichCastMember</i>
Description	Cast member property; lists the times of the cue points, in milliseconds, for a given cast member. Cue point times are useful for synchronizing sound, QuickTime, and animation. This property is supported by SoundEdit cast members, QuickTime digital video cast members, and Xtras cast members that support cue points. Xtras that generate cue points at run time may not be able to list cue point names.
Example	This statement obtains the time of the third cue point for a sound cast member. Dot syntax: <pre>put member("symphony").cuePointTimes[3]</pre> Verbose syntax: <pre>put (getAt(the cuePointTimes of member "symphony",3))</pre>
See also	cuePointNames , mostRecentCuePoint

currentSpriteNum

Syntax the currentSpriteNum

Description Movie property; indicates the channel number of the sprite whose script is currently running. It is valid in behaviors and cast member scripts. When used in frame scripts or movie scripts, the currentSpriteNum property's value is 0.

The currentSpriteNum property is similar to spriteNum of me, but it doesn't require the me reference.

This property can be tested but not set.

Note: This property was more useful during transitions from older movies to Director 6, when behaviors were introduced. It allowed some behavior-like functionality without having to completely rewrite Lingo code. It is not necessary when authoring with behaviors and is therefore less useful than in the past.

Example The following handler in a cast member or movie script switches the cast member assigned to the sprite involved in the mouseDown event:

```
on mouseDown
    sprite(the currentSpriteNum).member = member "DownPict"
end
```

See also me, spriteNum

currentTime

Syntax sprite(*whichSprite*).currentTime

the currentTime of sprite *whichSprite*

sound(*channelNum*).currentTime

Description Sprite and sound channel property; returns the current playing time, in milliseconds, for a sound sprite, QuickTime digital video sprite, or any Xtra that supports cue points. For a sound channel, returns the current playing time of the sound member currently playing in the given sound channel.

This property can be tested and set. When this property is set, the range of allowable values is from zero to the duration of the member.

Shockwave Audio (SWA) sounds can appear as sprites in sprite channels, but they play sound in a sound channel. You should refer to SWA sound sprites by their sprite channel number rather than by a sound channel number.

Example This statement displays the current time, in seconds, of the sound sprite in sprite channel 10.

Dot syntax:

```
member("time").text = string(sprite (10).currentTime/ 1000)
```

Verbose syntax:

```
set the text of member "time" to (the currentTime of sprite 10) / 1000
```

Example This statement causes the sound playing in sound channel 2 to skip to the point 2.7 seconds from the beginning of the sound cast member:

```
sound(2).currentTime = 2700
```

See also movieTime, duration

cursor (command)

Syntax cursor [*castNumber*, *maskCastNumber*]

cursor *whichCursor*

cursor (member *whichCursorCastMember*)

Description Command; changes the cast member or built-in cursor that is used for a cursor and stays in effect until you turn it off by setting the cursor to 0.

- Use the syntax cursor [*castNumber*, *maskCastNumber*] to specify the number of a cast member to use as a cursor and its optional mask. The cursor's hot spot is the registration point of the cast member.

The cast member that you specify must be a 1-bit cast member. If the cast member is larger than 16 by 16 pixels, Director crops it to a 16-by-16-pixel square, starting in the upper left corner of the image. The cursor's hot spot is still the registration point of the cast member.

- Use the syntax cursor *whichCursor* to specify default system cursors. The term *whichCursor* must be one of the following integer values:

0*	No cursor set
-1	Arrow (pointer) cursor
1	I-beam cursor
2	Crosshair cursor
3*	Crossbar cursor
4	Watch cursor (Macintosh only)
200*	Blank cursor (hides cursor)

* The Director player for Java does not support these cursor types and displays an arrow cursor instead.

- Use the syntax `cursor (member whichCursorCastMember)` for the custom cursors available through the Cursor Xtra.

Be sure not to confuse the syntax `cursor 1` with `cursor [1]`. The first selects the I-beam from the system cursor set; the second uses cast member 1 as the custom cursor.

Note: Although the Cursor Xtra allows cursors of different cast types, text cast members cannot be used as cursors.

During system events such as file loading, the operating system may display the watch cursor and then change to the pointer cursor when returning control to the application, overriding the cursor command settings from the previous movie. To use `cursor` at the beginning of any new movie that is loaded in a presentation using a custom cursor for multiple movies, store any special cursor resource number as a global variable that remains in memory between movies.

Cursor commands can be interrupted by an Xtra or other external agent. If the cursor is set to a value in Director and an Xtra or external agent takes control of the cursor, resetting the cursor to the original value has no effect because Director doesn't perceive that the cursor has changed. To work around this, explicitly set the cursor to a third value and then reset it to the original value.

Example

This statement changes the cursor to a watch cursor on the Macintosh, and hourglass in Windows, whenever the value in the variable named `status` equals 1:

```
if status = 1 then cursor 4
```

This handler checks whether the cast member assigned to the variable is a 1-bit cast member and then uses it as the cursor if it is:

```
on myCursor someMember
  if the depth of member someMember = 1 then
    cursor[someMember]
  else
    beep
  end if
end
```

See also

`cursor (sprite property)`, `rollOver()`

cursor (sprite property)

Syntax

```
sprite(whichSprite).cursor = [castNumber, maskCastNumber]
```

set the cursor of sprite *whichSprite* to [*castNumber*, *maskCastNumber*]

```
sprite(whichSprite).cursor = whichCursor
```

set the cursor of sprite *whichSprite* to *whichCursor*

Description

Sprite property; determines the cursor used when the pointer is over the sprite specified by the integer expression *whichSprite*. This property stays in effect until you turn it off by setting the cursor to 0. Use the cursor sprite property to change the cursor when the mouse pointer is over specific regions of the screen and to indicate regions where certain actions are possible when the user clicks on them.

When you set the cursor sprite property in a given frame, Director keeps track of the sprite rectangle to determine whether to alter the cursor. This rectangle persists when the movie enters another frame unless you set the cursor sprite property for that channel to 0.

- Use the syntax `cursor of sprite...[castNumber, maskCastNumber]` to specify the number of a cast member to use as a cursor and its optional mask.
- Use the syntax `cursor of sprite...whichCursor` to specify default system cursors. The term *whichCursor* must be one of the following integer values:

0*	No cursor set
-1	Arrow (pointer) cursor
1	I-beam cursor
2	Crosshair cursor
3*	Crossbar cursor
4	Watch cursor (Macintosh only)
200*	Blank cursor (hides cursor)

* The Director player for Java does not support these cursor types and displays an arrow cursor instead.

To use custom cursors, set the cursor sprite property to a list containing the cast member to be used as a cursor or to the number that specifies a system cursor. In Windows, a cursor must be a cast member, not a resource; if a cursor is not available because it is a resource, Director displays the standard arrow cursor instead. For best results, don't use custom cursors when creating cross-platform movies.

If the sprite is a bitmap that has matte ink applied, the cursor changes only when the cursor is over the matte portion of the sprite.

When the cursor is over the location of a sprite that has been removed, rollover still occurs. Avoid this problem by not performing rollovers at these locations or by relocating the sprite up above the menu bar before deleting it.

On the Macintosh, you can use a numbered cursor resource in the current open movie file as the cursor by replacing *whichCursor* with the number of the cursor resource.

This property can be tested and set.

Example This statement changes the cursor that appears over sprite 20 to a watch cursor.

Dot syntax:

```
sprite(20).cursor = 4
```

Verbose syntax:

```
set the cursor of sprite 20 to 4
```

See also cursor (command)

cursorSize

Syntax member(*whichCursorCastMember*).cursorSize

the cursorSize of member *whichCursorCastMember*

Description Cursor cast member property; specifies the size of the animated color cursor cast member *whichCursorCastMember*.

Specify size: For cursors up to:

16 16 by 16 pixels

32 32 by 32 pixels

Bitmap cast members smaller than the specified size are displayed at full size, and larger ones are scaled proportionally to the specified size.

The default value is 32 for Windows and 16 for the Macintosh. If you set an invalid value, an error message appears when the movie runs (but not when you compile).

This property can be tested and set.

Example This command resizes the animated color cursor stored in cast member 20 to 32 by 32 pixels.

Dot syntax:

```
member(20).cursorSize = 32
```

Verbose syntax:

```
set the cursorSize of member 20 = 32
```

curve

Syntax `member.curve[curveListIndex]`

Description This property contains the vertexList of an individual curve (shape) from a vector shape cast member. You can use the curve property along with the vertex property to get individual vertices of a specific curve in a vector shape.

A vertexList is a list of vertices, and each vertex is a property list containing up to three properties: a #vertex property with the location of the vertex, a #handle1 property with the location of the first control point for that vertex, and a #handle2 property with the location of the second control point for that vertex. See vertexList.

Example This statement displays the list of vertices of the third curve of vector shape member SimpleCurves:

```
put member("SimpleCurves").curve[3]
-- [[#vertex: point(113.0000, 40.0000), #handle1: point(32.0000, 10.0000), #handle2:
point(-32.0000, -10.0000)], [#vertex: point(164.0000, 56.0000)]]
```

Example This statement moves the first vertex of the first curve in a vector shape down and to the right by 10 pixels:

```
member(1).curve[1].vertex[1] = member(1).curve[1].vertex[1] + point(10, 10)
```

Example The code below moves a sprite to the location of the first vertex of the first curve in a vector shape. The vector shape's originMode must be set to #topLeft for this to work.

```
vertexLoc = member(1).curve[1].vertex[1]
spriteLoc = mapMemberToStage(sprite(3), vertexLoc)
sprite(7).loc = spriteLoc
vertex, vertexList
```

date() (system clock)

Syntax the abbr date

the abbrev date

the abbreviated date

the date

the long date

the short date

Description Function; returns the current date in the system clock in one of three formats: abbreviated, long, or short (default). The abbreviated format can also be referred to as abbrev and abbr.

In Java, the date function is available, but it doesn't accept abbrev, long, or short modifiers. When the movie plays back as an applet, the date's format is MM/DD/YY, where MM represents the month, DD represents the day, and YY represents the last two digits of the current year. For the months January through September, the value for MM is a single digit.

The format Director uses for the date varies, depending on how the date is formatted on the computer.

- In Windows, you can customize the date display by using the International control panel. (Windows stores the current short date format in the System.ini file. Use this value to determine what the parts of the short date indicate.)
- On the Macintosh, you can customize the date display by using the Date and Time control panel.

Example This statement displays the abbreviated date:

```
put the abbreviated date  
-- "Sat, Sep 7, 1991"
```

Example This statement displays the long date:

```
put the long date  
-- "Saturday, September 7, 1991"
```

Example This statement displays the short date:

```
put the short date  
-- "9/7/91"
```

Example This statement tests whether the current date is January 1 by checking whether the first four characters of the date are 1/1. If it is January 1, the alert "Happy New Year!" appears:

```
if char 1 to 4 of the date = "1/1/" then alert "Happy New Year!"
```

Note: The three date formats vary, depending on the country for which your operating system was designed. These examples are for the United States. Use the date object to create and manipulate dates in a standard format.

See also [time\(\)](#), [date\(\) \(formats\)](#), [systemDate](#)

date() (formats)

Syntax

```
date(ISOFormatString)
date(ISOFormatInteger)
date(ISOFormatIntegerYear, ISOFormatIntegerMonth, ISOFormatIntegerDay)
```

Description

Function and data type; creates a standard, formatted date object instance for use with other date object instances in arithmetic operations and for use in manipulating dates across platforms and in international formats.

When creating the date, use four digits for the year, two digits for the month, and two digits for the day. The following expressions are equivalent:

integer:	set vacationStart = date(19980618)
string:	set vacationStart = date("19980618")
comma separated:	set vacationStart = date(1998, 06, 18)

Addition and subtraction operations on the date are interpreted as the addition and subtraction of days.

The individual properties of the date object instance returned are:

#year	Integer representing the year
#month	Integer representing the month of the year
#day	Integer representing the day of the month

Example

These statements create and determine the number of days between two dates:

```
myBirthDay = date(19650712)
yourBirthDay = date(19450529)
put "There are" && abs(yourBirthday - myBirthday) && "days between our birthdays."
```

Example

These statements access an individual property of a date:

```
myBirthDay = date(19650712)
put "I was born in month number"&&myBirthday.month
```

See also

[date\(\) \(system clock\)](#)

deactivateApplication

Syntax on deactivateApplication

Description Built-in handler; runs when the projector is sent to the background. This handler is useful when a projector runs in a window and the user can send it to the background to work with other applications. Any MIAWs running in the projector can also make use of this handler.

During authoring, this handler is called only if Animate in Background is turned on in General Preferences.

On Windows, this handler is not called if the projector is merely minimized and no other application is brought to the foreground.

Example This handler plays a sound each time the user sends the projector to the background:

```
on deactivateApplication
    sound(1).queue(member("closeSound"))
    sound(1).play()
end
```

See also activateApplication, activateWindow, deactivateWindow

on deactivateWindow

Syntax on deactivateWindow
 statement(s)
 end

Description System message and event handler; contains statements that run when the window that the movie is playing in is deactivated. The on deactivate event handler is a good place for Lingo that you want executed whenever a window is deactivated.

Example This handler plays the sound Snore when the window that the movie is playing in is deactivated:

```
on deactivateWindow
    puppetSound 2, "Snore"
end
```

defaultRect

Syntax

member(*whichFlashOrVectorShapeMember*).defaultRect

the defaultRect of member *whichFlashOrVectorShapeMember*

Description

Cast member property; controls the default size used for all new sprites created from a Flash movie or vector shape cast member. The defaultRect setting also applies to all existing sprites that have not been stretched on the Stage. You specify the property values as a Director rectangle; for example, rect(0,0,32,32).

The defaultRect member property is affected by the cast member's defaultRectMode member property. The defaultRectMode property is always set to #Flash when a movie is inserted into a cast, which means the original defaultRect setting is always the size of the movie as it was originally created in Flash. Setting defaultRect after that implicitly changes the cast member's defaultRectMode property to #fixed.

This property can be tested and set.

Example

This handler accepts a cast reference and a rectangle as parameters. It then searches the specified cast for Flash cast members and sets their defaultRect property to the specified rectangle.

```
on setDefaultFlashRect whichCast, whichRect
    repeat with i = 1 to the number of members of castLib whichCast
        if member(i, whichCast).type = #flash then
            member(i, whichCast).defaultRect = whichRect
        end if
    end repeat
end
```

See also

defaultRectMode, flashRect

defaultRectMode

Syntax

member(*whichVectorOrFlashMember*).defaultRectMode

the defaultRectMode of member *whichVectorOrFlashMember*

Description

Cast member property; controls how the default size is set for all new sprites created from Flash movie or vector shape cast members. You specify the property value as a Director rectangle; for example, rect(0,0,32,32).

The defaultRectMode property does not set the actual size of a Flash movie's default rectangle; it only determines how the default rectangle is set. The defaultRectMode member property can have these values:

- #flash (default)—Sets the default rectangle using the size of the movie as it was originally created in Flash.
- #fixed—Sets the default rectangle using the fixed size specified by the defaultRect member property.

The defaultRect member property is affected by the cast member's defaultRectMode member property. The defaultRectMode property is always set to #flash when a movie is inserted into a cast, which means the original defaultRect setting is always the size of the movie as it was originally created in Flash. Setting defaultRect after that implicitly changes the cast member's defaultRectMode property to #fixed.

This property can be tested and set.

Example This handler accepts a cast reference and a rectangle as parameters. It then searches the specified cast for Flash cast members, sets their defaultRectMode property to #fixed, and then sets their defaultRect property to rect(0,0,320,240).

```
on setDefaultRectSize whichCast
repeat with i = 1 to the number of members of castLib whichCast
    if member(i, whichCast).type = #flash then
        member(i, whichCast).defaultRectMode = #fixed
        member(i, whichCast).defaultRect = rect(0,0,320,240)
    end if
end repeat
end
```

See also flashRect, defaultRect

delay

Syntax delay *numberOfTicks*

Description Command; pauses the playback head for a given amount of time. The integer expression *numberOfTicks* specifies the number of ticks to wait, where each tick is 1/60 of a second. The only mouse and keyboard activity possible during this time is stopping the movie by pressing Control+Alt+period (Windows) or Command+period (Macintosh). Because it increases the time of individual frames, the delay command is useful for controlling the playback rate of a sequence of frames.

The delay command can be applied only when the playback head is moving. However, when delay is in effect, handlers still run: only the playback head halts, not script execution. Place scripts that use the delay command in either an on enterFrame or on exitFrame handler.

To mimic the behavior of a halt in a handler when the playback head is not moving, use the startTimer command or assign the current value of timer to a variable and wait for the specified amount of time to pass before exiting the frame.

Example This handler delays the movie for 2 seconds when the playback head exits the current frame:

```
on exitFrame
    delay 2 * 60
end
```

Example This handler, which can be placed in a frame script, delays the movie a random number of ticks:

```
on keyDown
    if the key = RETURN then delay random(180)
end
```

Example The first of these handlers sets a timer when the playback head leaves a frame. The second handler, assigned to the next frame, loops in the frame until the specified amount of time passes:

```
--script for first frame
on exitFrame
    global gTimer
    set gTimer = the ticks
end

--script for second frame
on exitFrame
    global gTimer
    if the ticks < gTimer + (10 * 60) then
        go to the frame
    end if
end
```

See also startTimer, ticks, timer

delete

Syntax delete *chunkExpression*

Description Command; deletes the specified chunk expression (character, word, item, or line) in any string container. Sources of strings include field cast members and variables that hold strings.

To see an example of delete used in a completed movie, see the Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement deletes the first word of line 3 in the field cast member Address:

```
delete word 1 of line 3 of member "Address"
```

Example The same chunk of text may also be deleted with the syntax:

```
delete member("Address").line[3].word[1]
```

Example This statement deletes the first character of the string in the variable bidAmount if that character is the dollar sign ("\$"):

```
if bidAmount.char[1] = "$" then delete bidAmount.char[1]
```

See also char...of, field, item...of, line...of, word...of; hilite (cast member property), paragraph

deleteAll

Syntax `list.deleteAll()`

`deleteAll list`

Description List command; deletes all items in the specified list without changing the list type.

Example This statement deletes every item in the list named propList:

`propList.deleteAll()`

deleteAt

Syntax `list.deleteAt(number)`

`deleteAt list, number`

Description List command; deletes the item in the position specified by *number* from the linear or property list specified by *list*.

The deleteAt command checks whether an item is in a list; if you try to delete an object that isn't in the list, Director displays an alert.

Example This statement deletes the second item from the list named designers, which contains [gee, kayne, ohashi]:

```
designers = ["gee", "kayne", "ohashi"]  
designers.deleteAt(2)
```

The result is the list [gee, ohashi].

This handler checks whether an object is in a list before attempting to delete it:

```
on myDeleteAt theList, theIndex  
    if theList.count < theIndex then  
        beep  
    else  
        theList.deleteAt(theIndex)  
    end if  
end
```

See also addAt

deleteFrame

Syntax deleteFrame

Description Command; deletes the current frame and makes the next frame the new current frame during a Score generation session only.

Example The following handler checks whether the sprite in channel 10 of the current frame has gone past the right edge of a 640-by-480-pixel Stage and deletes the frame if it has:

```
on testSprite
    beginRecording
        if sprite(10).locH > 640 then deleteFrame
    endRecording
end
```

See also beginRecording, endRecording, updateFrame

deleteOne

Syntax *list.deleteOne(value)*

deleteOne list, value

Description List command; deletes a value from a linear or property list. For a property list, deleteOne also deletes the property associated with the deleted value. If the value appears in the list more than once, deleteOne deletes only the first occurrence.

Attempting to delete a property has no effect.

Example The first statement creates a list consisting of the days Tuesday, Wednesday, and Friday. The second statement deletes the name Wednesday from the list.

```
days = ["Tuesday", "Wednesday", "Friday"]
days.deleteOne("Wednesday")
put days
```

The put days statement causes the Message window to display the result:

```
-- ["Tuesday", "Friday"].
```

deleteProp

Syntax *list.deleteProp(item)*

deleteProp list, item

Description List command; deletes the specified item from the specified list.

- For linear lists, replace *item* with the number identifying the list position of the item to be deleted. The deleteProp command for linear lists is the same as the deleteAt command. If the number is greater than the number of items in the list, a script error occurs.
- For property lists, replace *item* with the name of the property to be deleted. Deleting a property also deletes its associated value. If the list has more than one of the same property, only the first property in the list is deleted.

Example This statement deletes the color property from the list [#height:100, #width: 200, #color: 34, #ink: 15], which is called spriteAttributes:

```
spriteAttributes.deleteProp(#color)
```

The result is the list [#height:100, #width: 200, #ink: 15].

See also deleteAt

deleteVertex()

Syntax *member(memberRef).deleteVertex(indexToRemove)*

deleteVertex(member memberRef, indexToRemove)

Description Vector shape command; removes an existing vertex of a vector shape cast member in the index position specified.

Example This line removes the second vertex point in the vector shape Archie:

```
member("Archie").deleteVertex(2)
```

See also addVertex, moveVertex(), originMode, vertexList

depth

Syntax

imageObject.depth

member(*whichCastMember*).depth

the depth of member *whichCastMember*

Description

Image object or bitmap cast member property; displays the color depth of the given image object or bitmap cast member.

Depth	Number of Colors
1	Black and white
2	4 colors
4, 8	16 or 256 palette-based colors, or gray levels
16	Thousands of colors
32	Millions of colors

This property can be tested but not set.

Example

This statement displays the color depth of the image object stored in the variable newImage. The output appears in the Message window.

```
put newImage.depth
```

Example

This statement displays the color depth of the cast member Shrine in the Message window:

```
put member("Shrine").depth
```

deskTopRectList

Syntax

the deskTopRectList

Description

System property; displays the size and position on the desktop of the monitors connected to a computer. This property is useful for checking whether objects such as windows, sprites, and pop-up windows appear entirely on one screen.

The result is a list of rectangles, where each rectangle is the boundary of a monitor. The coordinates for each monitor are relative to the upper left corner of monitor 1, which has the value (0,0). The first set of rectangle coordinates is the size of the first monitor. If a second monitor is present, a second set of coordinates shows where the corners of the second monitor are relative to the first monitor.

This property can be tested but not set.

Example This statement tests the size of the monitors connected to the computer and displays the result in the Message window:

```
put the deskTopRectList  
-- [rect(0,0,1024,768), rect(1025, 0, 1665, 480)]
```

The result shows that the first monitor is 1024 by 768 pixels and the second monitor is 640 by 480 pixels.

Example This handler tells how many monitors are in the current system:

```
on countMonitors  
    return the deskTopRectList.count  
end
```

digitalVideoTimeScale

Syntax the digitalVideoTimeScale

Description System property; determines the time scale, in units per second, that the system uses to track digital video cast members.

For example, if digitalVideoTimeScale is set to:

- 100—The time scale is 1/100 of a second (and the movie is tracked in 100 units per second).
- 500—The time scale is 1/500 of a second (and the movie is tracked in 500 units per second).
- 0—Director uses the time scale of the movie that is currently playing.

Set digitalVideoTimeScale to precisely access tracks by ensuring that the system's time unit for video is a multiple of the digital video's time unit.

This property can be tested and set.

Example This statement sets the time scale that the system uses to measure digital video to 600 units per second:

```
the digitalVideoTimeScale to 600
```

digitalVideoType

Syntax	<code>member(<i>whichCastMember</i>).digitalVideoType</code> the digitalVideoType of member <i>whichCastMember</i>
Description	Cast member property; indicates the format of the specified digital video. Possible values are #quickTime or #videoForWindows. This property can be tested but not set.
Example	The following statement tests whether the cast member Today's Events is a QuickTime or AVI (Audio-Video Interleaved) digital video and displays the result in the Message window: <code>put member("Today's Events").digitalVideoType</code>
See also	<code>quickTimeVersion()</code>

directToStage

Syntax	<code>member(<i>whichCastMember</i>).directToStage</code> the directToStage of member <i>whichCastMember</i> <code>sprite(<i>whichSprite</i>).directToStage</code> the directToStage of sprite <i>whichSprite</i>
Description	Sprite property and member property; determines the layer where a digital video, animated GIF, vector shape, or Flash Asset cast member plays. If this property is TRUE (1), the cast member plays in front of all other layers on the Stage, and ink effects have no affect. If this property is FALSE (0), the cast member can appear in any layer of the Stage's animation planes, and ink effects affect the appearance of the sprite. <ul style="list-style-type: none">• Use the syntax <code>member(<i>whichCastMember</i>).directToStage</code> for digital video or animated GIFs.• Use the syntax <code>sprite(<i>whichSprite</i>).directToStage</code> for Flash or vector shapes. Using this property may improve the playback performance of supported cast members. No other cast member can appear in front of a directToStage sprite. Also, ink effects do not affect the appearance of a directToStage sprite. When a sprite's directToStage property is TRUE, Director draws the sprite directly to the screen without first compositing it in Director's offscreen buffer. The result can be similar to the trails ink effect of the Stage. Explicitly refresh a trailed area by turning the directToStage property off and on, using a full-screen transition, or “wiping” another sprite across this area. (In Windows, you can branch to another similar screen, and the video won't completely disappear.)

To see an example of `directToStage` used in a completed movie, see the QT and Flash movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement makes the QuickTime movie *The Residents* always play in the top layer of the Stage:

```
member("The Residents").directToStage = 1
```

dither

Syntax `member(whichMember).dither`

the dither of member *whichMember*

Description Bitmap cast member property; dithers the cast member when it is displayed at a color depth of 8 bits or less (256 colors) if the display must show a color gradation not in the cast member (TRUE), or tells Director to choose the nearest color out of those available in the current palette (FALSE).

For both performance and quality reasons, you should set `dither` to TRUE only when higher display quality is necessary. Dithering is slower than remapping, and artifacts may be more apparent when animating over a dithered image.

If the color depth is greater than 8 bits, this property has no effect.

See also `depth`

do

Syntax `do stringExpression`

Description Command; evaluates *stringExpression* and executes the result as a Lingo statement. This command is useful for evaluating expressions that the user has typed and for executing commands stored in string variables, fields, arrays, and files.

Using uninitialized local variables within a `do` command creates a compile error. Initialize any local variables in advance.

Note: This command does not allow global variables to be declared; these variables must be declared in advance.

The `do` command works with multiple-line strings as well as single lines.

Example This statement performs the statement contained within quotation marks:

```
do "beep 2"
```

```
do commandList[3]
```

doubleClick

Syntax the doubleClick

Description Function; tests whether two mouse clicks within the time set for a double-click occurred as a double-click rather than two single clicks (TRUE), or if they didn't occur within the time set, treats them as single clicks (FALSE).

Example This statement branches the playback head to the frame Enter Bid when the user double-clicks the mouse button:

```
if the doubleClick then go to frame "Enter Bid"
```

Example The following handler tests for a double-click. When the user clicks the mouse, a repeat loop runs for the time set for a double-click (20 ticks in this case). If a second click occurs within 20 ticks, the doubleClickAction handler runs. If a second click doesn't occur within the specified period, the singleClickAction handler runs:

```
on mouseUp
    if the doubleClick then exit
    startTimer
repeat while the timer < 20
    if the mouseDown then
        doubleClickAction
        exit
    end if
end repeat
singleClickAction
end mouseUp
```

See also clickOn, the mouseDown (system property), the mouseUp (system property)

downloadNetThing

Syntax downloadNetThing *URL*, *localFile*

Description Command; copies a file from the Internet to a file on the local disk, while the current movie continues playing. Use netDone to find out whether downloading is finished.

- *URL*—The file name of any object that can be downloaded: for example, an FTP or HTTP server, an HTML page, an external cast member, a Director movie, or a graphic
- *localFile*—The pathname and file name for the file on the local disk

Director movies in authoring mode and projectors support the downloadNetThing command, but the Shockwave player does not. This protects users from unintentionally copying files from the Internet.

Although many network operations can be active at one time, running more than four concurrent operations usually slows down performance unacceptably.

Neither the Director movie's cache size nor the setting for the Check Documents option affects the behavior of the downloadNetThing command.

Note: Director for Java does not support the downloadNetThing command.

Example These statements download an external cast member from a URL to the Director application folder and then make that file the external cast member named Cast of Thousands:

```
downLoadNetThing("http://www.cbDeMille.com/Thousands.cst", the \
applicationPath&"Thousands.Cst")
castLib("Cast of Thousands").fileName = the applicationPath&"Thousands.Cst"
```

See also importFileInto, netDone(), preloadNetThing()

draw()

Syntax

```
imageObject.draw(x1, y1, x2, y2, colorObjectOrParameterList)
```

```
imageObject.draw(point(x, y), point(x, y), colorObjectOrParameterList)
```

```
imageObject.draw(rect, colorObjectOrParameterList)
```

Description

This function draws a line or an unfilled shape of color *colorObject* in a rectangular region of the given image object, as specified in any of the three ways shown. The draw returns a value of 1 if there is no error. You can use the optional property list *ParameterList* function to specify the following shape properties:

Property	Description
#shapeType	A symbol value of #oval, #rect, #roundRect, or #line. The default is #line.
#lineSize	The width of the line to use in drawing the shape.
#color	A color object, which determines the color of the shape border.

If you do not provide a parameter list, this function draws a 1-pixel line between the first and second points given or between the upper left and lower right corners of the given rectangle.

For best performance, with 8-bit or lower images the *colorObject* should contain an indexed color value. For 16- or 32-bit images, use an RGB color value.

If you want to fill a solid region, use the fill() function.

Example This statement draws a 1-pixel, dark red, diagonal line from point (0, 0) to point (128, 86) within the image of member Happy.

```
member("Happy").image.draw(0, 0, 128, 86, rgb(150,0,0))
```

Example	This statement draws a dark red, 3-pixel unfilled oval within the image of member Happy. The oval is drawn within the rectangle (0, 0, 128, 86).
	<pre>member("Happy").image.draw(0, 0, 128, 86, [#shapeType:#oval, #lineSize:3, \ #color: rgb(150, 0, 0)])</pre>
See also	color() , copyPixels() , fill() , setPixel()

drawRect

Syntax	<pre>window <i>windowName</i>.drawRect</pre>
	the drawRect of window <i>windowName</i>
Description	Window property; identifies the rectangular coordinates of the Stage of the movie that appears in the window. The coordinates are given as a rectangle, with entries in the order left, top, right, and bottom.
	This property is useful for scaling or panning movies, but it does not rescale text and field cast members. Scaling bitmaps can affect performance.
	This property can be tested and set.
Example	This statement displays the current coordinates of the movie window called Control Panel:
	<pre>put the drawRect of window "Control Panel" -- rect(10, 20, 200, 300).</pre>
Example	This statement sets the rectangle of the movie to the values of the rectangle named movieRectangle. The part of the movie within the rectangle is what appears in the window.
	<pre>set the drawRect of window "Control Panel" to movieRectangle</pre>
Example	The following lines will cause the Stage to fill the main monitor area:
	<pre>(the stage).drawRect = the desktopRectList[1] (the stage).rect = the desktopRectList[1]</pre>
See also	deskTopRectList , rect() , sourceRect

dropShadow

Syntax	<pre>member(<i>whichCastMember</i>).dropShadow</pre>
	the dropShadow of member <i>whichCastMember</i>
Description	Cast member property; determines the size of the drop shadow in pixels, for text in a field cast member.
Example	This statement sets the drop shadow of the field cast member Comment to 5 pixels:

```
member("Comment").dropShadow = 5
```

duplicateFrame

Syntax `duplicateFrame`

Description Command; duplicates the current frame and its content, inserts the duplicate frame after the current frame, and then makes the duplicate frame the current frame. This command can be used during Score generation only.

The `duplicateFrame` command performs the same function as the `insertFrame` command.

Example When used in the following handler, the `duplicateFrame` command creates a series of frames that have cast member Ball in the external cast Toys assigned to sprite channel 20. The number of frames is determined by the argument `numberOfFrames`.

```
on animBall numberOfFrames
    beginRecording
        sprite(20).member = member("Ball", "Toys")
        repeat with i = 0 to numberOfFrames
            duplicateFrame
        end repeat
    endRecording
end
```

duplicate() (list function)

Syntax `(oldList).duplicate()`

`duplicate(oldList)`

Description List function; returns a copy of a list and copies nested lists (list items that also are lists) and their contents. The function is useful for saving a list's current content.

When you assign a list to a variable, the variable contains a reference to the list, not the list itself. This means any changes to the copy also affect the original list.

To see an example of `duplicate()` (list function) used in a completed movie, see the Vector Shapes movie in the LearningLingo Examples folder inside the Director application folder.

Example This statement makes a copy of the list `CustomersToday` and assigns it to the variable `CustomerRecord`:

```
CustomerRecord = CustomersToday.duplicate()
```

duplicate() (image function)

Syntax `imageObject.duplicate()`

Description This function creates and returns a copy of the given `imageObject`. The new image is completely independent of the original, and isn't linked to any cast member.

If you plan to make a lot of changes to an image, it's better to make a copy that's independent of a cast member.

Example This statement creates a new image object from the image of cast member Lunar Surface and places the new image object into the variable `workingImage`:

```
workingImage = member("Lunar Surface").image.duplicate()
```

See also [duplicate member](#)

duplicate member

Syntax `member(originalMember).duplicate()`

`member(originalMember).duplicate({new})`

`duplicate member original {, new}`

Description Command; makes a copy of the cast member specified by `original`. The optional `new` parameter specifies a specific Cast window location for the duplicate cast member. If the `new` parameter isn't included, the duplicate cast member is placed in the first open Cast window position.

This command is best used during authoring rather than run time because it creates another cast member in memory, which could result in memory problems. Use the command during authoring if you want the changes to the cast to be permanently saved with the file.

Example This statement makes a copy of cast member Desk and places it in the first empty Cast window position:

```
member("Desk").duplicate()
```

Example This statement makes a copy of cast member Desk and places it in the Cast window at position 125:

```
member("Desk").duplicate(125)
```

duration

Syntax

```
member(whichCastMember).duration
```

the duration of member *whichCastMember*

Description

Cast member property; determines the duration of the specified Shockwave Audio (SWA), transition, and QuickTime cast members.

- When *whichCastMember* is a streaming sound file, this property indicates the duration of the sound. The duration property returns 0 until streaming begins. Setting preLoadTime to 1 second allows the bit rate to return the actual duration.
- When *whichCastMember* is a digital video cast member, this property indicates the digital video's duration. The value is in ticks.
- When *whichCastMember* is a transition cast member, this property indicates the transition's duration. The value for the transition is in milliseconds. During playback, this setting has the same effect as the Duration setting in the Frame Transition dialog box.

This property can be tested for all cast members that support it, but only set for transitions.

To see an example of duration used in a completed movie, see the QT and Flash movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

If the SWA cast member Louie Prima has been preloaded, this statement displays the sound's duration in the field cast member Duration Displayer:

```
on exitFrame
    if member("Louie Prima").state = 2 then
        member("Duration Displayer").text = member("Louie Prima").duration
    end if
end
```

Example

You can use a behavior on a digital video sprite to loop the playback head in the current frame until the movie is finished playing, allowing it to continue when the end is reached:

```
property spriteNum

on exitFrame me
    myMember = sprite(spriteNum).member
    myDuration = member(myMember).duration
    myMovietime = sprite(spriteNum).movieTime
    if myDuration > myMovietime then
        go to the frame
    else
        go to the frame + 1
    end if
end
```

editable

Syntax

```
member(whichCastMember).editable  
the editable of member whichCastMember  
sprite(whichSprite).editable  
the editable of sprite whichSprite
```

Description

Cast member and sprite property; determines whether the specified field cast member can be edited on the Stage (TRUE) or not (FALSE).

When the cast member property is set, the setting is applied to all sprites that contain the field. When the sprite property is set, only the specified sprite is affected.

You can also make a field cast member editable by using the Editable option in the Field Cast Member Properties dialog box.

You can make a field sprite editable by using the Editable option in the Score.

For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet.

This property can be tested and set.

Example

This statement makes the field cast member Answer editable:

```
member("Answer").editable = TRUE
```

Example

This handler first makes the sprite channel a puppet and then makes the field sprite editable:

```
on myNotes  
    puppetSprite 5, TRUE  
    sprite(5).editable = TRUE  
end
```

Example

This statement checks whether a field sprite is editable and displays a message if it is:

```
if sprite(13).editable = TRUE then  
    member("Notice").text = "Please enter your answer below."  
end if
```

editShortCutsEnabled

Syntax	the editShortCutsEnabled
Description	Movie property; determines whether cut, copy, and paste operations and their keyboard shortcuts function in the current movie. When set to TRUE, these text operations function. When set to FALSE, these operations are not allowed.
Example	This property can be tested and set. The default is TRUE for movies made in Director 8, FALSE for movies made in versions of Director prior to Director 8. This statement disables cut, copy, and paste operations: the editShortCutsEnabled = 0

elapsedTime

Syntax	sound(<i>channel/Num</i>).elapsedTime the elapsedTime of sound <i>channel/Num</i>
Description	This read-only property gives the time, in milliseconds, that the current sound member in the given sound channel has been playing. It starts at 0 when the sound begins playing and increases as the sound plays, regardless of any looping, setting of the currentTime or other manipulation. Use the currentTime to test for the current absolute time within the sound. The value of this property is a floating-point number, allowing for measurement of sound playback to fractional milliseconds.
Example	This idle handler displays the elapsed time for sound channel 4 in a field on the Stage during idles: on idle member("time").text = string(sound(4).elapsedTime) end idle
See also	currentTime, loopCount, loopsRemaining, rewind()

EMPTY

Syntax	EMPTY
Description	Character constant; represents the empty string, "", a string with no characters.
Example	This statement erases all characters in the field cast member Notice by setting the field to EMPTY: member("Notice").text = EMPTY

emulateMultiButtonMouse

Syntax the emulateMultiButtonMouse

Description System property; determines whether a movie interprets a mouse click with the Control key pressed on the Macintosh the same as a right mouse click in Windows (TRUE) or not (FALSE).

Right-clicking has no direct Macintosh equivalent.

Setting this property to TRUE lets you provide consistent mouse button responses for cross-platform movies.

Example The following statement checks if the computer is a Macintosh and if so, sets the emulateMultiButtonMouse property to TRUE:

```
if the platform contains "Macintosh" then the emulateMultiButtonMouse = TRUE
```

See also keyPressed(), rightMouseDown (system property), rightMouseUp (system property)

enabled

Syntax the enabled of menuItem *whichItem* of menu *whichMenu*

Description Menu item property; determines whether the menu item specified by *whichItem* is displayed in plain text and is selectable (TRUE, default) or appears dimmed and is not selectable (FALSE).

The expression *whichItem* can be either a menu item name or a menu item number. The expression *whichMenu* can be either a menu name or a menu number.

The enabled property can be tested and set.

Note: Menus are not available in Shockwave

Example This handler enables or disables all the items in the specified menu. The argument *theMenu* specifies the menu; the argument *Setting* specifies TRUE or FALSE. For example, the calling statement ableMenu ("Special", FALSE) disables all the items in the Special menu.

```
on ableMenu theMenu, vSetting
    set n = the number of menuitems of menu theMenu
    repeat with i = 1 to n
        set the enabled of menuItem i of menu theMenu to vSetting
    end repeat
end ableMenu
```

See also name (menu property), number (menus), checkMark, script, number (menu items)

enableHotSpot

Syntax

enableHotSpot(sprite *whichQTVRSprite*, hotSpotID, trueOrFalse)

Description

QTVR (QuickTime VR) command; determines whether the specified hot spot for the specified QTVR sprite is enabled (TRUE), or disabled (FALSE).

end

Syntax

end

Description

Keyword; marks the end of handlers and multiple-line control structures.

end case

Syntax

end case

Description

Keyword; ends a case statement.

Example

This handler uses the end case keyword to end the case statement:

```
on keyDown
    case the key
        of "A": go to frame "Apple"
        of "B", "C" :
            puppetTransition 99
            go to frame "Mango"
        otherwise beep
    end case
end keyDown
```

See also

case

endColor

Syntax

the endColor of member *whichCastMember*

Description

Vector shape cast member property; the ending color of a gradient shape's fill specified as an RGB value.

endColor is only valid when the fillMode is set to #gradient, and the starting color is set with fillColor.

This property can be tested and set.

To see an example of endColor used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.

See also

color(), fillColor, fillMode

endFrame()

Syntax `sprite(whichSprite).endFrame`

Description Function; returns the frame number of the end frame of the sprite span.

This function is useful in determining the span in the Score of a particular sprite. This function is available only in a frame that contains the sprite. It cannot be applied to sprites in different frames of the movie, nor is it possible to set this value.

Example This statement output reports the ending frame of the sprite in channel 5 in the Message window:

```
put sprite(5).endFrame
```

See also [startFrame](#)

endRecording

Syntax `endRecording`

Description Keyword; ends a Score update session. You can resume control of Score channels through puppeting after the `endRecording` keyword is issued.

Example When used in the following handler, the `endRecording` keyword ends the Score generation session:

```
on animBall numberOfRowsInSection
beginRecording
    horizontal = 0
    vertical = 100
repeat with i = 1 to numberOfRowsInSection
    go to frame i
    sprite(20).member = member "Ball"
    sprite(20).locH = horizontal
    sprite(20).locV = vertical
    horizontal = horizontal + 3
    vertical = vertical + 2
    updateFrame
end repeat
endRecording
end
```

See also [beginRecording](#), [scriptNum](#), [tweened](#), [scriptNum](#), [updateFrame](#)

end repeat

See [repeat while](#), [repeat with](#), [repeat with...in list](#), [repeat with...down to](#)

on endSprite

Syntax on endSprite

```
    statement(s)  
end
```

Description System message and event handler; contains Lingo that runs when the playback head leaves a sprite and goes to a frame in which the sprite doesn't exist. It is generated after exitFrame.

Place on endSprite handlers in a behavior script.

Director destroys instances of any behavior scripts attached to the sprite immediately after the endSprite event occurs.

The event handler is passed the behavior or frame script reference me if used in a behavior. This endSprite message is sent after the exitFrame message if the playback head plays to the end of the frame.

The go, play, and updateStage commands are disabled in an on endSprite handler.

Example This handler runs when the playback head exits a sprite:

```
on endSprite me  
    -- clean up  
    gNumberOfSharks = gNumberOfSharks - 1  
    puppetSound(5,0)  
end
```

See also on beginSprite, on exitFrame

endTime

Syntax sound(*channelNum*).endTime

the endTime of sound *channelNum*

Description This property is the specified end time of the currently playing, paused or queued sound. This is the time within the sound member when it will stop playing. It's a floating-point value, allowing for measurement and control of sound playback to fractions of milliseconds. The default value is the normal end of the sound.

This property may be set to a value other than the normal end of the sound only when passed as a parameter with the queue() or setPlayList() commands.

Example This Lingo checks whether the sound member Jingle is set to play all the way through in sound channel 1:

```
if sound(1).startTime > 0 and sound(1).endTime < member("Jingle").duration then  
    alert "Not playing the whole sound"  
end if
```

See also setPlaylist(), queue(), startTime

ENTER

Syntax	ENTER
Description	Character constant; represents the Enter key (Windows) or the Return key (Macintosh) for a carriage return. On PC keyboards, the element ENTER refers only to the Enter key on the numeric keypad.
Example	For a movie that plays back as an applet, use RETURN to specify both the Return key in Windows and the Enter key on the Macintosh.
See also	This statement checks whether the Enter key is pressed and if it is, sends the playback head to the frame addSum: on keyDown if the key = ENTER then go to frame "addSum" end

on enterFrame

Syntax	on enterFrame <i>statement(s)</i> end
Description	System message and event handler; contains statements that run each time the playback head enters the frame. Place on enterFrame handlers in behavior, frame, or movie scripts, as follows: <ul style="list-style-type: none">• To assign the handler to an individual sprite, put the handler in a behavior attached to the sprite.• To assign the handler to an individual frame, put the handler in the frame script.• To assign the handler to every frame (unless you explicitly instruct the movie otherwise), put the on enterFrame handler in a movie script. The handler executes every time the playback head enters a frame unless the frame script has its own handler. If the frame script has its own handler, the on enterFrame handler in the frame script overrides the on enterFrame handler in the movie script. The order of frame events is stepFrame, prepareFrame, enterFrame, and exitFrame. This event is passed the object reference me if used in a behavior.

Example This handler turns off the puppet condition for sprites 1 through 5 each time the playback head enters the frame:

```
on enterFrame
    repeat with i = 1 to 5
        puppetSprite i, FALSE
    end repeat
end
```

environment

Syntax

the environment
the envirionment.*propertyName*

Description System property; this property contains a list with information about the environment under which the Director content is currently running.

This design enables Macromedia to add information to the environment property in the future, without affecting existing movies.

The information is in the form of property and value pairs for that area.

#shockMachine	Integer TRUE or FALSE value indicating whether the movie is playing in ShockMachine.
#shockMachineVersion	String indicating the installed version number of ShockMachine.
#platform	String containing "Macintosh,PowerPC", or "Windows,32". This is based on the current OS and hardware that the movie is running under.
#runMode	String containing "Author", "Projector", "Plugin", or "Java Applet". This is based on the current application that the movie is running under.
#colorDepth	Integer representing the bit depth of the monitor the Stage appears on. Possible values are 1, 2, 4, 8, 16, or 32.
#internetConnected	Symbol indicating whether the computer the movie is playing on has an active Internet connection. Possible values are #online and #offline.
#uiLanguage	String indicating the language the computer is using to display its user interface. This can be different from the #osLanguage on computers with specific language kits installed.
#osLanguage	String indicating the native language of the computer's operating system.
#productBuildVersion	String indicating the internal build number of the playback application.

The properties contain exactly the same information as the properties and functions of the same name.

Example This statement displays the environment list in the Message window:

```
put the environment  
-- [#shockMachine: 0, #shockMachineVersion: "", #platform: "Macintosh,PowerPC",  
#runMode: "Author", #colorDepth: 32, #internetConnected: #online, #uiLanguage:  
"English", #osLanguage: "English", #productBuildVersion: "151"]
```

See also colorDepth, platform, runMode

erase member

Syntax member(*whichCastMember*).erase()

erase member *whichCastMember*

Description Command; deletes the specified cast member and leaves its slot in the Cast window empty.

For best results, use this command during authoring and not in projectors, which can cause memory problems.

Example This statement deletes the cast member named Gear in the Hardware cast:

```
member("Gear", "Hardware").erase()
```

Example This handler deletes cast members start through finish:

```
on deleteMember start, finish  
    repeat with i = start to finish  
        member(i).erase()  
    end repeat  
end on deleteMember
```

See also new()

on EvalScript

Syntax on EvalScript *aParam*

statement(s)

end

Description System message and event handler; in a Shockwave movie, contains statements that run when the handler receives an EvalScript message from a browser. The parameter is a string passed in from the browser.

- The EvalScript message can include a string that Director can interpret as a Lingo statement. Lingo cannot accept nested strings. If the handler you are calling expects a string as a parameter, pass the parameter as a symbol.
- The on EvalScript handler is called by the EvalScript() scripting method from JavaScript or VBScript in a browser.

The Director player for Java doesn't support the on EvalScript handler. To enable communication between an applet and a browser, use Java, JavaScript, or VBScript.

Include only those behaviors in on EvalScript that you want users to control; for security reasons, don't give complete access to behaviors.

Note: If you place a return at the end of your EvalScript handler, the value returned can be used by JavaScript in the browser.

Example This shows how to make the playback head jump to a specific frame depending on what frame is passed in as the parameter:

```
on EvalScript aParam
    go frame aParam
end
```

Example This handler runs the statement go frame aParam if it receives an EvalScript message that includes dog, cat, or tree as an argument:

```
on EvalScript aParam
    case aParam of
        "dog", "cat", "tree": go frame aParam
    end case
end
```

A possible calling statement for this in JavaScript would be EvalScript ("dog").

Example This handler takes an argument that can be a number or symbol:

```
on EvalScript aParam
    if word 1 of aParam = "myHandler" then
        do aParam
    end if
end
```

Example This handler normally requires a string as its argument. The argument is received as a symbol and then converted to a string within the handler by the string function:

```
on myHandler aParam
    go to frame string(aParam)
end
```

See also externalEvent, return (keyword)

eventPassMode

Syntax

```
sprite(whichFlashSprite).eventPassMode  
the eventPassMode of sprite whichFlashSprite  
member(whichFlashMember).eventPassMode  
the eventPassMode of member whichFlashMember
```

Description

Flash cast member property and sprite property; controls when a Flash movie passes mouse events to behaviors that are attached to sprites that lie underneath the flash sprite. The eventPassMode property can have these values:

- #passAlways (default)—Always passes mouse events.
- #passButton—Passes mouse events only when a button in the Flash movie is clicked.
- #passNotButton—Passes mouse events only when a nonbutton object is clicked.
- #passNever—Never passes mouse events.

This property can be tested and set.

Example

This frame script checks to see whether the buttons in a Flash movie sprite are currently enabled, and if so, sets eventPassMode to #passNotButton; if the buttons are disabled, the script sets eventPassMode to #passAlways. The effect of this script is that:

- Mouse events on nonbutton objects always pass to sprite scripts.
- Mouse events on button objects are passed to sprite scripts when the buttons are disabled. When the buttons are enabled, mouse events on buttons are stopped.

```
on enterFrame  
    if sprite(5).buttonsEnabled = TRUE then  
        sprite(5).eventPassMode= #passNotButton  
    else  
        sprite(5).eventPassMode = #passAlways  
    end if  
end
```

exit

Syntax

exit

Description Keyword; instructs Lingo to leave a handler and return to where the handler was called. If the handler is nested within another handler, Lingo returns to the main handler.

Example The first statement of this script checks whether the monitor is set to black and white and then exits if it is:

```
on setColors
    if the colorDepth = 1 then exit
    sprite(1).foreColor = 35
end
```

See also abort, halt, quit, pass, return (keyword)

on exitFrame

Syntax

on exitFrame

statement(s)

end

Description System message and event handler; contains statements that run each time the playback head exits the frame that the on exitFrame handler is attached to. The on exitFrame handler is a useful place for Lingo that resets conditions that are no longer appropriate after leaving the frame.

Place on exitFrame handlers in behavior, frame, or movie scripts, as follows:

- To assign the handler to an individual sprite, put the handler in a behavior attached to the sprite.
- To assign the handler to an individual frame, put the handler in the frame script.
- To assign the handler to every frame unless explicitly instructed otherwise, put the handler in a movie script. The on exitFrame handler then executes every time the playback head exits the frame unless the frame script has its own on exitFrame handler. When the frame script has its own on exitFrame handler, the on exitFrame handler in the frame script overrides the one in the movie script.

This event is passed the sprite script or frame script reference me if it is used in a behavior. The order of frame events is prepareFrame, enterFrame, and exitFrame.

Example	This handler turns off all puppet conditions when the playback head exits the frame:
	<pre>on exitFrame me repeat with i = 48 down to 1 sprite(i).puppet = FALSE end repeat end</pre>
Example	This handler branches the playback head to a specified frame if the value in the global variable vTotal exceeds 1000 when the playback head exits the frame:
	<pre>global vTotal on exitFrame if vTotal > 1000 then go to frame "Finished" end</pre>

See also on enterFrame

exitLock

Syntax	the exitLock
Description	Movie property; determines whether a user can quit to the Windows desktop or Macintosh Finder from projectors (TRUE) or not (FALSE, default). The user can quit to the desktop by pressing Control+period (Windows) or Command+period (Macintosh), Control+Q (Windows) or Command+Q (Macintosh), or Control+W (Windows) or Command+W (Macintosh); the Escape key is also supported in Windows.
	This property can be tested and set.
Example	This statement sets the exitLock property to TRUE: set the exitLock to TRUE
Example	Assuming that exitLock is set to TRUE, nothing occurs automatically when the Control+period/Q/W, Esc, or Command+period/Q/W keys are used. This handler checks keyboard input for keys to exit and takes the user to a custom quit sequence: <pre>on checkExit if the commandDown and (the key = "." or the key = "q") and the exitLock = TRUE then go to frame "quit sequence" end</pre>

exit repeat

Syntax exit repeat

Description Keyword; instructs Lingo to leave a repeat loop and go to the statement following the end repeat statement but to remain within the current handler or method.

The exit repeat keyword is useful for breaking out of a repeat loop when a specified condition—such as two values being equal or a variable being a certain value—exists.

Example This handler searches for the position of the first vowel in a string represented by the variable testString. As soon as the first vowel is found, the exit repeat command instructs Lingo to leave the repeat loop and go to the statement return i:

```
"on findVowel testString
repeat with i = 1 to testString.char[testString.char.count]
    if ""aeiou"" contains testString.char[i] then exit repeat
end repeat
return i
end"
```

See also repeat while, repeat with

exp()

Syntax *(integerOrFloat).exp*

exp(integerOrFloat)

Description Function; calculates e, the natural logarithm base, to the power specified by *integerOrFloat*.

Example The following statement calculates the value of e to the exponent 5:

```
put (5).exp
-- 148.4132
```

externalEvent

Syntax externalEvent "string"

Description Command; sends a string to the browser that the browser can interpret as a scripting language instruction, allowing a movie playing or a browser to communicate with the HTML page in which it is embedded. The string sent by externalEvent must be in a scripting language supported by the browser.

This command works only for movies in browsers. To enable communication between an applet and a browser, use Java, JavaScript, or VBScript.

Note: The externalEvent command does not produce a return value. There is no immediate way to determine whether the browser handled the event or ignored it. Use on EvalScript within the browser to return a message to the movie.

Example These statements use externalEvent in the LiveConnect scripting environment, which is supported by Netscape 3.x and later.

LiveConnect evaluates the string passed by externalEvent as a function call. JavaScript authors must define and name this function in the HTML header. In the movie, the function name and parameters are defined as a string in externalEvent. Because the parameters must be interpreted by the browser as separate strings, each parameter is surrounded by single quotation marks.

Statements within HTML:

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

Statements within a script in the movie:

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

Example These statements use externalEvent in the ActiveX scripting environment used by Internet Explorer in Windows. ActiveX treats externalEvent as an event and processes this event and its string parameter the same as an onClick event in a button object.

- Statements within HTML:

In the HTML header, define a function to catch the event; this example is in VBScript:

```
Sub  
NameOfShockwaveInstance_onExternalEvent(aParam)  
    'script here  
End Sub
```

Alternatively, define a script for the event:

```
<SCRIPT FOR="NameOfShockwaveInstance"  
EVENT="externalEvent(aParam)"  
LANGUAGE="VBScript">  
    'script here  
</SCRIPT>
```

Within the movie, include the function and any parameters as part of the string for externalEvent:

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

See also on EvalScript

externalParamCount()

Syntax externalParamCount()

Description Function; returns the number of parameters that an HTML <EMBED> or <OBJECT> tag is passing to a Shockwave movie.

This function is valid only for Shockwave movies that are running in a browser. It doesn't work for movies during authoring or for projectors.

Example This handler determines whether an <OBJECT> or <EMBED> tag is passing any external parameters to a Shockwave movie and runs Lingo statements if parameters are being passed:

```
if externalParamCount() > 0 then  
    -- perform some action  
end if
```

See also externalParamName(), externalParamValue()

externalParamName()

Syntax externalParamName(*n*)

Description Function; returns the name of a specific parameter in the list of external parameters from an HTML <EMBED> or <OBJECT> tag. This function is valid only for Shockwave movies that are running in a browser. It cannot be used with Director movies or projectors.

- If *n* is an integer, externalParamName returns the *n*th parameter name in the list.
- If *n* is a string, externalParamName returns *n* if any of the external parameter names matches *n*. The match is not case sensitive. If no matching parameter name is found, externalParamName returns VOID.

Example This statement places the value of a given external parameter in the variable myVariable:

```
if externalParamName ("swURLString") = "swURLString" then  
    myVariable = externalParamValue ("swURLString")  
end if
```

See also externalParamCount(), externalParamValue()

externalParamValue()

Syntax

`externalParamValue(n)`

Description

Function; returns a specific value from the external parameter list in an HTML <EMBED> or <OBJECT> tag. This function is valid only for Shockwave movies that are running in a browser. It can't be used with movies running in the authoring environment or projectors.

- If *n* is an integer, `externalParamValue` returns the *n*th parameter value from the external parameter list.
- If *n* is a string, `externalParamValue` returns the value associated with the first name that matches *n*. The match isn't case sensitive. If no such parameter value exists, `externalParamValue` returns VOID.

This function's behavior in an applet differs from that in other Director movies. In an applet, `externalParamValue` does the following:

- Returns the applet's parameters instead of the <EMBED> tag parameters.
- Accepts only string parameters.
- Returns a zero-length string rather than VOID.

See "Parameters for OBJECT and EMBED tags" and "Parameters accessible from Lingo" on the Director Developers Center Web site.

Example

This statement places the value of an external parameter in the variable `myVariable`:

```
if externalParamName ("swURLString") = "swURLString" then  
    myVariable = externalParamValue ("swURLString")  
end if
```

See also

`externalParamCount()`, `externalParamName()`

extractAlpha()

Syntax

`imageObject.extractAlpha()`

Description

This function copies the alpha channel from the given 32-bit image and returns it as a new image object. The result is an 8-bit grayscale image representing the alpha channel.

This function is useful for downsampling 32-bit images with alpha channels.

Example

This statement places the alpha channel of the image of member 1 into the variable `mainAlpha`.

```
mainAlpha = member(1).image.extractAlpha()  
setAlpha, useAlpha
```

fadeIn()

Syntax `sound(channelNum).fadeIn({milliseconds})`
`fadeIn(sound(channelNum) { , milliseconds })`

Description This function immediately sets the volume of sound channel *channelNum* to zero and then brings it back to the current volume over the given number of milliseconds. The default is 1000 milliseconds (1 second) value is given.
The current pan setting is retained for the the entire fade.

Example This Lingo fades in sound channel 3 over a period of 3 seconds from the beginning of cast member introMusic2.

```
sound(3).play(member("introMusic2"))
sound(3).fadeIn(3000)
```

See also `fadeOut()`, `fadeTo()`, `pan` (sound property), `volume` (sound channel)

fadeOut()

Syntax `sound(channelNum).fadeOut({milliseconds})`
`fadeOut(sound(channelNum) { , milliseconds })`

Description This function gradually reduces the volume of sound channel *channelNum* to zero over the given number of milliseconds, or 1000 milliseconds (1 second) if no value is given.

The current pan setting is retained for the the entire fade.

Example This statement fades out sound channel 3 over a period of 5 seconds:

```
sound(3).fadeOut(5000)
```

See also `fadeIn()`, `fadeTo()`, `pan` (sound property), `volume` (sound channel)

fadeTo()

Syntax

```
sound(channelNum).fadeTo(volume {, milliseconds })  
fadeTo(sound(channelNum), volume {, milliseconds })
```

Description

This function gradually changes the volume of sound channel channelNum to the specified volume over the given number of milliseconds, or 1000 milliseconds (1 second) if no value is given. The range of values for volume is 0-255.

The current pan setting is retained for the the entire fade.

To see an example of fadeTo() used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement changes the volume of sound channel 4 to 150 over a period of 2 seconds. It can be a fade up or a fade down, depending on the original volume of sound channel 4 when the fade begins.

```
sound(4).fadeTo(150, 2000)
```

See also

fadeIn(), fadeOut(), pan (sound property), volume (sound channel)

FALSE

Syntax

FALSE

Description

Constant; applies to an expression that is logically FALSE, such as $2 > 3$. When treated as a number value, FALSE has the numerical value of 0. Conversely, 0 is treated as FALSE.

Example

This statement turns off the soundEnabled property by setting it to FALSE:

```
the soundEnabled = FALSE
```

See also

if, not, TRUE

field

Syntax

```
field whichField
```

Description

Keyword; refers to the field cast member specified by *whichField*.

- When *whichField* is a string, it is used as the cast member name.
- When *whichField* is an integer, it is used as the cast member number.

Character strings and chunk expressions can be read from or placed in the field.

The term field was used in earlier versions of Director and is maintained for backward compatibility. For new movies, use member to refer to field cast members.

Example	This statement places the characters 5 through 10 of the field name entry in the variable myKeyword:
	myKeyword = field("entry").char[5..10]
Example	This statement checks whether the user entered the word <i>desk</i> and, if so, goes to the frame deskBid:
	if member "bid" contains "desk" then go to "deskBid"

See also char...of, item...of, line...of, word...of

fieldOfView

Syntax
 sprite(*whichQTVRSprite*).fieldOfView
 the fieldOfView of sprite *whichQTVRSprite*

Description QTVR sprite property; gives the specified sprite's current field of view in degrees. This property can be tested and set.

fileName (cast property)

Syntax castLib(*whichCast*).fileName
 the fileName of castLib *whichCast*

Description Property; specifies the file name of the specified cast.

- For an external cast, fileName gives the cast's full pathname and file name.
- For an internal cast, the fileName castLib property depends on which internal cast is specified. For the first internal cast library, the fileName castLib property specifies the name of the movie. For remaining internal casts, fileName is an empty string.

The fileName of castLib property accepts URLs as references. However, to use a cast from the Internet and minimize download time, use the downloadNetThing or preloadNetThing command to download the cast's file to a local disk first and then set fileName castLib to the file on the disk.

If a movie sets the file name of an external cast, don't use the Duplicate Cast Members for Faster Loading option in the Projector Options dialog box.

This property can be tested and set for external casts. It can be tested only for internal casts.

Note: Director for Java does not support the downloadNetThing command.

Example	This statement displays the pathname and file name of the Buttons external cast in the Message window:
	<pre>put castLib("Buttons").fileName</pre>
Example	This statement sets the file name of the Buttons external cast to Content.cst:
	<pre>castLib("Buttons").fileName = the moviePath & "Content.cst"</pre>
	The movie then uses the external cast file Content.cst as the Buttons cast.
Example	These statements download an external cast from a URL to the Director application folder and then make that file the external cast named Cast of Thousands:
	<pre>downLoadNetThing("http://www.cbDeMille.com Thousands.cst", the \ applicationPath&"Thousands.cst") castLib("Cast of Thousands").fileName = the applicationPath & "Thousands.cst"</pre>
See also	downloadNetThing , preloadNetThing()

fileName (cast member property)

Syntax	<code>member(<i>whichCastMember</i>).fileName</code>
	the fileName of member <i>whichCastMember</i>
Description	Cast member property; refers to the name of the file assigned to the linked cast member specified by <i>whichCastMember</i> . This property is useful for switching the external linked file assigned to a cast member while a movie plays, similar to the way you can switch cast members. When the linked file is in a different folder than the movie, you must include the file's pathname. You can also make unlinked media linked by setting the filename of those types of members that support linked media. The fileName member property accepts URLs as a reference. However, to use a file from a URL and minimize download time, use the <code>downloadNetThing</code> or <code>preloadNetThing</code> command to download the file to a local disk first and then set <code>fileName</code> member property to the file on the local disk. The Director player for Java doesn't support the <code>downLoadNetThing</code> command, so the player can't download files in the background before assigning a new file to a cast member. Changing the <code>fileName</code> member property in a movie playing as an applet can make the applet wait for the new file to download. This property can be tested and set. After the file name is set, Director uses that file the next time the cast member is used.
Example	This statement links the QuickTime movie “ChairAnimation” to cast member 40: <pre>member(40).fileName = "ChairAnimation"</pre>

These statements download an external file from a URL to the Director application folder and make that file the media for the sound cast member Norma Desmond Speaks:

```
downLoadNetThing("http://www.cbDeMille.com/ Talkies.AIF",the \  
applicationPath&"Talkies.AIF")  
member("Norma Desmond Speaks").fileName = the applicationPath & "Talkies.AIF"  
downloadNetThing, preloadNetThing()
```

See also

fileName (window property)

Syntax

window *whichWindow*.fileName

the fileName of window *whichWindow*

Description

Window property; refers to the file name of the movie assigned to the window specified by *whichWindow*. When the linked file is in a different folder than the movie, you must include the file's pathname.

To be able to play the movie in a window, you must set the fileName window property to the movie's file name.

The fileName of window property accepts URLs as a reference. However, to use a movie file from a URL and minimize the download time, use the downloadNetThing or preloadNetThing command to download the movie file to a local disk first and then set fileName window property to the file on the local disk.

This property can be tested and set.

Example

This statement assigns the file named Control Panel to the window named Tool Box:

```
window("Tool Box").fileName = "Control Panel"
```

Example

This statement displays the file name of the file assigned to the window named Navigator:

```
put window("Navigator").fileName
```

Example

These statements download a movie file from a URL to the Director application folder and then assign that file to the window named My Close Up:

```
downLoadNetThing("http://www.cbDeMille.com/Finale.DIR",the \  
applicationPath&"Finale.DIR")  
window("My Close Up").fileName = the applicationPath&"Finale.DIR"
```

See also

fill()

Syntax

imageObject.fill(left, top, right, bottom, colorObjectOrParameterList)

imageObject.fill(point(x, y), point(x, y), colorObjectOrParameterList)

imageObject.fill(rect, colorObjectOrParameterList)

Description

This function fills a rectangular region with the color *colorObject* in the given image object. You specify the rectangle in any of the three ways shown. The points specified are relative to the upper-left corner of the given image object. The return value is 1 if there is no error, zero if there is an error.

If you provide a *parameterList* instead of a simple *colorObject*, the rectangle is filled with a shape you specify with these parameters:

Property	Description
#shapeType	A symbol value of #oval, #rect, #roundRect, or #line. The default is #line.
#lineSize	The width of the line to use in drawing the shape.
#color	A color object, which determines the fill color of the shape.
#bgColor	A color object, which determines the color of the shape's border.

For best performance, with 8-bit or lower images the *colorObject* should contain an indexed color value. For 16- or 32-bit images, use an RGB color value.

Example

This statement renders the image object in the variable *myImage* completely black:
`myImage.fill(myImage.rect, rgb(0, 0, 0))`

Example

This statement draws a filled oval in the image object *TestImage*. The oval has a green fill and a 5-pixel-wide red border.

```
TestImage.fill(0, 0, 100, 100, [#shapeType: #oval, #lineSize: 5, #color: rgb(0, 255, 0), \#bgColor: rgb(255, 0, 0)])
```

See also

`color()`, `draw()`

fillColor

Syntax

`member(whichCastMember).fillColor`

Description

Vector shape cast member property; the color of the shape's fill specified as an RGB value.

It's possible to use fillColor when the fillMode property of the shape is set to #solid or #gradient, but not if it is set to #none. If the fillMode is #gradient, fillColor specifies the starting color for the gradient. The ending color is specified with endColor.

This property can be tested and set.

To see an example of fillColor used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement sets the fill color of the member Archie to a new RGB value.

```
member("Archie").fillColor = rgb( 24, 15, 153)
```

See also

endColor, fillMode

fillCycles

Syntax

`member(whichCastMember).fillCycles`

Description

Vector shape cast member property; the number of fill cycles in a gradient vector shape's fill, as specified by an integer value from 1 to 7.

This property is valid only when the fillMode property of the shape is set to #gradient.

This property can be tested and set.

To see an example of fillCycles used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement sets the fillCycles of member Archie to 3.

```
member("Archie").fillCycles = 3
```

See also

endColor, fillColor, fillMode

fillDirection

Syntax

member(*whichCastMember*).fillDirection

Description

Vector shape cast member property; specifies the amount in degrees to rotate the fill of the shape.

This property is only valid when the fillMode property of the shape is set to #gradient.

This property can be tested and set.

To see an example of fillDirection used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.

See also

fillMode

filled

Syntax

member(*whichCastMember*).filled

the filled of member *whichCastMember*

Description

Shape cast member property; indicates whether the specified cast member is filled with a pattern (TRUE) or not (FALSE).

Example

The following statements make the shape cast member Target Area a filled shape and assign it the pattern numbered 1, which is a solid color:

```
member("Target Area").filled = TRUE  
member("Target Area").pattern = 1
```

See also

filled

fillMode

Syntax

member(*whichCastMember*).fillMode

Description

Vector shape cast member property; indicates the fill method for the shape, using the following possible values:

- #none—The shape is transparent
- #solid—The shape uses a single fill color
- #gradient—The shape uses a gradient between two colors

This property can be tested and set when the shape is closed; open shapes have no fill.

To see an example of fillMode used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement sets the fillMode of member Archie to gradient.

```
member("Archie").fillMode = #gradient
```

See also endColor, fillColor

fillOffset

Syntax member(*whichCastMember*).fillOffset

Description Vector shape cast member property; specifies the horizontal and vertical amount in pixels (within the defaultRect space) to offset the fill of the shape.

This property is only valid when the fillMode property of the shape is set to #gradient, but can be both tested and set.

To see an example of fillOffset used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement changes the fill offset of the vector shape cast member miette to a horizontal offset of 33 pixels and a vertical offset of 27 pixels:

```
member("miette").fillOffset = point(33, 27)
```

See also defaultRect, fillMode

fillScale

Syntax member(*whichCastMember*).fillScale

Description Vector shape cast member property; specifies the amount to scale the fill of the shape. This property is referred to as “spread” in the vector shape window.

This property is only valid when the fillMode property of the shape is set to #gradient, but can be both tested and set.

To see an example of fillScale used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement sets the fillScale of member Archie to 33.

```
member("Archie").fillScale = 33.00
```

See also fillMode

findEmpty()

Syntax

`findEmpty(member whichCastMember)`

Description

Function; for the current cast only, displays the next empty cast member position or the position after the cast member specified by *whichCastMember*.

Example

This statement finds the first empty cast member on or after cast member 100:

```
put findEmpty(member 100)
```

findLabel()

Syntax

`sprite(whichFlashSprite).findLabel(whichLabelName)`

`findLabel(sprite whichFlashSprite, whichLabelName)`

Description

Function: this function returns the frame number (within the Flash movie) that is associated with the label name requested.

A 0 is returned if the label doesn't exist, or if that portion of the Flash movie has not yet been streamed in.

findPos

Syntax

`list.findPos(property)`

`findPos(list, property)`

Description

List command; identifies the position of the property specified by *property* in the property list specified by *list*.

Using `findPos` with linear lists returns a bogus number if the value of *property* is a number and a script error if the value of *property* is a string.

The `findPos` command performs the same function as the `findPosNear` command, except that `findPos` is VOID when the specified property is not in the list.

Example

This statement identifies the position of the property *c* in the list *Answers*, which consists of [*#a:10, #b:12, #c:15, #d:22*]:

```
Answers.findPos(#c)
```

The result is 3, because *c* is the third property in the list.

See also

`findPosNear`, `sort`

findPosNear

Syntax

sortedList.findPosNear(*valueOrProperty*)

findPosNear(*sortedList*, *valueOrProperty*)

Description

List command; for sorted lists only, identifies the position of the item specified by *valueOrProperty* in the specified sorted list.

The `findPosNear` command works only with sorted lists. Replace *valueOrProperty* with a value for sorted linear lists, and with a property for sorted property lists.

The `findPosNear` command is similar to the `findPos` command, except that when the specified property is not in the list, the `findPosNear` command identifies the position of the value with the most similar alphanumeric name. This command is useful in finding the name that is the closest match in a sorted directory of names.

Example

This statement identifies the position of a property in the sorted list `Answers`, which consists of [`#Nile:2`, `#Pharaoh:4`, `#Raja:0`]:

```
Answers.findPosNear(#Ni)
```

The result is 1, because Ni most closely matches Nile, the first property in the list.

See also

`findPos`

finishIdleLoad

Syntax

`finishIdleLoad` *loadTag*

Description

Command; forces completion of loading for all the cast members that have the specified load tag.

Example

This statement completes the loading of all cast members that have the load tag 20:

```
finishIdleLoad 20
```

See also

`idleHandlerPeriod`, `idleLoadDone()`, `idleLoadMode`, `idleLoadPeriod`, `idleLoadTag`, `idleReadChunkSize`

firstIndent

Syntax

chunkExpression.`firstIndent`

Description

Text cast member property; contains the number of pixels the first indent in *chunkExpression* is offset from the left margin of the *chunkExpression*.

The value is an integer: less than 0 indicates a hanging indent, 0 is no indentation, and greater than 0 is a normal indentation.

This property can be tested and set.

Example	This statement sets the indent of the first line of member Desk to 0 pixels. member("Desk").firstIndent = 0
See also	leftIndent, rightIndent

fixedLineSpace

Syntax	<i>chunkExpression</i> .fixedLineSpace
Description	Text cast member property; controls the height of each line in the <i>chunkExpression</i> portion of the text cast member. The value itself is an integer, indicating height in absolute pixels of each line. The default value is 0, which results in natural height of lines.
Example	This statement sets the height in pixels of each line of member Desk to 24: member("Desk").fixedLineSpace = 24

fixedRate

Syntax	sprite(<i>whichFlashOrGIFSprite</i>).fixedRate the fixedRate of sprite <i>whichFlashOrGIFSprite</i> member(<i>whichFlashOrGIFMember</i>).fixedRate the fixedRate of member <i>whichFlashOrGIFMember</i>
Description	Cast member property and sprite property; controls the frame rate of a Flash movie or animated GIF. The fixedRate property can have integer values. The default value is 15. This property is ignored if the sprite's playbackMode property is anything other than #fixed. This property can be tested and set.
Example	This handler adjusts the frame rate of a Flash movie sprite. As parameters, the handler accepts a sprite reference, an indication of whether to speed up or slow down the Flash movie, and the amount to adjust the speed. on adjustFixedRate whichSprite, adjustType, howMuch case adjustType of #faster: sprite(whichSprite).fixedRate = sprite(whichSprite).fixedRate + howMuch #slower: sprite(whichSprite).fixedRate = sprite(whichSprite).fixedRate - howMuch end case end
See also	playBackMode

fixStageSize

Syntax the fixStageSize

Description Movie property; determines whether the Stage size remains the same when you load a new movie (TRUE, default), or not (FALSE), regardless of the Stage size saved with that movie, or the setting for the centerStage.
The fixStageSize property cannot change the Stage size for a movie that is currently playing.

This property can be tested and set.

Example This statement determines whether the fixStageSize property is turned on. If fixStageSize is FALSE, it sends the playback head to a specified frame.
if the fixStageSize = FALSE then go to frame "proper size"

This statement sets the fixStageSize property to the opposite of its current setting:
the fixStageSize = not the fixStageSize

See also centerStage

flashRect

Syntax member(*whichVectorOrFlashMember*).flashRect

the flashRect of member *whichVectorOrFlashMember*

Description Cast member property; indicates the size of a Flash movie or vector shape cast member as it was originally created. The property values are indicated as a Director rectangle: for example, rect(0,0,32,32).

For linked Flash cast members, the FlashRect member property returns a valid value only when the cast member's header has finished loading into memory.

This property can be tested but not set.

Example This sprite script resizes a Flash movie sprite so that it is equal to the original size of its Flash movie cast member:

```
on beginSprite me
    sprite(me.spriteNum).rect = sprite(me.spriteNum).member.FlashRect
end
```

See also defaultRect, defaultRectMode, state

flashToStage()

Syntax

```
sprite(whichFlashSprite).flashToStage(pointInFlashMovie)
```

```
flashToStage (sprite whichFlashSprite, pointInFlashMovie)
```

Description

Function; returns the coordinate on the Director Stage that corresponds to a specified coordinate in a Flash movie sprite. The function accepts both the Flash channel and movie coordinate and returns the Director Stage coordinate as Director point values: for example, point(300,300).

Flash movie coordinates are measured in Flash movie pixels, which are determined by a movie's original size when it was created in Flash. For the purpose of calculating Flash movie coordinates, point(0,0) of a Flash movie is always at its upper left corner. (The cast member's originPoint property is used only for rotation and scaling, not to calculate movie coordinates.)

The flashToStage and the corresponding stageToFlash functions are helpful for determining which Flash movie coordinate is directly over a Director Stage coordinate. For both Flash and Director, point(0,0) is the upper left corner of the Flash Stage or Director Stage. These coordinates may not match on the Director Stage if a Flash sprite is stretched, scaled, or rotated.

Example

This handler accepts a point value and a sprite reference as a parameter, and it then sets the upper left coordinate of the specified sprite to the specified point within a Flash movie sprite in channel 10:

```
on snapSprite whichFlashPoint, whichSprite
    sprite(whichSprite).loc = sprite(1).FlashToStage(whichFlashPoint)
    updatestage
end
```

See also

stageToFlash()

flipH

Syntax

```
sprite(whichSpriteNumber).flipH
```

the flipH of sprite *whichSpriteNumber*

Description

Sprite property; indicates whether a sprite's image has been flipped horizontally on the Stage (TRUE) or not (FALSE).

The image itself is flipped around its registration point.

This means any rotation or skew remains constant; only the image data itself is flipped.

Example

This statement displays the flipH of sprite 5:

```
put sprite (5).flipH
```

See also

flipV, rotation, skew

flipV

Syntax

```
sprite(whichSpriteNumber).flipV
```

the flipV of sprite *whichSpriteNumber*

Description

Sprite property; indicates whether a sprite's image has been flipped vertically on the Stage (TRUE) or not (FALSE).

The image itself is flipped around its registration point.

This means any rotation or skew remains constant; only the image data itself is flipped.

Example

This statement displays the flipV of sprite 5:

```
sprite (5).flipV = 1
```

See also

flipH, rotation, skew

float()

Syntax

```
(expression).float
```

```
float (expression)
```

Description

Function; converts an expression to a floating-point number. The number of digits that follow the decimal point (for display purposes only, calculations are not affected) is set using the floatPrecision property.

Example

This statement converts the integer 1 to the floating-point number 1:

```
put (1).float
```

-- 1.0

Example

Math operations can be performed using float; if any of the terms is a float value, the entire operation is performed with float:

```
"the floatPrecision = 1
put 2 + 2
-- 4
put (2).float + 2
-- 4.0
the floatPrecision = 4
put 22/7
-- 3
put (22).float / 7
-- 3.1429"
```

See also

floatPrecision, ilk()

floatP()

Syntax `(expression).floatP`

`floatP(expression)`

Description Function; indicates whether the value specified by *expression* is a floating-point number (1 or TRUE) or not (0 or FALSE).

The *P* in floatP stands for *predicate*.

Example This statement tests whether 3.0 is a floating-point number. The Message window displays the number 1, indicating that the statement is TRUE.

```
put (3.0).floatP
```

```
-- 1
```

This statement tests whether 3 is a floating-point number. The Message window displays the number 0, indicating that the statement is FALSE.

```
put (3).floatP
```

```
-- 0
```

See also `float()`, `ilk()`, `integerP()`, `objectP()`, `stringP()`, `symbolP()`

floatPrecision

Syntax the `floatPrecision`

Description Movie property; rounds off the display of floating-point numbers to the number of decimal places specified. The value of `floatPrecision` must be an integer. The maximum value is 15 significant digits; the default value is 4.

The `floatPrecision` property determines only the number of digits used to display floating-point numbers; it does not change the number of digits used to perform calculations.

- If `floatPrecision` is a number from 1 to 15, floating-point numbers display that number of digits after the decimal point. Trailing zeros remain.
- If `floatPrecision` is zero, floating-point numbers are rounded to the nearest integer. No decimal points appear.
- If `floatPrecision` is a negative number, floating-point numbers are rounded to the absolute value for the number of decimal places. Trailing zeros are dropped.

This property can be tested and set.

Example This statement rounds off the square root of 3.0 to three decimal places:

```
the floatPrecision = 3  
x = sqrt(3.0)  
put x  
-- 1.732
```

Example This statement rounds off the square root of 3.0 to eight decimal places:

```
the floatPrecision = 8  
put x  
-- 1.73205081
```

flushInputEvents

Syntax flushInputEvents()

Description This command will flush any waiting mouse or keyboard events from Director's message queue. Generally this is useful when Lingo is in a tight repeat loop and the author wants to make sure any mouse clicks or keyboard presses don't get through. This command operates at runtime only and has no effect during authoring.

Example This Lingo disables mouse and keyboard events while a repeat loop executes:

```
repeat with i = 1 to 10000  
    flushInputEvents()  
    sprite(1).loc = sprite(1).loc + point(1, 1)  
end repeat  
  
mouseDown, mouseUp, keyDown, keyUp
```

font

Syntax member(*whichCastMember*).font
the font of member *whichCastMember*

Description Text and field cast member property; determines the font used to display the specified cast member and requires that the cast member contain characters, if only a space. The parameter *whichCastMember* can be either a cast member name or number.

The Director player for Java doesn't map to other fonts when converting a movie; Java substitutes the default font for any unsupported font. Use only Java's supported fonts as values for the font member property in a movie that plays back as an applet.

Java offers only the following cross-platform fonts:

Java font name	Corresponding Windows font	Corresponding Macintosh font
Helvetica	Arial	Helvetica
TimesRoman	Times New Roman	Times
Courier	Courier-New	Courier
Dialog	MS Sans Serif	Chicago or Charcoal
DialogInput	MS Sans Serif	Geneva
ZapfDingbats	WingDings	Zapf Dingbats
Default	Arial	Helvetica

The font member property can be tested and set.

To see an example of font used in a completed movie, see the Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement sets the variable named *oldFont* to the current font setting for the field cast member *Rokujo Speaks*:

```
oldFont = member("Rokujo Speaks").font
```

See also text; alignment, fontSize, fontStyle, lineHeight (cast member property)

fontSize

Syntax

`member(whichCastMember).fontSize`

the `fontSize` of member *whichCastMember*

Description

Field cast member property; determines the size of the font used to display the specified field cast member and requires that the cast member contain characters, if only a space. The parameter *whichCastMember* can be either a cast member name or number.

This property can be tested and set. When tested, it returns the height of the first line in the field. When set, it affects every line in the field.

To see an example of `fontSize` used in a completed movie, see the Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement sets the variable named `oldSize` to the current `fontSize` of member setting for the field cast member *Rokujo Speaks*:

```
oldSize = member("Rokujo Speaks").fontSize
```

Example

This statement sets the third line of the text cast member *myMenu* to 24 points:

```
member("myMenu").fontSize = 12
```

See also

`text`; `alignment`, `font`, `lineHeight` (cast member property)

fontStyle

Syntax

`member(whichCastMember).fontStyle`

the `fontStyle` of member *whichCastMember*

`member(whichCastMember).char[whichChar].fontStyle`

the `fontStyle` of char *whichChar*

`member(whichCastMember).word[whichWord].fontStyle`

the `fontStyle` of word *whichWord*

`member(whichCastMember).line[whichLine].fontStyle`

the `fontStyle` of line *whichLine*

Description

Cast member property; determines the styles applied to the font used to display the specified field cast member, character, line, word, or other chunk expression and requires that the field cast member contain characters, if only a space.

The value of the property is a string of styles delimited by commas. Lingo uses a font that is a combination of the styles in the string. The available styles are plain, bold, italic, underline, shadow, outline, and extended; on the Macintosh, condensed also is available.

Use the style plain to remove all currently applied styles. The parameter *whichCastMember* can be either a cast member name or number.

For a movie playing back as an applet, plain, bold, and italic are the only valid styles for the fontStyle member property. The Director player for Java doesn't support underline, shadow, outline, extended, or condensed font styles.

This property can be tested and set.

To see an example of fontStyle used in a completed movie, see the Text movie in the LearningLingo Examples folder inside the Director application folder.

Example This statement sets the variable named oldStyle to the current fontStyle setting for the field cast member Rokujo Speaks:

```
oldStyle = member("Rokujo Speaks").fontStyle
```

Example This statement sets the fontStyle member property for the field cast member Poem to bold italic:

```
member("Poem").fontStyle = [#bold, #italic]
```

Example This statement sets the fontStyle property of the third word of the cast member Son's Names to italic:

```
member("Son's Names").word[3].fontStyle = [#italic]
```

Example This statement sets the fontStyle member property of word 1 through word 4 of text member myNote to bold italic:

```
member("myNote").word[1..4].fontStyle = [#bold, #italic]
```

See also text; alignment, font, lineHeight (cast member property), fontSize

foreColor

Syntax `member(castName).foreColor = colorNumber`

set the foreColor of member *castName* to *colorNumber*

sprite *whichSprite*.foreColor

the foreColor of sprite *whichSprite*

Description Cast member property; sets the foreground color of a field cast member.

For a movie that plays back as an applet, specify colors for the foreColor sprite property as the decimal equivalent of the 24-bit hexadecimal values used in an HTML document.

It is not recommended to apply this property to bitmap cast members deeper than 1-bit, as the results are difficult to predict.

To see an example of foreColor used in a completed movie, see the Text movie in the LearningLingo Examples folder inside the Director application folder.

Example The hexadecimal value for pure red, FF0000, is equivalent to 16711680 in decimal numbers. This statement specifies pure red as a cast member's forecolor:

```
member(20).foreColor = 16711680
```

Example	This statement changes the color of the field in cast member 1 to the color in palette entry 250: member(1).foreColor = 250
Example	The following statement sets the variable oldColor to the foreground color of sprite 5: oldColor = sprite(5).foreColor
Example	The following statement makes 36 the number for the foreground color of a random sprite from sprites 11 to 13. sprite(10 + random(3)).foreColor = 36
Example	The following statement sets the foreColor of word 3 of line 2 of text member myDescription to a value of 27: member("myDescription").line[2].word[3].forecolor = 27
See also	backColor, color (sprite property)

forget()

Syntax	timeout("timeoutName").forget() forget(timeout("timeoutName"))
Description	This timeout object function removes the given <i>timeoutObject</i> from the <i>timeoutList</i> , and prevents it from sending further timeout events.
Example	This statement deletes the timeout object named AlarmClock from the <i>timeoutList</i> : timeout("AlarmClock").forget()
See also	timeout(), timeoutHandler, timeoutList, new()

forget window

Syntax	window(<i>whichWindow</i>).forget() forget window <i>whichWindow</i>
Description	Window property; instructs Lingo to close and delete the window specified by <i>whichWindow</i> when it's no longer in use and no other variables refer to it. When a forget window command is given, the window and the movie in a window (MIAW) disappear without calling the on stopMovie, on closeWindow, or on deactivateWindow handlers. If there are many global references to the movie in a window, the window doesn't respond to the forget command.

Example	This statement instructs Lingo to delete the window Control Panel when the movie no longer uses the window:
	<code>window("Control Panel").forget()</code>
See also	close window , open window

frame() (function)

Syntax	<code>the frame</code>
Description	Function; returns the number of the current frame of the movie.
Example	This statement sends the playback head to the frame before the current frame: <code>go to (the frame - 1)</code>
See also	go , label() , marker()

frame (sprite property)

Syntax	<code>sprite(<i>whichFlashSprite</i>).frame</code>
	the frame of sprite <i>whichFlashSprite</i>
Description	Sprite property; controls which frame of the current Flash movie is displayed. The default value is 1. This property can be tested and set.
Example	This frame script checks to see if a Flash movie has finished playing (by checking to see if the current frame is equal to the total number of frames in the movie). If the movie has not finished, the playback head continues to loop in the current frame; when the movie finishes, the playback head continues to the next frame. (This script assumes that the movie was designed to stop on its final frame and that it has not been set for looped playback.) <code>on exitFrame if sprite(5).frame < sprite(5).member.frameCount then go to the frame end if end</code>

frameCount

Syntax member(*whichFlashMember*).frameCount

the frameCount of member *whichFlashMember*

Description Flash cast member property; indicates the number of frames in the Flash movie cast member. The frameCount member property can have integer values.

This property can be tested but not set.

Example This sprite script displays, in the Message window, the channel number and the number of frames in a Flash movie.

```
"property spriteNum  
  
on beginSprite me  
    put ""The Flash movie in channel"" && spriteNum && has"" &&  
    sprite(spriteNum).member.frameCount && ""frames.""  
end"
```

frameLabel

Syntax the frameLabel

Description Frame property; identifies the label assigned to the current frame. When the current frame has no label, the value of the frameLabel property is 0.

This property can be tested at any time. It can be set during a Score generation session.

Example This statement checks the label of the current frame. In this case, the current frameLabel value is Start:

```
put the frameLabel  
-- "Start"
```

See also labelList

framePalette

Syntax the framePalette

Description Frame property; identifies the cast member number of the palette used in the current frame, which is either the current palette or the palette set in the current frame.

Because the browser controls the palette for the entire Web page, the Director player for Java always uses the browser's palette. For the most reliable color when authoring a movie for playback as a Director player for Java, use the default palette for the authoring system. When you want exact control over colors, use Shockwave instead of Java.

This property can be tested. It can also be set during a Score generation session.

Example This statement checks the palette used in the current frame. In this case, the palette is cast member 45.

```
put the framePalette  
-- 45
```

Example This statement makes palette cast member 45 the palette for the current frame:

```
the framePalette = 45
```

See also puppetPalette

frameRate

Syntax member(*whichCastMember*).frameRate

the frameRate of member *whichCastMember*

Description Cast member property; specifies the playback frame rate for the specified digital video, or Flash movie cast member.

The possible values for the frame rate of a digital video member correspond to the radio buttons for selecting digital video playback options.

- When the frameRate member property is between 1 and 255, the digital video movie plays every frame at that frame rate. The frameRate member property cannot be greater than 255.
- When the frameRate member property is set to -1 or 0, the digital video movie plays every frame at its normal rate. This allows the video to sync to its soundtrack. When the frameRate is set to any value other than -1 or 0, the digital video soundtrack will not play.
- When the frameRate member property is set to -2, the digital video movie plays every frame as fast as possible.

For Flash movie cast members, the property indicates the frame rate of the movie created in Flash.

This property can be tested but not set.

Example This statement sets the frame rate of the QuickTime digital video cast member Rotating Chair to 30 frames per second:

```
member("Rotating Chair").frameRate = 30
```

Example This statement instructs the QuickTime digital video cast member Rotating Chair to play every frame as fast as possible:

```
member("Rotating Chair").frameRate = -2
```

Example This sprite script checks to see if the sprite's cast member was originally created in Flash with a frame rate of less than 15 frames per second. If the movie's frame rate is slower than 15 frames per second, the script sets the playBackMode property for the sprite so it can be set to another rate. The script then sets the sprite's fixedRate property to 15 frames per second.

```
property spriteNum  
on beginSprite me  
    if sprite(spriteNum).member.frameRate < 15 then  
        sprite(spriteNum).playBackMode = #fixed  
        sprite(spriteNum).fixedRate = 15  
    end if  
end
```

See also [fixedRate](#), [movieRate](#), [movieTime](#), [playBackMode](#)

frameReady()

Syntax

```
frameReady(frameN)  
frameReady(frameN, frameZ)  
frameReady()  
frameReady(sprite whichFlashSprite, frameNumber)
```

Description

Function; for a Flash movie, determines whether a streaming movie is ready for display. If enough of a sprite has streamed into memory to render the frame (integer for frame number, string for label) specified in the frameNumber parameter, this function is TRUE; otherwise it is FALSE. For a Director movie, this function determines whether all the cast members for *frameN* (the number of the frame) are downloaded from the Internet and available locally.

This function is useful only when streaming a movie, range of frames, cast, or linked cast member. To activate streaming, set the Movie:Playback properties in the Modify menu to Use Media as Available or Show Placeholders.

For Director movies, projectors, and Shockwave movies:

- `frameReady(frameN)`—Determines whether the cast members for `frameN` are downloaded.
- `frameReady(frameN, frameZ)`—Determines whether the cast members for `frameN` through `frameZ` are downloaded.
- `frameReady()`—Determines if cast member used in any frame of the Score are downloaded.

For a demonstration of the `frameReady` function used with a Director movie, see the sample movie “Streaming Shockwave” in Director Help.

This function can be tested but not set.

Example

This statement determines whether the cast members for frame 20 are downloaded and ready to be viewed:

```
on exitFrame
    if frameReady(20) then
        -- go to frame 20 if all the required
        --castmembers are locally available
        go to frame 20
    else
        -- resume animating loop while background
        --is streaming
        got to frame 1
    end if
end
```

Example

This frame script checks to see if frame 25 of a Flash movie sprite in channel 5 can be rendered. If it can't, the script keeps the playback head looping in the current frame of the Director movie. When frame 25 can be rendered, the script starts the movie and lets the playback head proceed to the next frame of the Director movie.

```
on exitFrame
    if the frameReady(sprite 5, 25) = FALSE then
        go to the frame
    else
        play sprite 5
    end if
end
```

See also

[mediaReady](#)

frameScript

Syntax the frameScript

Description Frame property; contains the unique cast member number of the frame script assigned to the current frame.

The frameScript property can be tested. During a Score recording session, you can also assign a frame script to the current frame by setting the frameScript property.

Example This statement displays the number of the script assigned to the current frame. In this case, the script number is 25.

```
put the frameScript  
-- 25
```

Example This statement makes the script cast member Button responses the frame script for the current frame:

```
the frameScript = member "Button responses"
```

frameSound1

Syntax the frameSound1

Description Frame property; determines the number of the cast member assigned to the first sound channel in the current frame.

This property can be tested and set. This property can also be set during a Score recording session.

Example As part of a Score recording session, this statement assigns the sound cast member Jazz to the first sound channel:

```
the frameSound1 = member("Jazz").number
```

frameSound2

Syntax the frameSound2

Description Frame property; determines the number of the cast member assigned to the second sound channel for the current frame.

This property can be tested and set. This property can also be set during a Score recording session.

Example As part of a Score recording session, this statement assigns the sound cast member Jazz to the second sound channel:

```
the frameSound2 = member("Jazz").number
```

framesToHMS()

Syntax

`framesToHMS(frames, tempo, dropFrame, fractionalSeconds)`

Description

Function; converts the specified number of frames to their equivalent length in hours, minutes, and seconds. This function is useful for predicting the actual playtime of a movie or controlling a video playback device.

- *frames*—Integer expression that specifies the number of frames.
- *tempo*—Integer expression that specifies the tempo in frames per second.
- *dropFrame*—Compensates for the color NTSC frame rate, which is not exactly 30 frames per second and is meaningful only if FPS is set to 30 frames per second. Normally, this argument is set to FALSE.
- *fractionalSeconds*—Determines whether the residual frames are converted to the nearest hundredth of a second (TRUE) or returned as an integer number of frames (FALSE).

The resulting string uses the form `SHH:MM:SS.FFD`, where:

s	A character is used if the time is less than zero, or a space if the time is greater than or equal to zero.
HH	Hours.
MM	Minutes.
SS	Seconds.
FF	Indicates a fraction of a second if <i>fractionalSeconds</i> is TRUE or frames if <i>fractionalSeconds</i> is FALSE.
D	A "d" is used if <i>dropFrame</i> is TRUE, or a space if <i>dropFrame</i> is FALSE.

Example

This statement converts a 2710-frame, 30 frame-per-second movie. The *dropFrame* and *fractionalSeconds* arguments are both turned off:

```
put framesToHMS(2710, 30, FALSE, FALSE)  
-- "00:01:30.10"
```

See also

[HMSToFrames\(\)](#)

frameTempo

Syntax	the frameTempo
Description	Frame property; indicates the tempo assigned to the current frame. This property can be tested. It can be set during a Score recording session.
Example	This statement checks the tempo used in the current frame. In this case, the tempo is 15 frames per second. <pre>put the frameTempo ~ 15</pre>
See also	puppetTempo

frameTransition

Syntax	the frameTransition
Description	Frame property; specifies the number of the transition cast member assigned to the current frame. This property can be set during a Score recording session to specify transitions.
Example	When used in a Score recording session, this statement makes the cast member Fog the transition for the frame that Lingo is currently recording: <pre>set the frameTransition to member "Fog"</pre>

freeBlock()

Syntax	the freeBlock
Description	Function; indicates the size of the largest free contiguous block of memory, in bytes. A kilobyte (K) is 1024 bytes. A megabyte (MB) is 1024 kilobytes. Loading a cast member requires a free block at least as large as the cast member.
Example	This statement determines whether the largest contiguous free block is smaller than 10K and displays an alert if it is: <pre>if (the freeBlock < (10 * 1024)) then alert "Not enough memory!"</pre>
See also	freeBytes(), memorySize, ramNeeded(), size

freeBytes()

Syntax the freeBytes

Description Function; indicates the total number of bytes of free memory, which may not be contiguous. A kilobyte (K) is 1024 bytes. A megabyte (MB) is 1024 kilobytes.

This function differs from freeBlock in that it reports all free memory, not just contiguous memory.

On the Macintosh, selecting Use System Temporary Memory in the Director General Preferences or in a projector's Options dialog box tells the freeBytes function to return all the free memory that is available to the application. This amount equals the application's allocation shown in its Get Info dialog box and the Largest Unused Block value in the About This Macintosh dialog box.

Example This statement checks whether more than 200K of memory is available and plays a color movie if it is:

```
if (the freeBytes > (200 * 1024)) then play movie "colorMovie"
```

See also freeBlock(), memorySize, objectP(), ramNeeded(), size

frontWindow

Syntax the frontWindow

Description System property; indicates which movie in a window (MIAW) is currently frontmost on the screen. When the Stage is frontmost, front window is the Stage. When a media editor or floating palette is frontmost, frontWindow returns VOID.

This property can be tested but not set.

Example This statement determines whether the window "Music" is currently the frontmost window and, if it is, brings the window "Try This" to the front:

```
if the frontWindow = "Music" then window("Try This").moveToFront
```

See also activeWindow, on activateWindow, on deactivateWindow, moveToFront

getaProp

Syntax

```
propertyList.propertyName  
getaProp(list, item)  
list[listPosition]  
propertyList [ #propertyName ]  
propertyList [ "ipropertyName" ]
```

Description List command; for linear and property lists, identifies the value associated with the item specified by *item*, *listPosition*, or *propertyName* in the list specified by *list*.

- When the list is a linear list, replace *item* with the number for an item's position in a list as shown by *listPosition*. The result is the value at that position.
- When the list is a property list, replace *item* with a property in the list as in *propertyName*. The result is the value associated with the property.

The getaProp command returns VOID when the specified value is not in the list.

When used with linear lists, the getaProp command has the same function as the getAt command.

Example This statement identifies the value associated with the property #joe in the property list ages, which consists of [#john:10, #joe:12, #cheryl:15, #barbara:22]:
put getaProp(ages, #joe)

The result is 12, because this is the value associated with the property #joe.

Example The same result can be achieved using bracket access on the same list:

```
put ages[#joe]
```

The result is again 12.

Example If you want the value at a certain position in the list, you can also use bracket access. To get the third value in the list, associated with the third property, use this syntax:

```
put ages[3]  
-- 15
```

Note: Unlike the getAProp command where VOID is returned when a property doesn't exist, a script error will occur if the property doesn't exist when using bracket access.

See also getAt, getOne(), getProp(), setaProp, setAt

getAt

Syntax

```
getAt(list, position)
```

list [*position*]

Description

List command; identifies the item in the position specified by *position* in the specified list. If the list contains fewer elements than the specified position, a script error occurs.

The getAt command works with linear and property lists. This command has the same function as the getProp command for linear lists.

This command is useful for extracting a list from within another list, such as the deskTopRectList.

Example

This statement causes the Message window to display the third item in the answers list, which consists of [10, 12, 15, 22]:

```
put getAt(answers, 3)
```

-- 15

Example

The same result can be returned using bracket access:

```
put answers[3]
```

-- 15

Example

This example extracts the first entry in a list containing two entries that specify name, department, and employee number information. Then the second element of the newly extracted list is returned, identifying the department in which the first person in the list is employed. The format of the list is [[“Dennis”, “consulting”, 510], [“Sherry”, “Distribution”, 973]], and the list is called employeeInfoList.

```
firstPerson = getAt(employeeInfoList, 1)
put firstPerson
-- ["Dennis", "consulting", 510]
firstPersonDept = getAt(firstPerson, 2)
put firstPersonDept
-- "consulting"
```

Example

It's also possible to nest getAt commands without assigning values to variables in intermediate steps. This format can be more difficult to read and write, but less verbose.

```
firstPersonDept = getAt(getAt(employeeInfoList, 1), 2)
put firstPersonDept
-- "consulting"
```

Example You can also use the bracket list access:

```
firstPerson = employeeInfoList[1]
put firstPerson
-- ["Dennis", "consulting", 510]
firstPersonDept = firstPerson[2]
put firstPersonDept
-- "consulting"
```

Example As with getAt, brackets can be nested:

```
firstPersonDept = employeeInfoList[1][2]
```

See also [getaProp](#), [setaProp](#), [setAt](#)

on getBehaviorDescription

Syntax on getBehaviorDescription

```
    statement(s)
```

```
end
```

Description System message and event handler; contains Lingo that returns the string that appears in a behavior's description pane in the Behavior Inspector when the behavior is selected.

The description string is optional.

Director sends the getBehaviorDescription message to the behaviors attached to a sprite when the Behavior Inspector opens. Place the on getBehaviorDescription handler within a behavior.

The handler can contain embedded Return characters for formatting multiple-line descriptions.

Example This statement displays “Vertical Multiline textField Scrollbar” in the description pane.

```
on getBehaviorDescription
    return "Vertical Multiline textField Scrollbar"
end
```

See also [on getPropertyDescriptionList](#), [on getBehaviorTooltip](#), [on runPropertyDialog](#)

on getBehaviorTooltip

Syntax

```
on getBehaviorTooltip  
    statement(s)  
end
```

Description

System message and event handler; contains Lingo that returns the string that appears in a tooltip for a script in the Library palette.

Director sends the getBehaviorTooltip message to the script when the cursor stops over it in the Library palette. Place the on getBehaviorTooltip handler within the behavior.

The use of the handler is optional. If no handler is supplied, the cast member name appears in the tooltip.

The handler can contain embedded Return characters for formatting multiple-line descriptions.

Example

This statement displays “Jigsaw puzzle piece” in the description pane.

```
on getBehaviorTooltip  
    return "Jigsaw puzzle piece"  
end
```

See also

on getPropertyDescriptionList, on getBehaviorDescription, on runPropertyDialog

getError()

Syntax

```
member(whichSWAmember).getError()  
getError(member whichSWAmember)  
member(whichFlashmember).getError()  
getError(member whichFlashmember)
```

Description

Function; for Shockwave Audio (SWA) or Flash cast members, indicates whether an error occurred as the cast member streamed into memory and returns a value.

Shockwave Audio cast members have the following possible `getError()` integer values and corresponding `getErrorString()` messages:

<code>getError()</code> value	<code>getErrorString()</code> message
0	OK
1	memory
2	network
3	playback device
99	other

Flash movie cast members have the following possible `getError` values:

- FALSE—No error occurred.
- #memory—There is not enough memory to load the cast member.
- #fileNotFound—The file containing the cast member's assets could not be found.
- #network—A network error prevented the cast member from loading.
- #fileFormat—The file was found, but it appears to be of the wrong type, or an error occurred while reading the file.
- #other—Some other error occurred.

When an error occurs as a cast member streams into memory, Director sets the cast member's state property to -1. Use the `getError` function to determine what type of error occurred.

Example

This handler uses `getError` to determine whether an error involving the Shockwave Audio cast member Norma Desmond Speaks occurred and displays the appropriate error string in a field if it did:

```
on exitFrame
    if member("Norma Desmond Speaks").getError() <> 0 then
        member("Display Error Name").text = member("Norma Desmond \
Speaks").getErrorString()
        end if
    end
```

Example This handler checks to see whether an error occurred for a Flash cast member named Dali, which was streaming into memory. If an error occurred, and it was a memory error, the script uses the unloadCast command to try to free some memory; it then branches the playback head to a frame in the Director movie named Artists, where the Flash movie sprite first appears, so Director can again try to load and play the Flash movie. If something other than an out-of-memory error occurred, the script goes to a frame named Sorry, which explains that the requested Flash movie can't be played.

```
on CheckFlashStatus
    errorCheck = member("Dali").getError()
    if errorCheck <> 0 then
        if errorCheck = #memory then
            member("Dali").clearError()
            unloadCast
            go to frame ("Artists")
        else
            go to frame ("Sorry")
        end if
    end if
end
```

See also [clearError](#), [getErrorString\(\)](#), [state](#)

getErrorString()

Syntax

```
member(whichCastMember).getErrorString()  
getErrorString(member whichCastMember)
```

Description

Function; for Shockwave Audio (SWA) cast members, returns the error message string that corresponds to the error value returned by the `getError()` function.

Possible `getError()` integer values and corresponding `getErrorString()` messages are:

<code>getError()</code> value	<code>getErrorString()</code> message
0	OK
1	memory
2	network
3	playback device
99	other

Example This handler uses `getError()` to determine whether an error occurred for Shockwave Audio cast member Norma Desmond Speaks, and if so, uses `getErrorString()` to obtain the error message and assign it to a field cast member:

```
on exitFrame
    if member("Norma Desmond Speaks").getError() <> 0 then
        member("Display Error Name").text = member("Norma Desmond \
Speaks").getErrorString()
    end if
end
```

See also [getError\(\)](#)

getFlashProperty()

Syntax `sprite(spriteNum).getFlashProperty("targetName", #property)`

Description This function allows Lingo to invoke the Flash action script function `getProperty()` on the given Flash sprite. This Flash action script function is used to get the value of properties of movie clips or levels within a Flash movie. This is similar to testing sprite properties within Director.

The `targetName` is the name of the movie clip or level whose property you want to get within the given Flash sprite.

The `#property` is the name of the property to get. These movie clip properties are gettable: `#posX`, `#posY`, `#scaleX`, `#scaleY`, `#visible`, `#rotate`, `#alpha`, `#name`, `#width`, `#height`, `#target`, `#url`, `#dropTarget`, `#totalFrames`, `#currentFrame`, and `#lastframeLoaded`.

To get a global property of the Flash sprite, pass an empty string as the `targetName`. These global Flash properties are gettable: `#focusRect` and `#spriteSoundBufferTime`.

See the Flash documentation for descriptions of these properties.

Example This statement gets the value of the `#rotate` property of the movie clip Star in the Flash member in sprite 3.

```
sprite(3).setFlashProperty("Star", #rotate)
setFlashProperty()
```

getFrameLabel

Syntax

```
sprite(whichFlashSprite).getFrameLabel(whichFlashFrameNumber)  
getFrameLabel(sprite whichFlashSprite, whichFlashFrameNumber)
```

Description

Function; returns the frame label within a Flash movie that is associated with the frame number requested. If the label doesn't exist, or that portion of the Flash movie has not yet been streamed in, this function returns an empty string.

Example

The following handler looks to see if the marker on frame 15 of the Flash movie playing in sprite 1 is called "Lions". If it is, the Director movie navigates to frame "Lions". If it isn't, the Director movie stays in the current frame and the Flash movie continues to play.

```
on exitFrame  
    if sprite(1).getFrameLabel(15) = "Lions" then  
        go "Lions"  
    else  
        go the frame  
    end if  
end
```

getHotSpotRect()

Syntax

```
sprite(whichQTVRSprite).getHotSpotRect(hotSpotID)  
getHotSpotRect(whichQTVRSprite, hotSpotID)
```

Description

QuickTime VR function; returns an approximate bounding rectangle for the hot spot specified by *hotSpotId*. If the hot spot doesn't exist or isn't visible on the Stage, this function returns rect(0, 0, 0, 0). If the hot spot is partially visible, this function returns the bounding rectangle for the visible portion.

getLast()

Syntax

```
list.getLast()  
getLast(list)
```

Description

List function; identifies the last value in a linear or property list specified by *list*.

Example

This statement identifies the last item, 22, in the list Answers, which consists of [10, 12, 15, 22]:

```
put Answers.getLast()
```

Example

This statement identifies the last item, 850, in the list Bids, which consists of [#Gee:750, #Kayne:600, #Ohashi:850]:

```
put Bids.getLast()
```

getLatestNetID

Syntax getLatestNetID

Description This function returns an identifier for the last network operation that started.

The identifier returned by getLatestNetID can be used as a parameter in the netDone, netError, and netAbort functions to identify the last network operation.

Note: This function is included for backward compatibility. It is recommended that you use the network ID returned from a net lingo function rather than getLatestNetID. However, if you use getLatestNetID, use it immediately after issuing the netLingo command.

Example This script assigns the network ID of a getNetText operation to the field cast member Result so results of that operation can be accessed later.

```
on startOperation
    global gNetID
    getNetText("url")
    set gNetID = getLatestNetID()
end

on checkOperation
    global gNetID
    if netDone(gNetID) then
        put netTextResult into member "Result"
    end if
end

netAbort command; netDone() and netError() functions
```

getNetText()

Syntax getNetText(*URL* {, *serverOSString*} {, *characterSet*})

getNetText(*URL*, *propertyList* {, *serverOSString*} {, *characterSet*})

Description Function; starts the retrieval of text from a file usually on an HTTP or FTP server, or initiates a CGI query.

The first syntax shown starts the text retrieval. You can submit HTTP CGI queries this way and must properly encode them in the URL. The second syntax includes a property list and submits a CGI query, providing the proper URL encoding.

Use the optional parameter *propertyList* to take a property list for CGI queries. The property list is URL encoded and the URL sent is (*urlstring* & "?" & *encodedproplist*).

Use the optional parameter *serverOSString* to encode any return characters in *propertylist*. The value defaults to UNIX but may be set to Win or Mac and translates any carriage returns in the *propertylist* argument into those used on the server. For most applications, this setting is unnecessary because line breaks are usually not used in form responses.

The optional parameter *characterSet* applies only if the user is running Director on a shift-JIS (Japanese) system. Possible character set settings are JIS, EUC, ASCII, and AUTO. Lingo converts the retrieved data from shift-JIS to the named character set. Using the AUTO setting, character set tries to determine what character set the retrieved text is in and translate it to the character set on the local machine. The default setting is ASCII.

For a movie that plays back as an applet, the `getNetText` command retrieves text only from the domain that contains the applet. This behavior differs from Shockwave and is necessary due to Java's security model.

Use `netDone` to find out when the `getNetText` operation is complete, and `netError` to find out if the operation was successful. Use `netTextResult` to return the text retrieved by `getNetText`.

The function works with relative URLs.

To see an example of `getNextText()` used in a completed movie, see the Forms and Post movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This script retrieves text from the URL `http://BigServer.com/sample.txt` and updates the field cast member the mouse is on when it's clicked:

```
property spriteNum
property theNetID

on mouseUp me
    theNetID = getNetText ("http://BigServer.com/sample.txt")
end

on exitFrame me
    if netDone(theNetID) then
        sprite(spriteNum).member.text = netTextResult(theNetID)
    end if
end
```

Example This example retrieves the results of a CGI query:

```
getNetText("http://www.yourserver.com/cgi-bin/query.cgi?name=Bill")
```

Example This is the same as the previous example, but it uses a property list to submit a CGI query, and does the URL encoding for you.

```
getNetText("http://www.yourserver.com/cgi-bin/query.cgi", [#name:"Bill"])
```

See also `netDone()`, `netError()` , `netTextResult()`

getNthFileNameInFolder()

Syntax

```
getNthFileNameInFolder(folderPath, fileNumber)
```

Description

Function; returns a file name from the directory folder based on the specified path and number within the folder. To be found by the getNthFileNameInFolder function, Director movies must be set to visible in the folder structure. (On the Macintosh, other types of files are found whether they are visible or invisible.) If this function returns an empty string, you have specified a number greater than the number of files in the folder.

The getNthFileNameInFolder function doesn't work with URLs.

To specify other folder names, use the @ pathname operator or the full path defined in the format for the specific platform on which the movie is running. For example:

- In Windows, use a directory path such as C:\Director\Movies.
- On the Macintosh, use a pathname such as HardDisk:Director:Movies. To look for files on the Macintosh desktop, use the path HardDisk:Desktop Folder
- This function is not available in Shockwave.

Example

The following handler returns a list of file names in the folder on the current path. To call the function, use parentheses, as in put currentFolder().

```
on currentFolder
    fileList = [ ]
    repeat with i = 1 to 100
        n = getNthFileNameInFolder(the moviePath, i)
        if n = EMPTY then exit repeat
        fileList.append(n)
    end repeat
    return fileList
end currentFolder
```

See also

[@ \(pathname\)](#)

getOne()

Syntax

```
list.getOne(value)
```

```
getOne(list, value)
```

Description

List function; identifies the position (linear list) or property (property list) associated with the value specified by *value* in the list specified by *list*.

For values contained in the list more than once, only the first occurrence is displayed. The getOne command returns the result 0 when the specified value is not in the list.

When used with linear lists, the `getOne` command performs the same functions as the `getPos` command.

Example This statement identifies the position of the value 12 in the linear list `Answers`, which consists of [10, 12, 15, 22]:

```
put Answers.getOne(12)
```

The result is 2, because 12 is the second value in the list.

Example This statement identifies the property associated with the value 12 in the property list `Answers`, which consists of [#a:10, #b:12, #c:15, #d:22]:

```
put Answers.getOne(12)
```

The result is #b, which is the property associated with the value 12.

See also `getPos()`

getPixel()

Syntax `imageObject.getPixel(x, y {, #integer})`

```
imageObject.getPixel(point(x, y) {, #integer})
```

Description This function returns the color value of the pixel at the specified point in the given image object. This value is normally returned as an indexed or RGB color object, depending on the bit depth of the image.

If you include the optional parameter value `#integer`, however, it's returned as a raw number. If you're setting a lot pixels to the color of another pixel, it's faster to set them as raw numbers. Raw integer color values are also useful because they contain alpha layer information as well as color when the image is 32-bit. The alpha channel information can be extracted from the raw integer by dividing the integer by 2^{8+8+8} .

`GetPixel()` returns 0 if the given pixel is outside the specified image object.

Example These statements get the color of the pixel at point (90, 20) in member `Happy` and set sprite 2 to that color.

```
myColor=member("Happy").image.getPixel(90, 20)  
sprite(2).color=myColor
```

Example This statement sets the variable `alpha` to the alpha channel value of the point (25, 33) in the 32-bit image object `myImage`.

```
alpha = myImage.getPixel(25, 33, #integer) / power(2, 8+8+8)
```

See also `depth, color(), setPixel(), power()`

getPlaylist()

Syntax

```
sound(channelNum).getPlaylist()  
getPlaylist(sound(channelNum))
```

Description

This function returns a copy of the list of queued sounds for *soundObject*. This list does not include the currently playing sound.

The list of queued sounds may not be edited directly. You must use *setPlayList()*.

The playlist is a linear list of property lists. Each property list corresponds to one queued sound cast member. Each queued sound may specify these properties:

Property	Description
#member	The sound cast member to play. This property will always be present; all others are optional.
#startTime	The time within the sound at which playback begins, in milliseconds. See <i>startTime</i> .
#endTime	The time within the sound at which playback ends, in milliseconds. See <i>endTime</i> .
#loopCount	The number of times to play a portion of the sound. See <i>loopCount</i> .
#loopStartTime	The time within the sound at which a loop begins, in milliseconds. See <i>loopStartTime</i> .
#loopEndTime	The time within the sound at which a loop ends, in milliseconds. See <i>loopEndTime</i> .
#preloadTime	The amount of the sound to buffer before playback, in milliseconds. See <i>preloadTime</i> .

Example

This handler queues two sounds in sound channel 2, starts playing them, and then displays the playList in the message window. Notice that the playlist only includes the second sound queued, because the first sound is already playing.

```
on playMusic  
    sound(2).queue([#member:member("chimes")])  
    sound(2).queue([#member:member("introMusic"), #startTime:3000,\n        #endTime:10000, #loopCount:5,#loopStartTime:8000, #loopEndTime:8900])  
    sound(2).play()  
    put sound(2).getPlaylist()  
end  
-- [[#member: (member 12 of castLib 2), #startTime: 3000, #endTime: 10000,\n#loopCount: 5, #loopStartTime: 8000, #loopEndTime: 8900]]
```

See also

endTime, *loopCount*, *loopEndTime*, *loopStartTime*, *member* (keyword), *preLoadTime*, *queue()*, *setPlaylist()*, *startTime*

getPos()

Syntax	list.getPos(<i>value</i>) getPos(<i>list</i> , <i>value</i>)
Description	List function; identifies the position of the value specified by <i>value</i> in the list specified by <i>list</i> . When the specified value is not in the list, the getPos command returns the value 0. For values contained in the list more than once, only the first occurrence is displayed. This command performs the same function as the getOne command when used for linear lists.
Example	This statement identifies the position of the value 12 in the list Answers, which consists of [#a:10, #b:12, #c:15, #d:22]: put Answers.getPos(12) The result is 2, because 12 is the second value in the list.
See also	getOne()

getPref()

Syntax	getPref(<i>prefFileName</i>)
Description	Function; retrieves the content of the specified file. When you use this function, replace <i>prefFileName</i> with the name of a file created by the setPref function. If no such file exists, getPref returns VOID. The file name used for <i>prefFileName</i> must be a valid file name only, not a full path; Director supplies the path. The path to the file is handled by Director. The only valid file extensions for <i>prefFileName</i> are .txt and .htm; any other extension is rejected. Do not use this command to access read-only or locked media.
Note:	In a browser, data written by setPref is not private. Any Shockwave movie can read this information and upload it to a server. Confidential information should not be stored using setPref.
	To see an example of getPref() used in a completed movie, see the Read and Write Text movie in the Learning\Lingo Examples folder inside the Director application folder.
Example	This handler retrieves the content of the file Test and then assigns the file's text to the field Total Score: on mouseUp theText = getPref("Test") member("Total Score").text = theText end
See also	setPref

getProp()

Syntax

getProp(*list*, *property*)

list.property

Description

Property list function; identifies the value associated with the property specified by *property* in the property list specified by *list*.

Almost identical to the getaProp command, the getProp command displays an error message if the specified property is not in the list or if you specify a linear list.

Example

This statement identifies the value associated with the property #c in the property list Answers, which consists of [#a:10, #b:12, #c:15, #d:22]:

```
getProp(Answers, #c)
```

The result is 15, because 15 is the value associated with #c.

See also

getOne()

getPropAt()

Syntax

list.getPropAt(*index*)

getPropAt(*list*, *index*)

Description

Property list function; for property lists only, identifies the property name associated with the position specified by *index* in the property list specified by *list*. If the specified item isn't in the list, or if you use getPropAt() with a linear list, a script error occurs.

Example

This statement displays the second property in the given list:

```
put Answers.getPropAt(2)  
-- #b
```

The result is 20, which is the value associated with #b.

on getPropertyDescriptionList

Syntax

```
on getPropertyDescriptionList  
    statement(s)  
end
```

Description

System message and event handler; contains Lingo that generates a list of definitions and labels for the parameters that appear in a behavior's Parameters dialog box.

Place the on getPropertyDescriptionList handler within a behavior script. Behaviors that don't contain an on getPropertyDescriptionList handler don't appear in the Parameters dialog box and can't be edited from Director's interface.

The on getPropertyDescriptionList message is sent when any action that causes the Behavior Inspector to open occurs: either when the user drags a behavior to the Score or the user double-clicks a behavior in the Behavior Inspector.

The #default, #format, and #comment settings are mandatory for each parameter. The following are possible values for these settings:

#default	The parameter's initial setting.
#format	#integer #float #string #symbol #member #bitmap #filmloop #field #palette #picture #sound #button #shape #movie #digitalvideo #script #richtext #ole #transition #xtra #frame #marker #ink #boolean
#comment	A descriptive string that appears to the left of the parameter's editable field in the Parameters dialog box.
#range	A range of possible values that can be assigned to a property. The range is specified as a linear list with several values or as a minimum and maximum in the form of a property list: [#min: minValue, #max: maxValue].

Example

This handler defines a behavior's parameters that appear in the Parameters dialog box. Each statement that begins with addProp adds a parameter to the list named description. Each element added to the list defines a property and the property's #default, #format, and #comment values:

```
on getPropertyDescriptionList  
    description = [:]  
    addProp description, #dynamic, [#default:1, #format:#boolean, #comment:"Dynamic"]  
    addProp description, #fieldNum, [#default:1, #format:#integer, \  
#comment:"Scroll which sprite:"]  
    addProp description, #extentSprite, [#default:1, #format:#integer, \  
#comment: "Extend Sprite:"]  
    addProp description, #proportional, [#default:1, #format:#boolean, \  
#comment: "Proportional:"]  
    return description  
end
```

See also

[addProp](#), [on getBehaviorDescription](#), [on runPropertyDialog](#)

getStreamStatus()

Syntax

```
getStreamStatus(netID)  
getStreamStatus(URLString)
```

Description

Function; returns a property list matching the format used for the globally available tellStreamStatus function that can be used with callbacks to sprites or objects. The list contains the following strings:

#URL	String containing the URL location used to start the network operation.
#state	String consisting of Connecting, Started, InProgress, Complete, "Error", or "NoInformation" (this last string is for the condition when either the net ID is so old that the status information has been dropped or the URL specified in URLString was not found in the cache).
#bytesSoFar	Number of bytes retrieved from the network so far.
#bytesTotal	Total number of bytes in the stream, if known. The value may be 0 if the HTTP server does not include the content length in the MIME header.
#error	String containing "" (EMPTY) if the download is not complete, OK if it completed successfully, or an error code if the download ended with an error.

For example, you can start a network operation with getNetText() and track its progress with getStreamStatus().

Example

This statement displays in the message window the current status of a download begun with getNetText() and the resulting net ID placed in the variable netID:

```
put getStreamStatus(netID)  
-- [#URL: "www.macromedia.com", #state: "InProgress", #bytesSoFar: 250, \  
#bytesTotal: 50000, #error: EMPTY]
```

See also

on streamStatus, tellStreamStatus()

getVariable()

Syntax

```
getVariable(sprite flashSpriteNum, "variableName")
```

Description

This function returns the current value of the given variable from the given Flash sprite. Flash variables were introduced in Flash version 4.

Example

This statement returns the value of the variable currentURL from the Flash cast member in sprite 3 and displays it in the Message window:

```
put getVariable(sprite 3, "currentURL")  
-- "http://www.macromedia.com/software/flash/"
```

See also

hitTest(), setVariable()

global

Syntax

```
global variable1 {, variable2} {, variable3}...
```

Description

Keyword; defines a variable as a global variable so that other handlers or movies can share it.

Every handler that examines or changes the content of a global variable must use the global keyword to identify the variable as global. Otherwise, the handler treats the variable as a local variable, even if it is declared to be global in another handler.

Note: To ensure that global variables are available throughout a movie, declare and initialize them in the prepareMovie handler. Then, if you leave and return to the movie from another movie, your global variables will be reset to the initial values unless you first check to see that they aren't already set.

A global variable can be declared in any handler or script. Its value can be used by any other handlers or scripts that also declare the variable as global. If the script changes the variable's value, the new value is available to every other handler that treats the variable as global.

A global variable is available in any script or movie, regardless of where it is first declared; it is not automatically cleared when you navigate to another frame, movie, or window.

Any variables manipulated in the Message window are automatically global, even though they are not explicitly declared as such.

Shockwave movies playing on the Internet cannot access global variables within other movies, even movies playing on the same HTML page. The only way movies can share global variables is if an embedded movie navigates to another movie and replaces itself through either goToNetMovie or go movie.

Example

This example sets the global variable StartingPoint to an initial value of 1 if it doesn't already contain a value. This allows navigation to and from the movie without loss of stored data.

```
global gStartingPoint  
on prepareMovie  
    if voidP(gStartingPoint) then gStartingPoint = 1  
end
```

See also

[showGlobals](#), [property](#), [gotoNetMovie](#)

globals

Syntax the globals

Description System property; this property contains a special property list of all current global variables with a value other than VOID. Each global variable is a property in the list, with the associated paired value.

You can use the following list operations on globals:

- count()—Returns the number of entries in the list.
- getPropAt(*n*)—Returns the name of the *n*th entry.
- getProp(*x*)—Returns the value of an entry with the specified name.
- getAProp(*x*)—Returns the value of an entry with the specified name.

Note: The globals property automatically contains the property #version, which is the version of Director running. This means there will always be at least one entry in the list, even if no global variables have been declared yet.

This property differs from showGlobals in that the globals can be used in contexts other than the Message window. To display the globals in the Message window, use showGlobals.

See also showGlobals, clearGlobals

go

Syntax go {to} {frame} *whichFrame*

go {to} movie *whichMovie*

go {to} {frame} *whichFrame* of movie *whichMovie*

Description Command; causes the playback head to branch to the frame specified by *whichFrame* in the movie specified by *whichMovie*. The expression *whichFrame* can be a marker label or an integer frame number. The expression *whichMovie* must specify a movie file. (If the movie is in another folder, *whichMovie* must specify the path.)

The phrase go loop tells the playback head to loop to the previous marker and is a convenient means of keeping the playback head in the same section of the movie while Lingo remains active and avoids the use of go to the frame in a frame that has a transition which would slow the movie and overwhelm the processor.

It is best to refer to marker labels instead of frame numbers; editing a movie can cause frame numbers to change. Using marker labels also makes it easier to read scripts.

The go to movie command loads frame 1 of the movie. If the command is called from within a handler, the handler in which it is placed continues executing. To suspend the handler while playing the movie, use the play command, which may be followed by a subsequent play done to return.

When you specify a movie to play, specify its path if the movie is in a different folder, but to prevent a potential load failure, don't include the movie's .dir, .dxe or .dcr file extension.

To more efficiently go to a movie at a URL, use the downloadNetThing command to download the movie file to a local disk first and then use the go to movie command to go to that movie on the local disk.

The following are reset when a movie is loaded: beepOn and constraint properties; keyDownScript, mouseDownScript, and mouseUpScript; cursor and immediate sprite properties; cursor and puppetSprite commands; and custom menus. However, the timeoutScript is not reset when loading a movie.

Example This statement sends the playback head to the marker named start:

```
go to "start"
```

Example This statement sends the playback head to the marker named Memory in the movie named Noh Tale to Tell:

```
go frame("Memory") of movie("Noh Tale to Tell")
```

Example This handler tells the movie to loop in the current frame. This handler is useful for making the movie wait in a frame while it plays so the movie can respond to events.

```
on exitFrame  
    go the frame  
end
```

See also downloadNetThing, gotoNetMovie, label(), marker(), pathName (movie property), play, play done

go loop

Syntax go loop

Description Command; sends the playback head to the previous marker in the movie, either one marker back from the current frame if the current frame does not have a marker, or to the current frame if the current frame has a marker.

Note: This command is equivalent to marker(0) in previous versions of Director.

If no markers are to the left of the playback head, the playback head branches to:

- The next marker to the right if the current frame does not have a marker.
- The current frame if the current frame has a marker.
- Frame 1 if the movie contains no markers.

The go loop command is equivalent to the statement go to the marker(0) used in earlier versions of Lingo.

Example This statement causes the movie to loop between the current frame and the previous marker:

```
go loop
```

See also go, go next, go previous

go next

Syntax go next

Description Command; sends the playback head to the next marker in the movie. If no markers are to the right of the playback head, the playback head goes to the last marker in the movie or to frame 1 if there are no markers in the movie.

The go next command is equivalent to the statement go marker(1) that was used in earlier versions of Lingo.

Example This statement sends the playback head to the next marker in the movie:

```
go next
```

See also go, go loop, go previous

go previous

Syntax go previous

Description Command; sends the playback head to the previous marker in the movie. This marker is two markers back from the current frame if the current frame does not have a marker or one marker back from the current frame if the current frame has a marker.

Note: This command is equivalent to marker(-1) in previous versions of Director.

If no markers are to the left of the playback head, the playback head branches to:

- The next marker to the right if the current frame does not have a marker
- The current frame if the current frame has a marker
- Frame 1 if the movie contains no markers

Example This statement sends the playback head to the previous marker in the movie:

```
go previous
```

See also go, go loop, go next

goToFrame

Syntax

```
sprite(whichFlashSprite).goToFrame(frameNumber)  
goToFrame(sprite whichFlashSprite, frameNumber)  
sprite(whichFlashSprite).goToFrame(labelNameString)  
goToFrame(sprite whichFlashSprite, labelNameString)
```

Description

Command; plays a Flash movie sprite beginning at the frame identified by the *frameNumber* parameter. You can identify the frame by either an integer indicating a frame number or by a string indicating a label name. Using the *goToFrame* command has the same effect as setting a Flash movie sprite's frame property.

Example

This handler branches to different points within a Flash movie in channel 5. It accepts a parameter that indicates which frame to go to.

```
on Navigate whereTo  
    sprite(5).goToFrame(whereTo)  
end
```

gotoNetMovie

Syntax

```
gotoNetMovie URL  
gotoNetMovie (URL)
```

Description

Command; retrieves and plays a new Shockwave movie from an HTTP or FTP server. The current movie continues to run until the new movie is available.

Only URLs are supported as valid parameters. The URL can specify either a file name or a marker within a movie. Relative URLs work if the movie is on an Internet server, but you must include the extension with the file name.

When performing testing on a local disk or network, media must be located in a directory named dswmedia.

If a *gotoNetMovie* operation is in progress and you issue a second *gotoNetMovie* command before the first is finished, the second command cancels the first.

Example

In this statement, the URL indicates a Director file name:

```
gotoNetMovie "http://www.yourserver.com/movies/movie1.dcr"
```

Example

In this statement, the URL indicates a marker within a file name:

```
gotoNetMovie "http://www.yourserver.com/movies/buttons.dcr#Contents"
```

Example

In this statement, *gotoNetMovie* is used as a function. The function returns the network ID for the operation.

```
myNetID = gotoNetMovie ("http://www.yourserver.com/movies/buttons.dcr#Contents")
```

gotoNetPage

Syntax

```
gotoNetPage "URL", {"targetName"}
```

Description

Command; opens a Shockwave movie or another MIME file in the browser.

Only URLs are supported as valid parameters. Relative URLs work if the movie is on an HTTP or FTP server.

The *targetName* argument is an optional HTML parameter that identifies the frame or window in which the page is loaded.

- If *targetName* is a window or frame in the browser, gotoNetPage replaces the contents of that window or frame.
- If *targetName* isn't a frame or window that is currently open, goToNetPage opens a new window.
- If *targetName* is not included, gotoNetPage replaces the current page, wherever it is located.

In the authoring environment, the gotoNetPage command launches the preferred browser if it is enabled. In projectors, this command tries to launch the preferred browser set with the Network Preferences dialog box or browserName command. If neither has been used to set the preferred browser, the goToNetPage command attempts to find a browser on the computer.

Example

This script loads the file Newpage.html into the frame or window named frwin. If a window or frame in the current window called frwin exists, that window or frame is used. If the window frwin doesn't exist, a new window named frwin is created.

```
on keyDown
    gotoNetPage "Newpage.html", "frwin"
end
```

Example

This handler opens a new window regardless of what window the browser currently has open:

```
on mouseUp
    goToNetPage "Todays_News.html", "_new"
end
```

See also

browserName(), netDone()

gradientType

Syntax

```
member(whichCastMember).gradientType
```

Description

Vector shape cast member property; specifies the actual gradient used in the cast member's fill.

Possible values are #linear or #radial. The gradientType is only valid when the fillMode is set to #gradient.

This property can be tested and set.

Example

This handler toggles between linear and radial gradients in cast member "backdrop".

```
on mouseUp me
    if member("backdrop").gradientType = #radial then
        member("backdrop").gradientType = #linear
    else
        member("backdrop").gradientType = #radial
    end if
end
```

See also

fillMode

halt

Syntax

```
halt
```

Description

Command; exits the current handler and any handler that called it and stops the movie during authoring or quits the projector during run time from a projector.

Example

This statement checks whether the amount of free memory is less than 50K and, if it is, exits all handlers that called it and then stops the movie:

```
if the freeBytes < 50*1024 then halt
```

See also

abort, exit, pass, quit

handler()

Syntax

```
scriptObject.handler(#handlerSymbol)
```

Description

This function returns TRUE if the given *scriptObject* contains a handler whose name is *#handlerSymbol*, and FALSE if it does not. The script object must be a parent script, a child object, or a behavior.

Example

This Lingo code invokes a handler on an object only if that handler exists:

```
if spiderObject.handler(#pounce) = TRUE then
    spiderObject.pounce()
end if
```

See also

handlers(), new(), rawNew(), script

handlers()

Syntax `scriptObject.handlers()`

Description This function returns a linear list of the handlers in the given `scriptObject`. Each handler name is presented as a symbol in the list. This function is useful for debugging movies.

Note that you cannot get the handlers of a script cast member directly. You have to get them via the `script` property of the member.

Example This statement displays the list of handlers in the child object RedCar in the Message window:

```
put RedCar.handlers()  
-- [#accelerate, #turn, #stop]
```

Example This statement displays the list of handlers in the parent script member CarParentScript in the Message window:

```
put member("CarParentScript").script.handlers()  
-- [#accelerate, #turn, #stop]  
  
handler(), script
```

height

Syntax `member(whichCastMember).height`

the height of member `whichCastMember`

`imageObject.height`

`sprite(whichSprite).height`

the height of sprite `whichSprite`

Description Cast member, image object and sprite property; for vector shape, Flash, animated GIF, bitmap, and shape cast members, determines the height, in pixels, of the cast member displayed on the Stage.

- For cast members, works with bitmap and shape cast members only. This property can be tested but not set.
- For image objects, this property can be tested but not set.
- For sprites: setting the sprite's height automatically sets the sprite's stretch property to TRUE. For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet. This property can be tested and set.

Example This statement assigns the height of cast member Headline to the variable vHeight:
`vHeight = member("Headline").height`

Example This statement sets the height of sprite 10 to 26 pixels:

```
sprite(10).height = 26
```

Example This statement assigns the height of sprite (i + 1) to the variable vHeight:

```
vHeight = sprite(i + 1).height
```

See also height, rect (sprite), width

hilite (command)

Syntax fieldChunkExpression.hilite()

hilite *fieldChunkExpression*

Description Command; highlights (selects) in the field sprite the specified chunk, which can be any chunk that Lingo lets you define, such as a character, word, or line. On the Macintosh, the highlight color is set in the Color control panel.

Example This statement highlights the fourth word in the field cast member Comments, which contains the string Thought for the Day:

```
member("Comments").word[4].hilite()
```

Example This statement causes highlighted text within the sprite for field myRecipes to be displayed without highlighting:

```
myLineCount = member("myRecipes").line.count  
member("myRecipes").line[myLineCount + 1].hilite()
```

See also char...of, item...of, line...of, word...of; delete, mouseChar, mouseLine, mouseWord; field; selection() (function); selEnd, selStart

hilite (cast member property)

Syntax member(*whichCastMember*).hilite

the hilite of member *whichCastMember*

Description Cast member property; determines whether a check box or radio button created with the button tool is selected (TRUE) or not (FALSE, default).

If *whichCastMember* is a string, it specifies the cast member name. If it is an integer, *whichCastMember* specifies the cast member number.

This property can be tested and set.

Example This statement checks whether the button named Sound on is selected and, if it is, turns sound channel 1 all the way up:

```
if member("Sound on").hilite = TRUE then sound(1).volume = 255
```

Example	This statement uses Lingo to select the button cast member powerSwitch by setting the hilite member property for the cast member to TRUE: member("powerSwitch").hilite = TRUE
See also	checkBoxAccess, checkBoxType

hitTest()

Syntax
 sprite(*whichFlashSprite*).hitTest(*point*)
 hitTest(*sprite whichFlashSprite*, *point*)

Description	Function; indicates which part of a Flash movie is directly over a specific Director Stage location. The Director Stage location is expressed as a Director point value: for example, point(100,50). The hitTest function returns these values:
	<ul style="list-style-type: none"> • #background—The specified Stage location falls within the background of the Flash movie sprite. • #normal—The specified Stage location falls within a filled object. • #button—The specified Stage location falls within the active area of a button. • #editText—The specified Stage location falls within a Flash editable text field.

Example	This frame script checks to see if the mouse is currently located over a button in a Flash movie sprite in channel 5 and, if it is, the script sets a text field used to display a status message:
	<pre>on exitFrame if sprite(5).hitTest(the mouseLoc) = #button then member("Message Line").text = "Click here to play the movie." updateStage else member("Message Line").text = "" end if go the frame end</pre>

HMStoFrames()

Syntax

HMStoFrames(*hms*, *tempo*, *dropFrame*, *fractionalSeconds*)

Description

Function; converts movies measured in hours, minutes, and seconds to the equivalent number of frames or converts a number of hours, minutes, and seconds into time if you set the *tempo* argument to 1 (1 frame = 1 second).

- *hms*—String expression that specifies the time in the form sHH:MM:SS.FFD, where:

s A character is used if the time is less than zero, or a space if the time is greater than or equal to zero.

HH Hours.

MM Minutes.

SS Seconds.

FF Indicates a fraction of a second if *fractionalSeconds* is TRUE or frames if *fractionalSeconds* is FALSE.

D A d is used if *dropFrame* is TRUE, or a space if *dropFrame* is FALSE.

- *tempo*—Specifies the tempo in frames per second.
- *dropFrame*—Logical expression that determines whether the frame is a drop frame (TRUE) or not (FALSE). If the string *hms* ends in a *d*, the time is treated as a drop frame, regardless of the value of *dropFrame*.
- *fractionalSeconds*—Logical expression that determines the meaning of the numbers after the seconds; they can be either fractional seconds rounded to the nearest hundredth of a second (TRUE) or the number of residual frames (FALSE).

Example

This statement determines the number of frames in a 1-minute, 30.1-second movie when the tempo is 30 frames per second. Neither the *dropFrame* nor *fractionalSeconds* arguments is used.

```
put HMStoFrames(" 00:01:30.10 ", 30, FALSE, FALSE)
```

```
-- 2710
```

Example

This statement converts 600 seconds into minutes:

```
>> put framesToHMS(600, 1,0,0)
>> -- " 00:10:00.00 "
```

Example

This statement converts an hour and a half into seconds:

```
>> put HMStoFrames("1:30:00", 1,0,0)
>> -- 5400
```

See also

framesToHMS()

hold

Syntax `sprite(whichFlashSprite).hitTest(point)`

`hold sprite whichFlashSprite`

Description Flash command; stops a Flash movie sprite that is playing in the current frame, but any audio continues to play.

Example This frame script holds the Flash movie sprites playing in channels 5 through 10 while allowing the audio for these channels to continue playing:

```
on enterFrame
    repeat with i = 5 to 10
        sprite(i).hold()
    end repeat
end
```

See also `movieRate`, `pause` (movie playback)

hotSpot

Syntax `member(whichCursorCastMember).hotspot`

 the hotspot of member `whichCursorCastMember`

Description Cursor cast member property; specifies the horizontal and vertical point location of the pixel that represents the hotspot within the animated color cursor cast member `whichCursorCastMember`. Director uses this point to track the cursor's position on the screen (for example, when it returns the values for the Lingo functions `mouseH` and `mouseV`) and to determine where a rollover (signaled by the Lingo message `mouseEnter`) occurs.

The upper left corner of a cursor is `point(0,0)`, which is the default `hotSpot` value. Trying to set a point outside the bounds of the cursor produces an error. For example, setting the hotspot of a 16-by-16-pixel cursor to `point(16,16)` produces an error (because the starting point is `0,0`, not `1,1`).

This property can be tested and set.

Example This handler sets the hotspot of a 32-by-32-pixel cursor (whose cast member number is stored in the variable `cursorNum`) to the middle of the cursor:

```
on startMovie
    member(cursorNum).hotSpot = point(16,16)
end
```

hotSpotEnterCallback

Syntax

`sprite(whichQTVRSprite).hotSpotEnterCallback`

the hotSpotEnterCallback of sprite *whichQTVRSprite*

Description

QuickTime VR sprite property; contains the name of the handler that runs when the cursor enters a QuickTime VR hot spot that is visible on the Stage. The QuickTime VR sprite receives the message first. The message has two arguments: the me parameter and the ID of the hot spot that the cursor entered.

To clear the callback, set this property to 0.

To avoid a performance penalty, set a callback property only when necessary.

This property can be tested and set.

See also

`hotSpotExitCallback`, `nodeEnterCallback`, `nodeExitCallback`, `triggerCallback`

hotSpotExitCallback

Syntax

`sprite(whichQTVRSprite).hotSpotExitCallback`

the hotSpotExitCallback of sprite *whichQTVRSprite*

Description

QuickTime VR sprite property; contains the name of the handler that runs when the cursor leaves a QuickTime VR hot spot that is visible on the Stage. The QuickTime VR sprite receives the message first. The message has two arguments: the me parameter and the ID of the hot spot that the cursor entered.

To clear the callback, set this property to 0.

To avoid a performance penalty, set a callback property only when necessary.

This property can be tested and set.

See also

`nodeEnterCallback`, `nodeExitCallback`, `triggerCallback`

HTML

Syntax

`member(whichMember).HTML`

Description

Cast member property; accesses text and tags that control the layout of the text within an HTML-formatted text cast member.

This property can be tested and set.

Example

This statement displays in the message window the HTML formatting information embedded in the text cast member Home Page:

```
put member("Home Page").HTML
```

See also

`importFileInto`, `RTF`

hyperlink

Syntax

chunkExpression.hyperlink

Description

Text cast member property; returns the hyperlink string for the specified chunk expression in the text cast member.

This property can be both tested and set.

When retrieving this property, the link containing the first character of *chunkExpression* is used.

Hyperlinks may not overlap. Setting a hyperlink over an existing link, even partially over it), replaces the initial link with the new one.

Setting a hyperlink to an empty string removes it.

Example

This handler creates a hyperlink in the first word of text cast member "MacroLink". The text is linked to Macromedia's web site.

```
on startMovie
    member("MacroLink").word[1].hyperlink = \
        "http://www.macromedia.com"
    end
```

See also

hyperlinkRange, hyperlinkState

on hyperlinkClicked

Syntax

```
on hyperlinkClicked me, data, range
    statement(s)
end
```

Description

System message and event handler; used to determine when a hyperlink is actually clicked.

This event handler has the following parameters:

- *me*—Used in a behavior to identify the sprite instance
- *data*—The hyperlink data itself; the string entered in the Text Inspector when editing the text cast member
- *range*—The character range of the hyperlink in the text (It's possible to get the text of the range itself by using the syntax member Ref.char[range[1]..range[2]])

This handler should be attached to a sprite as a behavior script. Avoid placing this handler in a cast member script.

Example This behavior shows a link examining the hyperlink that was clicked, jump to a URL if needed, then output the text of the link itself to the message window:

```
property spriteNum  
on hyperlinkClicked me, data, range  
    if data starts "http://" then  
        goToNetPage(data)  
    end if  
    currentMember = sprite(spriteNum).member  
    anchorString = currentMember.char[range[1]..range[2]]  
    put "The hyperlink on"&&anchorString&&"was just clicked."  
end
```

hyperlinkRange

Syntax *chunkExpression.hyperlinkRange*

Description Text cast member property; returns the range of the hyperlink that contains the first character of the chunk expression.

This property can be tested but not set.

Like `hyperLink` and `hyperLinkState`, the returned range of the link contains the first character of *chunkExpression*.

See also `hyperlink`, `hyperlinkState`

hyperlinks

Syntax *chunkExpression.hyperlinks*

Description Text cast member property; returns a linear list containing all the hyperlink ranges for the specified chunk of a text cast member. Each range is given as a linear list with two elements, one for the starting character of the link and one for the ending character.

Example This statement returns all the links for the text cast member `Glossary` to the message window:

```
put member("Glossary").hyperlinks  
-- [[3, 8], [10, 16], [41, 54]]
```

hyperlinkState

Syntax *textChunk.hyperlinkState*

Description Text cast member property; contains the current state of the hyperlink. Possible values for the state are: `#normal`, `#active`, and `#visited`.

This property can be tested and set.

Like `hyperLink` and `hyperLinkRange`, the returned range of the link contains the first character of `chunkExpression`.

Example This handler checks to see if the hyperlink clicked is a web address. If it is, the state of the hyperlink text state is set to `#visited`, and the movie branches to the web address.

```
property spriteNum  
on hyperlinkClicked me, data, range  
    if data starts "http://" then  
        currentMember = sprite(spriteNum).member  
        currentMember.word[4].hyperlinkState = #visited  
        goToNetPage(data)  
    end if  
end
```

See also `hyperlink`, `hyperlinkRange`

on idle

Syntax

```
on idle  
    statement(s)  
end
```

Description System message and event handler; contains statements that run whenever the movie has no other events to handle and is a useful location for Lingo statements that you want to execute as frequently as possible, such as statements that update values in global variables and displays current movie conditions.

Because statements in `on idle` handlers run frequently, it is good practice to avoid placing Lingo that takes a long time to process in an `on idle` handler.

It is often preferable to put `on idle` handlers in frame scripts instead of movie scripts to take advantage of the `on idle` handler only when appropriate.

Director can load cast members from an internal or external cast during an `idle` event. However, it cannot load linked cast members during an `idle` event.

The `idle` message is only sent to frame scripts and movie scripts.

Example This handler updates the time being displayed in the movie whenever there are no other events to handle:

```
on idle  
    member("Time").text = the short time  
end idle
```

See also `idleHandlerPeriod`

idleHandlerPeriod

Syntax the idleHandlerPeriod

Description Movie property; determines the maximum number of ticks that passes until the movie sends an `idle` message. The default value is 1, which tells the movie to send `idle` handler messages no more than 60 times per second.

When the playback head enters a frame, Director starts a timer, repaints the appropriate sprites on the Stage, and issues an `enterFrame` event. Then, if the amount of time set for the tempo has elapsed, Director generates an `exitFrame` event and goes to the next specified frame; if the amount of time set for this frame hasn't elapsed, Director waits until the time runs out and periodically generates an `idle` message. The amount of time between `idle` events is determined by `idleHandlerPeriod`.

Possible settings for `idleHandlerPeriod` are:

- 0—As many `idle` events as possible
- 1—Up to 60 per second
- 2—Up to 30 per second
- 3—Up to 20 per second
- n —Up to $60/n$ per second

The number of `idle` events per frame also depends on the frame rate of the movie and other activity, including whether Lingo scripts are executing. If the tempo is 60 frames per second (fps) and the `idleHandlerPeriod` value is 1, one `idle` event per frame occurs. If the tempo is 20 fps, three `idle` events per frame occur. Idle time results from Director doesn't have a current task to perform and cannot generate any events.

In contrast, if the `idleHandlerPeriod` property is set to 0 and the tempo is very low, thousands of `idle` events can be generated.

The default value for this property is 1, which differs from previous versions in which it defaulted to 0.

Example The following statement causes the movie to send an `idle` message a maximum of once per second:

```
the idleHandlerPeriod = 60
```

See also `idleHandlerPeriod`, `idleLoadDone()`, `idleLoadMode`, `idleLoadTag`, `idleReadChunkSize`

idleLoadDone()

Syntax `idleLoadDone(loadTag)`

Description Function; reports whether all cast members with the given tag have been loaded (TRUE) or are still waiting to be loaded (FALSE).

Example This statement checks whether all cast members whose load tag is 20 have been loaded and then plays the movie Kiosk if they are:

```
if idleLoadDone(20) then play movie("on idle")
```

See also `idleHandlerPeriod`, `idleLoadMode`, `idleLoadPeriod`, `idleLoadTag`, `idleReadChunkSize`

idleLoadMode

Syntax the `idleLoadMode`

Description System property; determines when the `preLoad` and `preLoadMember` commands try to load cast members during idle periods according to the following values:

- 0—Does not perform idle loading
- 1—Performs idle loading when there is free time between frames
- 2—Performs idle loading during idle events
- 3—Performs idle loading as frequently as possible

The `idleLoadMode` system property performs no function and works only in conjunction with the `preLoad` and `preLoadMember` commands.

Cast members that were loaded using idle loading remain compressed until the movie uses them. When the movie plays back, it may have noticeable pauses while it decompresses the cast members.

Example This statement causes the movie to try as frequently as possible to load cast members designated for preloading by the `preLoad` and `preLoadMember` commands:
`the idleLoadMode = 3`

See also `idleHandlerPeriod`, `idleLoadDone()`, `idleLoadPeriod`, `idleLoadTag`, `idleReadChunkSize`

idleLoadPeriod

Syntax	the idleLoadPeriod
Description	System property; determines the number of ticks that Director waits before trying to load cast members waiting to be loaded. The default value for idleLoadPeriod is 0, which instructs Director to service the load queue as frequently as possible.
Example	This statement instructs Director to try loading every 1/2 second (30 ticks) any cast members waiting to be loaded: set the idleLoadPeriod = 30
See also	idleLoadDone(), idleLoadMode, idleLoadTag, idleReadChunkSize

idleLoadTag

Syntax	the idleLoadTag
Description	System property; identifies or tags with a number the cast members that have been queued for loading when the computer is idle. The idleLoadTag is a convenience that identifies the cast members in a group that you want to preload. The property can be tested and set using any number that you choose.
Example	This statement makes the number 10 the idle load tag: the idleLoadTag = 10
See also	idleHandlerPeriod, idleLoadDone(), idleLoadMode, idleLoadPeriod, idleReadChunkSize

idleReadChunkSize

Syntax	the idleReadChunkSize
Description	System property; determines the maximum number of bytes that Director can load when it attempts to load cast members from the load queue. The default value is 32K. This property can be tested and set.
Example	This statement specifies that 500K is the maximum number of bytes that Director can load in one attempt at loading cast members in the load queue: the idleReadChunkSize = 500 * 1024
See also	idleHandlerPeriod, idleLoadDone(), idleLoadMode, idleLoadPeriod, idleLoadTag

if

Syntax

```
if logicalExpression then statement
  if logicalExpression then statement
    else statement
    end if
  if logicalExpression then
    statement(s)
  end if
  if logicalExpression then
    statement(s)
  else
    statement(s)
  end if
  if logicalExpression1 then
    statement(s)
  else if logicalExpression2 then
    statement(s)
  else if logicalExpression3 then
    statement(s)
  end if
  if logicalExpression1 then
    statement(s)
  else logicalExpression2
end if
```

Description

Keyword; if...then structure that evaluates the logical expression specified by *logicalExpression*.

- If the condition is TRUE, Lingo executes the statement(s) that follow then.
- If the condition is FALSE, Lingo executes the statement(s) following else. If no statements follow else, Lingo exits the if...then structure.
- All parts of the condition must be evaluated; execution does not stop at the first condition that is met or not met. Thus, faster code may be created by nesting if...then statements on separate lines instead of placing them all on the first line to be evaluated.

When the condition is a property, Lingo automatically checks whether the property is TRUE. You don't need to explicitly add the phrase = TRUE after the property.

The else portion of the statement is optional. To use more than one *then-statement* or *else-statement*, you must end with the form end if.

The else portion always corresponds to the previous if statement; thus, sometimes you must include an else nothing statement to associate an else keyword with the proper if keyword.

Note: A quick way to determine in the script window if a script is paired properly is to press Tab. This forces Director to check the open Script window and show the indentation for the contents. Any mismatches will be immediately apparent.

Example This statement checks whether the carriage return was pressed and then continues if it was:

```
if the key = RETURN then go the frame + 1
```

Example This handler checks whether the Command and Q keys were pressed simultaneously and, if so, executes the subsequent statements:

```
on keyDown
```

```
    if (the commandDown) and (the key = "q") then
        cleanUp
        quit
    end if
end keyDown
```

Example Compare the following two constructions and the performance results. The first construction evaluates both conditions, and so must determine the time measurement, which may take a while. The second construction evaluates the first condition; the second condition is checked only if the first condition is TRUE.

```
spriteUnderCursor = rollOver()
if (spriteUnderCursor > 25) AND MeasureTimeSinceStarted() then
    alert "You found the hidden treasure!"
end if
```

The alternate, and faster construction would be:

```
spriteUnderCursor = rollOver()
if (spriteUnderCursor > 25) then
    if MeasureTimeSinceStarted() then
        alert "You found the hidden treasure!"
    end if
end if
```

See also case

ilk()

Syntax `ilk(object)`

`ilk(object, type)`

Description Function; indicates the type of an object.

- The syntax `ilk(object)` returns a value indicating the type of an object. If the object is a list, `ilk(object)` returns `#list`; if the object is a property list, `ilk(object)` returns `#propList`.
- The syntax `ilk(object, type)` compares the object represented by `object` to the specified type. If the object is of the specified type, the `ilk()` function returns TRUE. If the object is not of the specified type, the `ilk()` function returns FALSE.

The following table shows the return value for each type of object recognized by `ilk()`:

Type of Object	<code>ilk(Object)</code> returns	<code>ilk(Object, Type)</code> returns 1 only if Type =	Example
linear list	#list	#list or #linearlist	<code>ilk ([1,2,3])</code>
property list	#proplist	#list or #proplist	<code>ilk ([#his: 1234, #hers: 7890])</code>
integer	#integer	#integer or #number	<code>ilk (333)</code>
float	#float	#float or #number	<code>ilk (123.456)</code>
string	#string	#string	<code>ilk ("asdf")</code>
rect	#rect	#rect or #list	<code>ilk (sprite(1).rect)</code>
point	#point	#point or #list	<code>ilk (sprite(1).loc)</code>
color	#color	#color	<code>ilk (sprite(1).color)</code>
date	#date	#date	<code>ilk (the systemdate)</code>
symbol	#symbol	#symbol	<code>ilk (#hello)</code>
void	#void	#void	<code>ilk (void)</code>
picture	#picture	#picture	<code>ilk (member (2).picture)</code>
parent script instance	#instance	#object	<code>ilk (new (script "blahblah"))</code>
xtra instance	#instance	#object	<code>ilk (new (xtra "fileio"))</code>
member	#member	#member	<code>ilk (member 1)</code>
xtra	#xtra	#object	<code>ilk (xtra "fileio")</code>
script	#script	#object	<code>ilk (script "blahblah")</code>
castlib	#castlib	#object	<code>ilk (castlib 1)</code>
sprite	#sprite	#object	<code>ilk (sprite 1)</code>
sound	#sound	#object	<code>ilk (sound "yaddayadda")</code>
window	#window	#object	<code>ilk (the stage)</code>
media	#media	#object	<code>ilk (member (2).media)</code>

Example The following `ilk` statement identifies the type of the object named `Bids`.

```
Bids = [:]
put ilk( Bids )
-- #proplist
```

Example The following `ilk` statement tests whether the variable `Total` is a list and displays the result in the Message window:

```
Total = 2+2  
put ilk( Total, #list )  
-- 0
```

In this case, since the variable `Total` is not a list, the Message window displays 0, which is the numeric equivalent of FALSE.

Example The following example tests a variable named `myVariable` and verifies that it is a date object before displaying it in the Message window:

```
myVariable = the systemDate  
if ilk(myVariable, #date) then put myVariable  
-- date( 1999, 2, 19 )
```

image

Syntax `whichMember.image`
`(the stage).image`
`window(windowName).image`

Description This property refers to the image object of a bitmap or text cast member, of the Stage, or of a window. You can get or set a cast member's image, but you can only get the image of the Stage or a window.

Setting a cast member's image property immediately changes the contents of the member. However, when you get the image of a member or window, Director creates a reference to the image of the specified member or window. If you make changes to the image, the contents of the cast member or window change immediately.

If you plan to make a lot of changes to an item's image property, it is faster to copy the item's image property into a new image object using the `duplicate()` function, apply your changes to the new image object, and then set the original item's image to the new image object. For non-bitmap members, it is always faster to use the `duplicate()` function.

Example This statement puts the image of cast member `originalFlower` into cast member `newFlower`.

```
member("newFlower").image = member("originalFlower").image
```

Example These statements place a reference to the image of the stage into the variable `myImage` and then put that image into cast member `flower`.

```
myImage=(the stage).image  
member("flower").image = myImage
```

See also `setPixel()`, `draw()`, `image()`, `fill()`, `duplicate()` (image function), `copyPixels()`

image()

Syntax

```
image(width, height, bitDepth {, alphaDepth } {, paletteSymbolOrMember})
```

Description

Function; creates and returns a new image object of the dimensions specified by *width*, *height*, and *bitDepth*, with optional *alphaDepth* and *paletteObject* values.

The *bitDepth* can be 1, 2, 4, 8, 16, or 32. The *alphaDepth*, if given, is used only for 32-bit images and must be 0 or 8. The *paletteObject*, if given, is used only for 2-, 4-, and 8-bit images and can be either a palette symbol, such as #grayscale, or a palette cast member. If no palette is specified, the movie's default palette is used.

Image objects created with `image()` are independent and do not refer to any cast member or window.

To see an example of `image()` used in a completed movie, see the Imaging movie in the LearningLingo Examples folder inside the Director application folder.

Example

These statements create a 200 x 200 pixel, 8-bit image object and fills the image object with red.

```
redSquare = image(200, 200, 8)  
redSquare.fill(0, 0, 200, 200, rgb(255, 0, 0))
```

See also

`palette`, `image`, `duplicate()` (image function), `fill()`

imageCompression

Syntax

```
member(whichMember).imageCompression
```

the `imageCompression` of member *whichMember*

Description

This bitmap cast member property indicates the type of compression that Director will apply to the member when saving the movie in Shockwave format. This property can be tested and set, and has no effect at runtime. Its value can be any one of these symbols:

Value	Meaning
#movieSetting	Use the compression settings of the movie, as stored in the <code>movieImageCompression</code> property. This is the default value for image formats not restricted to standard compression (see below).
#standard	Use Director's standard internal compression format.
#jpeg	Use JPEG compression. See <code>imageQuality</code> .

You normally set this property in the Property Inspector's Bitmap tab. However, if you want to set this property for a large number of images at once, you can set the property with a Lingo routine.

If a member doesn't support JPEG compression because it is 8-bit or lower, or if the image is linked from an external file, only #standard compression can be used. Image formats that do not support JPEG compression include GIF and 8-bit or lower images.

Example This statement displays the imageCompression of member Sunrise in the message window:

```
put member("Sunrise").imageCompression  
-- #movieSetting
```

See also imageQuality, movieImageCompression, movieImageQuality

imageEnabled

Syntax sprite(*whichVectorOrFlashSprite*).imageEnabled

the imageEnabled of sprite *whichVectorOrFlashSprite*

member(*whichVectorOrFlashMember*).imageEnabled

the imageEnabled of member *whichVectorOrFlashMember*

Description Cast member property and sprite property; controls whether a Flash movie or vector shape's graphics are visible (TRUE, default) or invisible (FALSE).

This property can be tested and set.

Example This beginSprite script sets up a linked Flash movie sprite to hide its graphics when it first appears on the Stage and begins to stream into memory and saves its sprite number in a global variable called gStreamingSprite for use in a frame script later in the Score:

```
global gStreamingSprite  
on beginSprite me  
    gStreamingSprite = me.spriteNum  
    sprite(gStreamingSprite).imageEnabled = FALSE  
end
```

In a later frame of the movie, this frame script checks to see if the Flash movie sprite specified by the global variable gStreamingSprite has finished streaming into memory. If it has not, the script keeps the playback head looping in the current frame until 100% of the movie has streamed into memory. It then sets the imageEnabled property to TRUE so that the graphics appear and lets the playback head continue to the next frame in the Score.

```
global gStreamingSprite  
on exitFrame me  
    if sprite(gStreamingSprite).member.percentStreamed < 100 then  
        go to frame  
    else  
        sprite(gStreamingSprite).imageEnabled = TRUE  
        updatestage  
    end if  
end
```

imageQuality

Syntax

`member(whichMember).imageQuality`
the `imageQuality` of member *whichMember*

Description

This bitmap cast member property indicates the level of compression to use when the member's `imageCompression` property is set to `#jpeg`. The range of acceptable values is 0–100. Zero yields the lowest image quality and highest compression; 100 yields the highest image quality and lowest compression.

This property is settable only during authoring and only affects cast members when saving a movie in Shockwave format. The compressed image can be previewed via the Optimize in Fireworks button in the Property Inspector's Bitmap tab or the Preview in Browser command in the File menu.

If an image cast member's `imageCompression` property is set to `#MovieSetting`, the `movie` property `movielImageQuality` is used instead of `imageQuality`.

See also

`imageCompression`, `movielImageCompression`, `movielImageQuality`

importFileInto

Syntax

`importFileInto member whichCastMember, fileName`
`importFileInto member whichCastMember of castLib whichCast, fileName`
`importFileInto member whichCastMember, URL`

Description

Command; replaces the content of the cast member specified by *whichCastMember* with the file specified by *fileName*.

The `importFileInto` command is useful in four situations:

- When finishing developing a movie, use it to embed media that you have kept linked and external so it can be edited during the project.
- When generating Score from Lingo during movie creation, use it to assign content to new cast members that you created.
- When downloading files from the Internet, use it to download the file at a specific URL and set the file name of linked media. (However, to import a file from a URL, it's usually more efficient and minimizes downloading to use the `preloadNetThing` command to download the file to a local disk first and then import the file from the local disk.)
- Use it to import both RTF and HTML documents into text cast members with formatting and links intact.

Use of the `importFileInto` command in projectors can quickly consume available memory, so it's best to reuse the same members for imported data if possible.

Also, the importFileInto command doesn't work with the Director player for Java. To change the content of a bitmap or sound cast member in a movie playing back as an applet, make the cast members linked cast members and change the cast member's fileName property.

Note: In Shockwave, you must issue a preloadNetThing and wait for a successful completion of the download before using importFileInto with the file. In Director and projectors, importFileInto automatically downloads the file for you.

Example This handler assigns a URL that contains a GIF file to the variable tempURL and then uses the importFileInto command to import the file at the URL into a new bitmap cast member:

```
on exitFrame
    tempURL = "http://www.dukeOfUrl.com/crown.gif"
    importFileInto new(#bitmap), tempURL
end
```

Example This statement replaces the content of the sound cast member Memory with the sound file Wind:

```
importFileInto member "Memory", "Wind.wav"
```

Example These statements download an external file from a URL to the Director application folder and then import that file into the sound cast member Norma Desmond Speaks:

```
downloadNetThing http://www.cbDeMille.com/Talkies.AIF, the \
applicationPath&"Talkies.AIF"\ \
importFileInto(member "Norma Desmond Speaks", the applicationPath&"Talkies.AIF")
```

See also downloadNetThing, fileName (cast member property), preloadNetThing()

in

See also number (characters), number (items), number (lines), number (words)

INF

Description Lingo return value; indicates that a specified Lingo expression evaluates as an infinite number.

See also NAN

inflate

Syntax

```
rectangle.inflate(widthChange , heightChange)
```

```
inflate (rectangle, widthChange, heightChange)
```

Description

Command; changes the dimensions of the rectangle specified by *rectangle* relative to the center of the rectangle, either horizontally (*widthChange*) or vertically (*heightChange*).

The total change in each direction is twice the number you specify. For example, replacing *widthChange* with 15 increases the rectangle's width by 30 pixels. A value less than 0 for the horizontal or vertical dimension reduces the rectangle's size.

Example

This statement increases the rectangle's width by 4 pixels and the height by 2 pixels:

```
rect(10, 10, 20, 20).inflate(2, 1)  
-- rect (8, 9, 22, 21)
```

Example

This statement decreases the rectangle's height and width by 20 pixels:

```
inflate (rect(0, 0, 100, 100), -10, -10)  
-- rect (10, 10, 90, 90)
```

ink

Syntax

```
sprite(whichSprite).ink
```

the ink of sprite *whichSprite*

Description

Sprite property; determines the ink effect applied to the sprite specified by *whichSprite*, as follows:

0—Copy	32—Blend
1—Transparent	33—Add pin
2—Reverse	34—Add
3—Ghost	35—Subtract pin
4—Not copy	36—Background transparent
5—Not transparent	37—Lightest
6—Not reverse	38—Subtract
7—Not ghost	39—Darkest
8—Matte	40—Lighten
9—Mask	41—Darken

For a movie that plays back as an applet, valid values for the `ink` sprite property vary for different sprites, as follows:

- For bitmap sprites, the `ink` sprite property can be 0 (Copy), 8 (Matte), 32 (Blend), or 36 (Background transparent).
- For vector shape, Flash, and shape sprites, the `ink` sprite property can be 0, 8, or 36.
- For field sprites, the `ink` sprite property can be 0 or 36. The player treats Blend and Matte inks as Background transparent.

In the case of 36 (background transparent), you select a sprite in the score and select a transparency color from the background color box in the Tools window. You can also do this by setting the `backColor` property.

If you set this property within a script while the playback head is not moving, be sure to use the `updateStage` command to redraw the Stage. If you change several sprite properties—or several sprites—use only one `updateStage` command at the end of all the changes.

This property can be tested and set.

Example This statement changes the variable `currentInk` to the value for the ink effect of sprite (3):

```
currentInk = sprite(3).ink
```

Example This statement gives sprite (i + 1) a matte ink effect by setting the ink effect of the sprite property to 8, which specifies matte ink:

```
sprite(i + 1).ink = 8
```

See also `backColor`, `foreColor`

inlinemeEnabled

Syntax the `inlinemeEnabled`

Description Global property; determines whether Director's Inline IME feature is turned on. When `TRUE`, this property allows the user to enter double-byte characters directly into Director's Text, Field, Script and Message windows on Japanese systems.

This property can be tested and set. The default value is determined by the Enable Inline IME setting in Director's General Preferences.

insertFrame

Syntax insertFrame

Description Command; duplicates the current frame and its content. The duplicate frame is inserted after the current frame and then becomes the current frame.

This command can be used only during a Score recording session and performs the same function as the duplicateFrame command.

Example This handler generates a frame that has the transition cast member Fog assigned in the transition channel followed by a set of empty frames. The argument `numberOfFrames` sets the number of frames.

```
on animBall numberOfFrames
    beginRecording
        the frameTransition = member ("Fog").number
        go the frame + 1
        repeat with i = 0 to numberOfFrames
            insertFrame
        end repeat
    endRecording
end
```

inside()

Syntax point.inside(rectangle)

inside(point, rectangle)

Description Function; indicates whether the point specified by `point` is within the rectangle specified by `rectangle` (TRUE), or outside the rectangle (FALSE).

Example This statement indicates whether the point Center is within the rectangle Zone and displays the result in the Message window:

```
put Center.inside(Zone)
```

See also map(), mouseH, mouseV, point()

installMenu

Syntax

```
installMenu whichCastMember
```

Description

Command; installs the menu defined in the field cast member specified by *whichCastMember*. These custom menus appear only while the movie is playing. To remove the custom menus, use the `installMenu` command with no argument or with 0 as the argument. This command doesn't work with hierarchical menus.

For an explanation of how menu items are defined in a field cast member, see the `menu` keyword.

Avoid changing menus many times because doing so affects system resources.

In Windows, if the menu is longer than the screen, only part of the menu appears; on the Macintosh, menus longer than the screen can scroll.

Note: Menus are not available in Shockwave.

Example

This statement installs the menu defined in field cast member 37:

```
installMenu 37
```

Example

This statement installs the menu defined in the field cast member named Menubar:

```
installMenu member "Menubar"
```

Example

This statement disables menus that were installed by the `installMenu` command:

```
installMenu 0
```

See also

`menu`

integer()

Syntax

```
(numericExpression).integer
```

```
integer(numericExpression)
```

Description

Function; rounds the value of *numericExpression* to the nearest whole integer.

You can force an integer to be a string by using the `string()` function.

Example

This statement rounds off the number 3.75 to the nearest whole integer:

```
put integer(3.75)  
-- 4
```

Example

This statement rounds off the value in parentheses. This provides a usable value for the `locH` sprite property, which requires an integer:

```
sprite(1).locH = integer(0.333 * stageWidth)
```

See also

`float()`, `string()`

integerP()

Syntax

```
expression.integerP  
(numericExpression).integerP  
integerP(expression)
```

Description Function; indicates whether the expression specified by *expression* can be evaluated to an integer (1 or TRUE) or not (0 or FALSE). *P* in integerP stands for *predicate*.

Example This statement checks whether the number 3 can be evaluated to an integer and then displays 1 (TRUE) in the Message window:

```
put(3).integerP  
-- 1
```

Example This statement checks whether the number 3 can be evaluated to an integer. Because 3 is surrounded by quotation marks, it cannot be evaluated to an integer, so 0 (FALSE) is displayed in the Message window:

```
put("3").integerP  
-- 0
```

Example This statement checks whether the numerical value of the string in field cast member Entry is an integer and if it isn't, displays an alert:

```
if field("Entry").value.integerP = FALSE then alert "Please enter an integer."
```

See also floatP(), integer(), ilk(), objectP(), stringP(), symbolP()

interface()

Syntax

```
xtra("XtraName").interface()  
interface(xtra "XtraName")
```

Description Function; returns a Return-delimited string that describes the Xtra and lists its methods. This function replaces the now obsolete mMessageList function.

Example This statement displays the output from the function used in the QuickTime Asset Xtra in the Message window:

```
put Xtra("QuickTimeSupport").interface()
```

intersect()

Syntax	rectangle1. Intersect(<i>rectangle2</i>) intersect(<i>rectangle1</i> , <i>rectangle2</i>)
Description	Function; determines the rectangle formed where <i>rectangle1</i> and <i>rectangle2</i> intersect.
Example	This statement assigns the variable newRectangle to the rectangle formed where rectangle toolKit intersects rectangle Ramp: newRectangle = toolKit.intersect(Ramp)
See also	map(), rect(), union()

interval

Syntax	member(<i>whichCursorCastMember</i>).interval the interval of member <i>whichCursorCastMember</i>
Description	Cursor cast member property; specifies the interval, in milliseconds (ms), between each frame of the animated color cursor cast member <i>whichCursorCastMember</i> . The default interval is 100 ms. The cursor interval is independent of the frame rate set for the movie using the tempo channel or the puppetTempo Lingo command. This property can be tested and set.
Example	In this sprite script, when the animated color cursor stored in the cast member named Butterfly enters the sprite, the interval is set to 50 ms to speed up the animation. When the cursor leaves the sprite, the interval is reset to 100 ms to slow down the animation. on mouseEnter member("Butterfly").interval = 50 end on mouseLeave member("Butterfly").interval = 100 end

into

This code fragment occurs in a number of Lingo constructs, such as put...into.

invertMask

Syntax	member(<i>whichQuickTimeMember</i>).invertMask
	the invertMask of member <i>whichQuickTimeMember</i>
Description	QuickTime cast member property; determines whether Director draws QuickTime movies in the white pixels of the movie's mask (TRUE) or in the black pixels (FALSE, default).
	This property can be tested and set.
Example	This handler reverses the current setting of the invertMask property of a QuickTime movie named Starburst:
	on toggleMask member("Starburst").invertMask = not member("Starburst").invertMask end
See also	mask

isBusy()

Syntax	sound(<i>channel/Num</i>).isBusy()
Description	Function; returns TRUE if sound channel <i>channel/Num</i> is currently playing or pausing a sound, and FALSE if it hasn't started playing any of its queued sounds or has been stopped.
	To see an example of isBusy() used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.
Example	This Lingo statement checks whether a sound is playing in sound channel 1. If there is, the text of member field is set to "You should hear music now." Otherwise, the text reads "The music has ended."
	if sound(1).isBusy() then member("field").text = "You should hear music now." else member("field").text = "The music has ended." end if
See also	pause() (sound playback), playNext(), queue(), status, stop() (sound)

on isOKToAttach

Syntax

```
on isOKToAttach me, spriteType, spriteNum
```

Description

Built-in handler; you can add this handler to a behavior in order to check the type of sprite the behavior is being attached to and prevent the behavior from being attached to inappropriate sprite types.

When the behavior is attached to a sprite, the handler executes and Director passes to it the type of the sprite and its sprite number. The *me* argument contains a reference to the behavior that is being attached to the sprite.

This handler runs before the *on getPropertyDescriptionList* handler.

The Lingo author can check for two types of sprites. *#graphic* includes all graphic cast members, such as shapes, bitmaps, digital video, text, and so on. *#script* indicates the behavior was attached to the script channel. In this case, the *spriteNum* is 1.

For each of these sprite types, the handler must return TRUE or FALSE. A value of TRUE indicates that the behavior can be attached to the sprite. A value of FALSE prevents the behavior from being attached to the sprite.

If the behavior contains no *on isOKToAttach* handler, then the behavior can be attached to any sprite or frame.

This handler will only be called during the initial attachment of the behavior to the sprite or script channel and will not be called again when any other changes are made to the sprite.

Example

This statement checks the sprite type the behavior is being attached to and returns TRUE for any graphic sprite except a shape and FALSE for the script channel:

```
on isOKToAttach me, spriteType, spriteNum
  case spriteType of
    #graphic: -- any graphic sprite type
      return sprite(spriteNum).member.type <> #shape
      -- works for everything but shape cast members
    #script: --the frame script channel
      return FALSE -- doesn't work as a frame script
  end case
end
```

isPastCuePoint()

Syntax

```
sprite(spriteNum).isPastCuePoint(cuePointID)  
isPastCuePoint(sprite(spriteNum), cuePointID)  
sound(channelNum).isPastCuePoint(cuePointID)  
isPastCuePoint(sound(channelNum), cuePointID)
```

Description

Function; determines whether a sprite or sound channel has passed a specified cue point in its media. This function can be used with sound (WAV, AIFF, SND, SWA, AU), QuickTime, or Xtra files that support cue points.

Replace *spriteNum* or *channelNum* with a sprite channel or a sound channel. Shockwave Audio (SWA) sounds can appear as sprites in sprite channels, but they play sound in a sound channel. It is recommended that you refer to SWA sound sprites by their sprite channel number rather than their sound channel number.

Replace *cuePointID* with a reference for a cue point:

- If *cuePointID* is an integer, *isPastCuePoint* returns 1 if the cue point has been passed and 0 if it hasn't been passed.
- If *cuePointID* is a name, *isPastCuePoint* returns the number of cue points passed that have that name.

If the value specified for *cuePointID* doesn't exist in the sprite or sound, the function returns 0.

The number returned by *isPastCuePoint* is based on the absolute position of the sprite in its media. For example, if a sound passes cue point Main and then loops and passes Main again, *isPastCuePoint* returns 1 instead of 2.

When the result of *isPastCuePoint* is treated as a Boolean operator, the function returns TRUE if any cue points identified by *cuePointID* have passed and FALSE if no cue points are passed.

Example

This statement plays a sound until the third time the cue point Chorus End is passed:

```
if (isPastCuePoint(sound 1, "Chorus End")=3) then  
    puppetSound 0  
end if
```

Example

This displays information in cast member “field 2” about the music playing in sound channel 1. If the music is not yet past cue point “climax”, the text of “field 2” is “This is the beginning of the piece.” Otherwise, the text reads “This is the end of the piece.”

```
if not sound(1).isPastCuePoint("climax") then  
    member("field 2").text = "This is the beginning of the piece."  
else  
    member("field 2").text = "This is the end of the piece."  
end if
```

isVRMovie

Syntax

`member(whichCastMember).isVRMovie`
isVRMovie of member *whichCastMember*
`sprite(whichSprite).isVRMovie`
isVRMovie of sprite *whichSprite*

Description

QuickTime cast member and sprite property; indicates whether a cast member or sprite is a QuickTime VR movie that has not yet been downloaded (TRUE), or whether the cast member or sprite isn't a QuickTime VR movie (FALSE).

Testing for this property in anything other than an asset whose type is #quickTimeMedia produces an error message.

This property can be tested but not set.

Example

This handler checks to see if the member of a sprite is a QuickTime movie. If it is, the handler further checks to see if it is a QTVR movie. An alert is posted in any case.

```
on checkForVR theSprite
    if sprite(theSprite).member.type = #quickTimeMedia then
        if sprite(theSprite).isVRMovie then
            alert "This is a QTVR asset."
        else
            alert "This is not a QTVR asset."
        end if
    else
        alert "This is not a QuickTime asset."
    end if
end
```

item...of

Syntax

`textMemberExpression.item[whichItem]`
item *whichItem* of *fieldOrStringVariable*
`textMemberExpression.item[firstItem..lastItem]`
item *firstItem* to *lastItem* of *fieldOrStringVariable*

Description

Keyword; specifies an item or range of items in a chunk expression. An item in this case is any sequence of characters delimited by the current delimiter as determined by the itemDelimiter property.

The terms *whichItem*, *firstItem*, and *lastItem* must be integers or integer expressions that refer to the position of items in the chunk.

Chunk expressions refer to any character, word, item, or line in any source of strings. Sources of strings include field and text cast members and variables that hold strings.

When the number that specifies the last item is greater than the item's position in the chunk expression, the actual last item is specified instead.

Example This statement looks for the third item in the chunk expression that consists of names of colors and then displays the result in the Message window:

```
put "red, yellow, blue green, orange".item[3]  
-- "blue green"
```

The result is the entire chunk “blue green” because this is the entire chunk between the commas.”

Example This statement looks for the third through fifth items in the chunk expression. Because there are only four items in the chunk expression, only the third item is used and fourth items are returned. The result appears in the Message window.

```
put "red, yellow, blue green, orange".item[3..5]  
-- " blue green, orange"  
put item 5 of "red, yellow, blue green, orange"  
-- ""
```

Example This statement inserts the item Desk as the fourth item in the second line of the field cast member All Bids:

```
member("All Bids").line[2].item[4] = "Desk"
```

See also char...of, itemDelimiter, number (items), word...of

itemDelimiter

Syntax the itemDelimiter

Description System property; indicates the special character used to separate items.

You can use the itemDelimiter to parse file names by setting itemDelimiter to a backslash (\) in Windows or a colon (:) on the Macintosh. Restore the itemDelimiter character to a comma (,) for normal operation.

This function can be tested and set.

Example This handler finds the last component in a Macintosh pathname. The handler first records the current delimiter and then changes the delimiter to a colon (:). When a colon is the delimiter, Lingo can use the last item of to determine the last item in the chunk that makes up a Macintosh pathname. Before exiting, the delimiter is reset to its original value.

```
on getLastComponent pathname  
    save = the itemDelimiter  
    the itemDelimiter = ":"  
    f = the last item of pathname  
    the itemDelimiter = save  
    return f  
end
```

kerning

Syntax member(*whichTextMember*).kerning

Description Text cast member property; this property specifies whether the text is automatically kerned when the contents of the text cast member are changed.

When set to TRUE, kerning is automatic; when set to FALSE, kerning is not done. This property defaults to TRUE.

See also kerningThreshold

kerningThreshold

Syntax member(*whichTextMember*).kerningThreshold

Description Text cast member property; this setting controls the size at which automatic kerning takes place in a text cast member. This has an effect only when the kerning property of the text cast member is set to TRUE.

The setting itself is an integer indicating the font point size at which kerning takes place.

This property defaults to 14 points.

See also kerning

key()

Syntax the key

Description Function; indicates the last key that was pressed. This value is the American National Standards Institute (ANSI) value assigned to the key, not the numerical value.

You can use the key in handlers that perform certain actions when the user presses specific keys as shortcuts and other forms of interactivity. When used in a primary event handler, the actions you specify are the first to be executed.

Note: The value of the key isn't updated if the user presses a key while Lingo is in a repeat loop.

Use the sample movie Keyboard Lingo to test which characters correspond to different keys on different keyboards.

Example These statements cause the movie to return to the main menu marker when the user presses the Return key. Because the keyDownScript property is set to checkKey, the on prepareMovie handler makes the on checkKey handler the first event handler executed when a key is pressed. The on checkKey handler checks whether the Return key is pressed and if it is, navigates to the main menu marker.

```
on prepareMovie
    the keyDownScript = "checkKey"
end prepareMovie

on checkKey
    if the key = RETURN then go to frame "Main Menu"
end
```

Example This on keyDown handler checks whether the last key pressed is the Enter key and if it is, then calls the on addNumbers handler:

```
on keyDown
    if the key = RETURN then addNumbers
end keyDown
```

See also commandDown, controlDown, keyCode(), optionDown

keyboardFocusSprite

Syntax set the keyboardFocusSprite = *textSpriteNum*

Description System property; lets the user set the focus for keyboard input (without controlling the cursor's insertion point) on a particular text sprite currently on the screen. This is the equivalent to using the Tab key when the AutoTab property of the member is selected.

Setting keyboardFocusSprite to -1 returns keyboard focus control to the score, and setting it to 0 disables keyboard entry into any editable sprite.

See also autoTab, editable

keyCode()

Syntax the keyCode

Description Function; gives the numerical code for the last key pressed. This keyboard code is the key's numerical value, not the American National Standards Institute (ANSI) value.

Note: When a movie plays back as an applet, this function returns the values of only function and arrow keys.

You can use the keyCode function to detect when the user has pressed an arrow or function key, which cannot be specified by the key function.

Use the sample movie Keyboard Lingo to test which characters correspond to different keys on different keyboards.

This function can be tested but not set.

Example This handler uses the Message window to display the appropriate key code each time a key is pressed:

```
on enterFrame
    the keydownScript = "put the keyCode"
end
```

Example This statement checks whether the up arrow (whose key code is 126) was pressed and if it was, goes to the previous marker:

```
if the keyCode = 126 then go to marker(-1)
```

Example This handler checks whether one of the arrow keys was pressed and if one was, responds accordingly:

```
on keyDown
    case (the keyCode) of
        123: TurnLeft
        126: GoForward
        125: BackUp
        124: TurnRight
    end case
end
```

See also commandDown, controlDown, key(), optionDown

on keyDown

Syntax on keyDown

```
    statement(s)
end
```

Description System message and event handler; contains statements that run when a key is pressed.

When a key is pressed, Director searches these locations, in order, for an on keyDown handler: primary event handler, editable field sprite script, field cast member script, frame script, and movie script. For sprites and cast members, on keyDown handlers work only for editable text and field members. A keyDown event on a different type of cast member, such as a bitmap, has no effect. (If pressing a key should have the same response throughout the movie, set keyDownScript.)

Director stops searching when it reaches the first location that has an on keyDown handler, unless the handler includes the pass command to explicitly pass the keyDown message on to the next location.

The on keyDown event handler is a good place to put Lingo that implements keyboard shortcuts or other interface features that you want to occur when the user presses keys.

The Director player for Java responds to keyDown messages only if the movie has focus in the browser. The user must click in the applet before the applet can receive any keys that the user types.

When the movie plays back as an applet, an on keyDown handler always traps key presses, even if the handler is empty. If the user is typing in an editable field, an on keyDown handler attached to the field must include the pass command for the key to appear in the field.

Where you place an on keyDown handler can affect when it runs.

- To apply the handler to a specific editable field sprite, put the handler in a sprite script.
- To apply the handler to an editable field cast member in general, put the handler in a cast member script.
- To apply the handler to an entire frame, put the handler in a frame script.
- To apply the handler throughout the entire movie, put the handler in a movie script.

You can override an on keyDown handler by placing an alternative on keyDown handler in a location that Lingo checks before it gets to the handler you want to override. For example, you can override an on keyDown handler assigned to a cast member by placing an on keyDown handler in a sprite script.

Example

This handler checks whether the Return key was pressed and if it was, sends the playback head to another frame:

```
on keyDown
    if the key = RETURN then go to frame "AddSum"
end keyDown
```

See also

charToNum(), keyDownScript, keyUpScript, key(), keyCode(), keyPressed()

keyDownScript

Syntax the keyDownScript

Description System property; specifies the Lingo that is executed when a key is pressed. The Lingo is written as a string, surrounded by quotation marks, and can be a simple statement or a calling script for a handler.

When a key is pressed and the keyDownScript property is defined, Lingo executes the instructions specified for the keyDownScript property first. Unless the instructions include the pass command so that the keyDown message can be passed on to other objects in the movie, no other on keyDown handlers are executed.

Setting the keyDownScript property performs the same function as using the when keyDown then command that appeared in earlier versions of Director.

When the instructions you specify for the keyDownScript property are no longer appropriate, turn them off by using the statement set the keyDownScript to EMPTY.

Example This statement sets keyDownScript to if the key = RETURN then go to the frame + 1. When this statement is in effect, the movie always goes to the next frame whenever the user presses the Return key.

the keyDownScript = "if the key = RETURN then go to the frame + 1"

Example This statement sets keyDownScript to the custom handler *myCustomHandler*. A Lingo custom handler must be enclosed in quotation marks when used with the keyDownScript property.

the keyDownScript = "myCustomHandler"

See also on keyDown, keyUpScript, mouseDownScript, mouseUpScript

keyPressed()

Syntax the keyPressed

keyPressed (keyCode)

keyPressed (asciiCharacterString)

Description Function; returns the character assigned to the key that was last pressed if no argument is used. The result is in the form of a string. When no key has been pressed, the keyPressed is an empty string.

If an argument is used, either a keyCode or the ASCII string for the key being pressed may be used. In either of these cases, the return value is TRUE if that particular key is being pressed, or FALSE if not.

The Director player for Java doesn't support this property. As a result, a movie playing back as an applet has no way to detect which key the user pressed while Lingo is in a repeat loop.

The keyPressed property is updated when the user presses keys while Lingo is in a repeat loop. This is an advantage over the key function, which doesn't update when Lingo is in a repeat loop.

Use the sample movie Keyboard Lingo to test which characters correspond to different keys on different keyboards.

This property can be tested but not set.

Example The following statement checks whether the user pressed the Enter key in Windows or the Return key on a Macintosh and runs the handler updateData if the key was pressed:

```
if the keyPressed = RETURN then updateData
```

Example This statement uses the keyCode for the *a* key to test if it's down, and displays the result in the Message window:

```
if keyPressed(0) then put "Key is down"
```

Example This statement uses the ASCII strings to test if the *a* and *b* keys are down, and displays the result in the Message window:

```
if keyPressed("a") and keyPressed ("b") then put "Keys are down"
```

See also keyCode(), key()

on keyUp

Syntax on keyUp

```
    statement(s)
```

```
end
```

Description System message and event handler; contains statements that run when a key is released. The on keyUp handler is similar to the on keyDown handler, except this event occurs after a character appears if a field or text sprite is editable on the screen.

When a key is released, Lingo searches these locations, in order, for an on keyUp handler: primary event handler, editable field sprite script, field cast member script, frame script, and movie script. For sprites and cast members, on keyUp handlers work only for editable strings. A keyUp event on a different type of cast member, such as a bitmap, has no effect. If releasing a key should always have the same response throughout the movie, set keyUpScript.

Lingo stops searching when it reaches the first location that has an on keyUp handler, unless the handler includes the pass command to explicitly pass the keyUp message on to the next location.

The on keyUp event handler is a good place to put Lingo that implements keyboard shortcuts or other interface features that you want to occur when the user releases keys.

The Director player for Java responds to keyUp messages only if the movie has focus in the browser. The user must click in the applet before the applet can receive any keys that the user types.

When the movie plays back as an applet, an on keyUp handler always traps key presses, even if the handler is empty. If the user is typing in an editable field, an on keyUp handler attached to the field must include the pass command for the key to appear in the field.

Where you place an on keyUp handler can affect when it runs, as follows:

- To apply the handler to a specific editable field sprite, put it in a behavior.
- To apply the handler to an editable field cast member in general, put it in a cast member script.
- To apply the handler to an entire frame, put it in a frame script.
- To apply the handler throughout the entire movie, put it in a movie script.

You can override an on keyUp handler by placing an alternative on keyUp handler in a location that Lingo checks before it gets to the handler you want to override. For example, you can override an on keyUp handler assigned to a cast member by placing an on keyUp handler in a sprite script.

Example This handler checks whether the Return key was released and if it was, sends the playback head to another frame:

```
on keyUp
    if the key = RETURN then go to frame "AddSum"
end keyUp
```

See also on keyDown, keyDownScript, keyUpScript

keyUpScript

Syntax the keyUpScript

Description System property; specifies the Lingo that is executed when a key is released. The Lingo is written as a string, surrounded by quotation marks, and can be a simple statement or a calling script for a handler.

When a key is released and the keyUpScript property is defined, Lingo executes the instructions specified for the keyUpScript property first. Unless the instructions include the pass command so that the keyUp message can be passed on to other objects in the movie, no other on keyUp handlers are executed.

When the instructions you've specified for the keyUpScript property are no longer appropriate, turn them off by using the statement set the keyUpScript to empty.

Example	This statement sets keyUpScript to if the key = RETURN then go the frame + 1. When this statement is in effect, the movie always goes to the next frame whenever the user presses the Return key.
	the keyUpScript = "if the key = RETURN then go to the frame + 1"
Example	This statement sets keyUpScript to the custom handler myCustomHandler. A Lingo custom handler must be enclosed in quotation marks when used with the keyUpScript property.
	the keyUpScript = "myCustomHandler"

See also on keyUp

label()

Syntax	label(<i>expression</i>)
Description	Function; indicates the frame associated with the marker label specified by <i>expression</i> . The term <i>expression</i> should be a label in the current movie; if it's not, this function returns 0.
Example	This statement sends the playback head to the tenth frame after the frame labeled Start:
	go to label("Start") + 10
Example	This statement assigns the frame number of the fourth item in the label list to the variable whichFrame:
	whichFrame = label(the labelList.line[4])
See also	go, frameLabel, labelList, marker(), play

labelList

Syntax	the labelList
Description	System property; lists the frame labels in the current movie as a Return-delimited string (not a list) containing one label per line. Labels are listed according to their order in the Score. (Because the entries are Return-delimited, the end of the string is an empty line after the last Return. Be sure to remove this empty line if necessary.)
Example	This statement makes a list of frame labels in the content of the field cast member Key Frames:

```
member("Key Frames").text = the labelList
```

Example This handler determines the label that starts the current scene:

```
on findLastLabel
    aa = label(0)
    repeat with i = 1 to (the labelList.line.count - 1)
        if aa = label(the labelList.line[i]) then
            return the labelList.line[i]
        end if
    end repeat
end
```

See also [frameLabel](#), [label\(\)](#), [marker\(\)](#)

last()

Syntax the last chunk of (*chunkExpression*)

the last *chunk* in (*chunkExpression*)

Description Function; identifies the last chunk specified by *chunk* in the chunk expression specified by *chunkExpression*.

Chunk expressions refer to any character, word, item, or line in a container of character. Supported containers are field cast members, variables that hold strings, and specified characters, words, items, lines, and ranges within containers.

Example This statement identifies the last word of the string “Macromedia, the multimedia company” and displays the result in the Message window:

```
put the last char of "Macromedia, the multimedia company" "
```

The result is the word *company*.

Example This statement identifies the last character of the string “Macromedia, the multimedia company” and displays the result in the Message window:

```
put last char("Macromedia, the multimedia company")
```

The result is the letter *y*.

See also [char...of](#), [word...of](#)

lastChannel

Syntax	the lastChannel
Description	Movie property; the number of the last channel in the movie, as entered in the Movie Properties dialog box.
	This property can be tested but not set.
	To see an example of lastChannel used in a completed movie, see the QT and Flash movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement displays the number of the last channel of the movie in the Message window:

```
put the lastChannel
```

lastClick()

Syntax	the lastClick
Description	Function; returns the time in ticks (1 tick = 1/60 of a second) since the mouse button was last pressed.
	This function can be tested but not set.
Example	This statement checks whether 10 seconds have passed since the last mouse click and, if so, sends the playback head to the marker No Click:
	if the lastClick > 10 * 60 then go to "No Click"

See also lastEvent(), lastKey, lastRoll, startTimer

lastEvent()

Syntax	the lastEvent
Description	Function; returns the time in ticks (1 tick = 1/60 of a second) since the last mouse click, rollover, or key press occurred.
Example	This statement checks whether 10 seconds have passed since the last mouse click, rollover, or key press, and, if so sends the playback head to the marker Help:
	if the lastEvent > 10 * 60 then go to "Help"

See also lastClick(), lastKey, lastRoll, startTimer

lastFrame

Syntax	the lastFrame
Description	Movie property; displays the number of the last frame in the movie. This property can be tested but not set.
Example	This statement displays the number of the last frame of the movie in the Message window: <pre>put the lastFrame</pre>

lastKey

Syntax	the lastKey
Description	System property; gives the time in ticks (1 tick = 1/60 of a second) since the last key was pressed.
Example	This statement checks whether 10 seconds have passed since the last key was pressed and, if so, sends the playback head to the marker No Key: <pre>if the lastKey > 10 * 60 then go to "No Key"</pre>
See also	lastClick(), lastEvent(), lastRoll, startTimer

lastRoll

Syntax	the lastRoll
Description	System property; gives the time in ticks (1 tick = 1/60 of a second) since the mouse was last moved.
Example	This statement checks whether 45 seconds have passed since the mouse was last moved and, if so, sends the playback head to the marker Attract Loop: <pre>if the lastRoll > 45 * 60 then go to "Attract Loop"</pre>
See also	lastClick(), lastEvent(), lastKey, startTimer

left

Syntax	<code>sprite(<i>whichSprite</i>).left</code>
	the left of sprite <i>whichSprite</i>
Description	Sprite property; identifies the left horizontal coordinate of the bounding rectangle of the sprite specified by <i>whichSprite</i> .

Sprite coordinates are measured in pixels, starting with (0,0) at the upper left corner of the Stage.

When a movie plays back as an applet, this property's value is relative to the left edge of the applet.

This property can be tested and set.

Example The following statement determines whether the sprite's left edge is to the left of the Stage's left edge. If the sprite's left edge is to the Stage's left edge, the script runs the handler offLeftEdge:

```
if sprite(3).left < 0 then offLeftEdge
```

Example This statement measures the left horizontal coordinate of the sprite numbered (i + 1) and assigns the value to the variable named vLowest:

```
set vLowest = sprite (i + 1).left
```

See also bottom, height, locH, locV, right, top, width

leftIndent

Syntax *chunkExpression.leftIndent*

Description Text cast member property; contains the number of pixels the left margin of *chunkExpression* is offset from the left side of the text cast member.

The value is an integer greater than or equal to 0.

This property can be tested and set.

Example This line indents the first line of text cast member "theStory" by ten pixels:

```
member("theStory").line[1].leftIndent = 10
```

See also firstIndent, rightIndent

length()

Syntax *string.length*

length(string)

Description Function; returns the number of characters in the string specified by *string*, including spaces and control characters such as TAB and RETURN.

Example This statement displays the number of characters in the string "Macro" & "media":

```
put ("Macro" & "media").length  
-- 10
```

Example This statement checks whether the content of the field cast member File Name has more than 31 characters and if it does, displays an alert:

```
if member("File Name").text.length > 31 then  
    alert "That file name is too long."  
end if"
```

See also chars(), offset() (string function)

line...of

Syntax *textMemberExpression.line[whichLine]*

line whichLine of fieldOrStringVariable

textMemberExpression.line[firstLine..lastLine]

line firstLine to lastLine of fieldOrStringVariable

Description Keyword; specifies a line or a range of lines in a chunk expression. A line chunk is any sequence of characters delimited by carriage returns, not by line breaks caused by text wrapping.

The expressions *whichLine*, *firstLine*, and *lastLine* must be integers that specify a line in the chunk.

Chunk expressions refer to any character, word, item, or line in any source of characters. Sources of characters include field cast members and variables that hold strings.

Example This statement assigns the first four lines of the variable Action to the field cast member To Do:

```
member("To Do").text = Action.line[1..4]
```

Example This statement inserts the word and after the second word of the third line of the string assigned to the variable Notes:

```
put "and" after Notes.line[3].word[2]
```

See also char...of, item...of, word...of, and, number (words)

lineCount

Syntax	<code>member(whichCastMember).lineCount</code>
	the lineCount of member <i>whichCastMember</i>
Description	Cast member property; indicates the number of lines that appear in the field cast member on the Stage according to the way the string wraps, not the number of carriage returns in the string.
Example	This statement determines how many lines the field cast member Today's News has when it appears on the Stage and assigns the value to the variable <i>numberOfLines</i> :

```
numberOfLines = member("Today's News").lineCount
```

lineDirection

Syntax	<code>member(whichCastMember).lineDirection</code>
Description	Shape member property; this property contains a 0 or 1 indicating the slope of the line drawn. If the line is inclined from left to right, the property is set to 1; and if it is declined from left to right, the property is set to 0. This property can be tested and set.
Example	This toggles the slope of the line in cast member "theLine", producing a see-saw effect:
	<pre>on seeSaw member("theLine").lineDirection = \ not member("theLine").lineDirection end</pre>

lineHeight() (function)

Syntax	<code>member(whichCastMember).lineHeight(<i>lineNumber</i>)</code>
	<code>lineHeight(member <i>whichCastMember</i>, <i>lineNumber</i>)</code>
Description	Function; returns the height, in pixels, of a specific line in the specified field cast member.
Example	This statement determines the height, in pixels, of the first line in the field cast member Today's News and assigns the result to the variable <i>headline</i> :

```
headline = member("Today's News").lineHeight(1)
```

lineHeight (cast member property)

Syntax

`member(whichCastMember).lineHeight`

the `lineHeight` of member `whichCastMember`

Description

Cast member property; determines the line spacing used to display the specified field cast member. The parameter `whichCastMember` can be either a cast member name or number.

Setting the `lineHeight` member property temporarily overrides the system's setting until the movie closes. To use the desired line spacing throughout a movie, set the `lineHeight` member property in an `on prepareMovie` handler.

This property can be tested and set.

Example

This statement sets the variable `oldHeight` to the current `lineHeight` setting for the field cast member `Rokujo Speaks`:

```
oldHeight = member("Rokujo Speaks").lineHeight
```

See also

`text`; `alignment`, `font`, `fontSize`, `fontStyle`

linePosToLocV()

Syntax

`member(whichCastMember).linePosToLocV(lineNumber)`

`linePosToLocV(member whichCastMember, lineNumber)`

Description

Function; returns a specific line's distance, in pixels, from the top edge of the field cast member.

Example

This statement measures the distance, in pixels, from the second line of the field cast member `Today's News` to the top of the field cast member and assigns the result to the variable `startOfString`:

```
startOfString = member("Today's News").linePosToLocV(2)
```

lineSize

Syntax

`member(whichCastMember).lineSize`

the `lineSize` of member `whichCastMember`

`sprite whichSprite.lineSize`

the `lineSize` of sprite `whichSprite`

Description

Shape cast member property; determines the thickness, in pixels, of the border of the specified shape cast member displayed on the Stage. For nonrectangular shapes, the border is the edge of the shape, not its bounding rectangle.

The lineSize setting of the sprite takes precedence over the lineSize setting of the member. If Lingo changes the member's lineSize setting while a sprite is on the Stage, the sprite's lineSize setting remains in effect until the sprite is finished.

For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet.

This property can be tested and set.

Example This statement sets the thickness of the shape cast member Answer Box to 5 pixels:

```
member("Answer Box").lineSize = 5
```

Example This statement displays the thickness of the border of sprite 4:

```
thickness = sprite(4).lineSize
```

Example This statement sets the thickness of the border of sprite 4 to 3 pixels:

```
sprite(4).lineSize = 3
```

linkAs()

Syntax *castMember*.linkAs()

Description Script cast member function; opens a save dialog box, allowing you to save the contents of the script to an external file. The script cast member is then linked to that file.

Linked scripts are imported into the movie when you save it as a projector, Shockwave movie, or Java movie. This differs from other linked media, which remains external to the movie unless you explicitly import it.

Example This statement, typed in the Message window, opens a save dialog box to save the script Random Motion as an external file:

```
member("Random Motion").linkAs()
```

```
importFileInto, linked
```

linked

Syntax	<code>member(<i>whichMember</i>).linked</code> the linked of member <i>whichMember</i>
Description	Cast member property; controls whether a script, Flash movie, or animated GIF file is stored in an external file (TRUE, default), or inside the Director cast (FALSE). When the data is stored externally, the cast member's pathName property must point to the location where the movie file can be found.
	This property can be tested and set for script, Flash, and GIF members. It may be tested for all member types.
Example	This converts Flash cast member "Homebodies" from a linked member to an internally stored member. <code>member("homeBodies").linked = 0</code>
See also	<code>fileName</code> (cast member property), <code>pathName</code> (cast member property)

list()

Syntax	<code>list(<i>value1</i>, <i>value2</i>, <i>value3</i>...)</code>
Description	Function and data type; defines a linear list made up of the values specified by <i>value1</i> , <i>value2</i> , <i>value3</i> This is an alternative to using square brackets ([]) to create a list. The maximum length of a single line of executable Lingo is 256 characters. You can't create a very large list using this command. If you have a large amount of data that you want to put in a list, enclose the data in square brackets and put the data into a field. You can then assign the field to a variable. The variable's content is a list of the data.
Example	This statement sets the variable named designers equal to a linear list that contains the names Gee, Kayne, and Ohashi: <code>designers = list("Gee", "Kayne", "Ohashi")</code> The result is the list ["Gee", "Kayne", "Ohashi"].
See also	<code>integer()</code> , <code>integerP()</code> , <code>value()</code>

listP()

Syntax	listP(<i>item</i>)
Description	Function; indicates whether the item specified by <i>item</i> is a list, rectangle, or point (1 or TRUE) or not (0 or FALSE).
Example	This statement checks whether the list in the variable <i>designers</i> is a list, rectangle, or point, and displays the result in the Message window: <pre>put listP(designers)</pre> The result is 1, which is the numerical equivalent of TRUE.
See also	ilk(), objectP()

loaded

Syntax	member(<i>whichCastMember</i>).loaded the loaded of member <i>whichCastMember</i>
Description	Cast member property; specifies whether the cast member specified by <i>whichCastMember</i> is loaded into memory (TRUE) or not (FALSE). Different cast member types have slightly different behaviors for loading: <ul style="list-style-type: none">• Shape and script cast members are always loaded into memory.• Movie cast members are never unloaded.• Digital video cast members can be preloaded and unloaded independent of whether they are being used. (A digital video cast member plays faster from memory than from disk.) This property can be tested but not set.
Example	This statement checks whether cast member Demo Movie is loaded in memory and if it isn't, goes to an alternative movie: <pre>if member("Demo Movie").loaded = FALSE then go to movie("Waiting")"</pre>
See also	preLoad (command), ramNeeded(), size, unLoad

loc

Syntax

`sprite whichSprite.loc`

the loc of sprite *whichSprite*

Description

Sprite property; determines the Stage coordinates of the specified sprite's registration point. The value is given as a point.

This property can be tested and set.

To see an example of `loc` used in a completed movie, see the Imaging movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement checks the Stage coordinates of sprite 6. The result is the point (50, 100):

```
put sprite(6).loc  
-- point(50, 100)
```

See also

`bottom, height, left, locV, right, top, width`

locH

Syntax

`sprite(whichSprite).locH`

the locH of sprite *whichSprite*

Description

Sprite property; indicates the horizontal position of the specified sprite's registration point. Sprite coordinates are relative to the upper left corner of the Stage.

This property can be tested and set. To make the value last beyond the current sprite, make the sprite a puppet.

Example

This statement checks whether the horizontal position of sprite 9's registration point is to the right of the right edge of the stage and moves the sprite to the left edge of the Stage if it is:

```
if sprite(9).locH > (the stageRight - the stageLeft) then  
    sprite(9).locH = 0  
end if
```

Example

This statement puts sprite 15 at the same horizontal location as the mouse click:

```
sprite(15).locH = the mouseH
```

See also

`bottom, height, left, locV, point(), right, top, width, updateStage`

locToCharPos()

Syntax

```
member(whichCastMember).locToCharPos(location)
```

```
locToCharPos(member whichCastMember, location)
```

Description

Function; returns a number that identifies which character in the specified field cast member is closest to the point within the field specified by *location*. The value for *location* is a point relative to the upper left corner of the field cast member.

The value 1 corresponds to the first character in the string, the value 2 corresponds to the second character in the string, and so on.

Example

This statement determines which character is closest to the point 100 pixels to the right and 100 pixels below the upper left corner of the field cast member Today's News. The statement then assigns the result to the variable PageDesign.

```
pageDesign = member("Today's News").locToCharPos(point(100, 100))
```

Example

This handler tells which character is under the cursor when the user clicks the mouse over the field sprite Information:

```
on mouseDown
    put member("Information").locToCharPos(the clickLoc - \
        (sprite(the clickOn).loc))
end
```

locV

Syntax

```
sprite(whichSprite).locV
```

```
the locV of sprite whichSprite
```

Description

Sprite property; indicates the vertical position of the specified sprite's registration point. Sprite coordinates are relative to the upper left corner of the Stage.

This property can be tested and set. To make the value last beyond the current sprite, make the sprite a puppet.

Example

This statement checks whether the vertical position of sprite 9's registration point is below the bottom of the Stage and moves the sprite to the top of the Stage if it is:

```
if sprite(9).locV > (the stageBottom - the stageTop) then
    sprite(9).locV = 0
end if
```

Example

This statement puts sprite 15 at the same vertical location as the mouse click:

```
sprite(15).locV = the mouseV
```

See also

bottom, height, left, locH, right, top, width, updateStage

locVToLinePos()

Syntax

member(*whichCastMember*). locVToLinePos(*locV*)

Description

Function; returns the number of the line of characters that appears at the vertical position specified by *locV*. The *locV* value is the number of pixels from the top of the field cast member, not the part of the field cast member that currently appears on the Stage.

Example

This statement determines which line of characters appears 150 pixels from the top of the field cast member Today's News and assigns the result to the variable pageBreak:

```
pageBreak = member("Today's News").locVToLinePos(150)
```

locZ of sprite

Syntax

sprite(*whichSprite*).locZ

Description

Sprite property; specifies the dynamic Z-order of a sprite, to control layering without having to manipulate sprite channels and properties.

This property can be tested and set.

This property can have an integer value from negative 2 billion to positive 2 billion. Larger numbers cause the sprite to appear in front of sprites with smaller numbers. If two sprites have the same locZ value, the channel number then takes precedence for deciding the final display order of those two sprites. This means sprites in lower numbered channels appear behind sprites in higher numbered channels even when the locZ values are equal.

By default, each sprite has a locZ value equal to its own channel number.

Layer-dependent operations such as hit detection and mouse events obey sprites' locZ values, so changing a sprite's locZ value can make the sprite partially or completely obscured by other sprites and the user may be unable to click on the sprite.

Other Director functions do not follow the locZ ordering of sprites. Generated events still begin with channel 1 and increase consecutively from there, regardless of the sprite's Z-order.

Example This handler uses a global variable called gHighestSprite which has been initialized in the startMovie handler to the number of sprites used. When the sprite is clicked, its locZ is set to gHighestSprite + 1, which moves the sprite to the foreground on the stage. Then gHighestSprite is incremented by 1 to prepare for the next mouseUp call.

```
on mouseUp me
    global gHighestSprite
    sprite(me.spriteNum).locZ = gHighestSprite + 1
    gHighestSprite = gHighestSprite + 1
end
```

See also locH, locV

log()

Syntax log(*number*)

Description Math function; calculates the natural logarithm of the number specified by *number*, which must be a decimal number greater than 0.

Example This statement assigns the natural logarithm of 10.5 to the variable Answer.

```
Answer = log(10.5)
```

Example This statement calculates the natural logarithm of the square root of the value Number and then assigns the result to the variable Answer:

```
Answer = log(Number.sqrt)
```

long

See date() (system clock), time() functions

loop (keyword)

Syntax loop

Description Keyword; refers to the marker. The loop keyword with the go to command is equivalent to the statement go to the marker.

Example This handler loops the movie between the previous marker and the current frame:

```
on exitFrame
    go loop
end exitFrame
```

loop (cast member property)

Syntax `member(whichCastMember).loop`
 the loop of member *whichCastMember*

Description Cast member property; determines whether the specified digital video, sound, or Flash movie cast member is set to loop (TRUE) or not (FALSE).

Example This statement sets the QuickTime movie cast member Demo to loop:
`member("Demo").loop = 1`

loop (Flash property)

Syntax `sprite(whichFlashSprite).loop`
 the loop of sprite *whichFlashSprite*
`member (whichFlashMember).loop`
 the loop of member *whichFlashMember*

Description Flash sprite and member property; controls whether a Flash movie plays in a continuous loop (TRUE) or plays once and then stops (FALSE).

The property can be both tested and set.

Example This frame script checks the download status of a linked Flash cast member called NetFlash using the percentStreamed property. While NetFlash is downloading, the movie loops in the current frame. When NetFlash finishes downloading, the movie advances to the next frame and the loop property of the Flash movie in channel 6 is set to FALSE so that it will continue playing through to the end and then stop (imagine that this sprite has been looping while NetFlash was downloading).

```
on exitFrame
    if member("NetFlash").percentStreamed = 100 then
        sprite(6).loop = FALSE
        go to the frame + 1
    end if
    go to the frame
end
```

loopBounds

Syntax `sprite(whichQuickTimeSprite).loopBounds`

the loopBounds of sprite *whichQuickTimeSprite*

Description QuickTime sprite property; sets the internal loop points for a QuickTime cast member or sprite. The loop points are specified as a Director list: `[startTime, endTime]`.

The *startTime* and *endTime* parameters must meet these requirements:

- Both parameters must be integers that specify times in Director ticks.
- The values must range from 0 to the duration of the QuickTime cast member.
- The starting time must be less than the ending time.

If any of these requirements is not met, the QuickTime movie loops through its entire duration.

The *loopBounds* property has no effect if the movie's *loop* property is set to FALSE. If the *loop* property is set to TRUE while the movie is playing, the movie continues to play. Director uses these rules to decide how to loop the movie:

- If the ending time specified by *loopBounds* is reached, the movie loops back to the starting time.
- If the end of the movie is reached, the movie loops back to the start of the movie.

If the *loop* property is turned off while the movie is playing, the movie continues to play. Director stops when it reaches the end of the movie.

This property can be tested and set. The default setting is [0,0].

Example This sprite script sets the starting and ending times for looping within a QuickTime sprite. Notice that the times are set by specifying seconds, which are then converted to ticks by multiplying by 60.

```
on beginSprite me
    sprite(me.spriteNum).loopBounds = [(16 * 60),(32 * 60)]
end
```

loopCount

Syntax

```
sound(channelNum).loopCount
```

the loopCount of sound *channelNum*

Description

Cast member property; the total number of times the current sound in sound channel *channelNum* is set to loop. The default is 1 for sounds that are simply queued with no internal loop.

You can loop a portion of a sound by passing the parameters *loopStartTime*, *loopEndTime*, and *loopCount* with a *queue()* or *setPlaylist()* command. These are the only methods for setting this property.

If *loopCount* is set to 0, the loop will repeat forever. If the sound cast member's *loop* property is set to TRUE, the *loopCount* will return 0.

Example

This handler queues and plays two sounds in sound channel 2. The first sound, cast member *introMusic*, loops five times between 8 seconds and 8.9 seconds. The second sound, cast member *creditsMusic*, loops three times. However, no *#loopStartTime* and *#loopEndTime* are specified, so these values default to the *#startTime* and *#endTime*, respectively.

```
on playMusic
    sound(2).queue([#member:member("introMusic"), #startTime:3000,\
        #loopCount:5,#loopStartTime:8000, #loopEndTime:8900])
    sound(2).queue([#member:member("creditsMusic"), #startTime:3000,\
        #endTime:8000, #loopCount:3])
    sound(2).play()
end
```

Example

This handler displays an alert indicating how many times the loop in the cast member of sound 2 plays. If no loop has been set in the current sound of sound channel 2, *sound(2).loopCount* returns 1.

```
on showLoopCount
    alert "The current sound's loop plays" && sound(2).loopCount && "times."
end
```

See also

breakLoop(), *setPlaylist()*, *loopEndTime*, *loopsRemaining*, *loopStartTime*, *queue()*

loopEndTime

Syntax
sound(*channelNum*).loopEndTime
the loopEndTime of sound *channelNum*

Description Sound property; the end time, in milliseconds, of the loop set in the current sound playing in sound channel *channelNum*. Its value is a floating-point number, allowing you to measure and control sound playback to fractions of a millisecond. This property can only be set when passed as a property in a queue() or setPlaylist() command.

Example This handler plays sound cast member introMusic in sound channel 2. Playback loops five times between the 8 seconds point and the 8.9 second point in the sound.

```
on playMusic
    sound(2).play([#member:member("introMusic"), #startTime:3000,\#loopCount:5,#loopStartTime:8000, #loopEndTime:8900])
end
```

Example This handler causes the text field TimWords to read "Help me, I'm stuck!" when the currentTime of sound channel 2 is between its loopStartTime and loopEndTime.

```
on idle
    if sound(2).currentTime > sound(2).loopStartTime and \
    sound(2).currentTime < sound(2).loopEndTime then
        member("TimWords").text = "Help me, I'm stuck!"
    else
        member("TimWords").text = "What's this sticky stuff?"
    end if
end
```

See also breakLoop(), getPlaylist(), loopCount, loopsRemaining, loopStartTime, queue()

loopsRemaining

Syntax
sound(*channelNum*).loopsRemaining
the loopsRemaining of sound(*channelNum*)

Description Read-only property; the number of times left to play a loop in the current sound playing in sound channel *channelNum*. If the sound had no loop specified when it was queued, this property is 0. If this property is tested immediately after a sound starts playing, it returns one less than the number of loops defined with the #loopCount property in the queue() or setPlaylist() command.

See also breakLoop(), loopCount, loopEndTime, loopStartTime, queue()

loopStartTime

Syntax	sound(<i>channelNum</i>).loopStartTime the loopStartTime of sound(<i>channelNum</i>)
Description	Cast member property; the start time, in milliseconds, of the loop for the current sound being played by <i>soundObject</i> . Its value is a floating-point number, allowing you to measure and control sound playback to fractions of a millisecond. The default is the startTime of the sound if no loop has been defined.
	This property can only be set when passed as a property in a queue() or setPlaylist() command.
Example	This handler plays sound cast member introMusic in sound channel 2. Playback loops five times between two points 8 seconds and 8.9 seconds into the sound. <pre>on playMusic sound(2).play([#member:member("introMusic"), #startTime:3000,\ #loopCount:5,#loopStartTime:8000, #loopEndTime:8900]) end</pre>
Example	This handler causes the text field TimWords to read "Help me, I'm stuck!" when the currentTime of sound channel 2 is between its loopStartTime and loopEndTime: <pre>on idle if sound(2).currentTime > sound(2).loopStartTime and \ sound(2).currentTime < sound(2).loopEndTime then member("TimWords").text = "Help me, I'm stuck!" else member("TimWords").text = "What's this sticky stuff?" end if end</pre>
See also	breakLoop(), setPlaylist(), loopCount, loopEndTime, loopsRemaining, queue()

map()

Syntax	map(<i>targetRect</i> , <i>sourceRect</i> , <i>destinationRect</i>) map(<i>targetPoint</i> , <i>sourceRect</i> , <i>destinationRect</i>)
Description	Function; positions and sizes a rectangle or point based on the relationship of a source rectangle to a target rectangle. The relationship of the targetRect to the sourceRect governs the relationship of the result of the function to the destinationRect.

Example	In this behavior, all of the sprites have already been set to draggable. Sprite 2b contains a small bitmap. Sprite 1s is a rectangular shape sprite large enough to easily contain sprite 2b. Sprite 4b is a larger version of the bitmap in sprite 2b. Sprite 3s is a larger version of the shape in sprite 1s. Moving sprite 2b or sprite 1s will cause sprite 4b to move. When you drag sprite 2b, its movements are mirrored by sprite 4b. When you drag sprite 1s, sprite 4b moves in the opposite direction. Resizing sprite 2b or sprite 1s will also produce interesting results. <pre>on exitFrame sprite(4b).rect = map(sprite(2b).rect, sprite(1s).rect, sprite(3s).rect) go the frame end</pre>
----------------	--

mapMemberToStage()

Syntax	<code>sprite(<i>whichSpriteNumber</i>). mapMemberToStage(<i>whichPointInMember</i>)</code> <code>mapMemberToStage(sprite <i>whichSpriteNumber</i>, <i>whichPointInMember</i>)</code>
Description	Function; uses the specified sprite and point to return an equivalent point inside the dimensions of the Stage. This properly accounts for the current transformations to the sprite using quad, or the rectangle if not transformed. This is useful for determining if a particular area of a cast member has been clicked, even if there have been major transformations to the sprite on the Stage. If the specified point on the Stage is not within the sprite, a VOID is returned.
See also	<code>map()</code> , <code>mapStageToMember()</code>

mapStageToMember()

Syntax	<code>sprite(<i>whichSpriteNumber</i>). mapStageToMember(<i>whichPointOnStage</i>)</code> <code>mapStageToMember(sprite <i>whichSpriteNumber</i>, <i>whichPointOnStage</i>)</code>
Description	Function; uses the specified sprite and point to return an equivalent point inside the dimensions of the cast member. This properly accounts for any current transformations to the sprite using quad, or the rectangle if not transformed. This is useful for determining if a particular area on a cast member has been clicked even if there have been major transformations to the sprite on the Stage. If the specified point on the Stage is not within the sprite, this function returns VOID.

See also `map()`, `mapMemberToStage()`

margin

Syntax member(*whichCastMember*).margin
the margin of member *whichCastMember*

Description Field cast member property; determines the size, in pixels, of the margin inside the field box.

Example The following statement sets the margin inside the box for the field cast member Today's News to 15 pixels:

```
member("Today's News").margin = 15
```

marker()

Syntax marker(*integerExpression*)
marker("string")

Description Function; returns the frame number of markers before or after the current frame. This function is useful for implementing a Next or Previous button or for setting up an animation loop.

The argument *integerExpression* can evaluate to any positive or negative integer or 0. For example:

- marker(2)—Returns the frame number of the second marker after the current frame.
- marker(1)—Returns the frame number of the first marker after the current frame.
- marker(0)—Returns the frame number of the current frame if the current frame is marked, or the frame number of the previous marker if the current frame is not marked.
- marker(-1)—Returns the frame number of the first marker before the marker(0).
- marker(-2)—Returns the frame number of the second marker before the marker(0).

If the argument for marker is a string, marker returns the frame number of the first frame whose marker label matches the string.

Example This statement sends the playback head to the beginning of the current frame if the current frame has a marker, otherwise it sends the playback head to the previous marker:

```
go to marker(0)
```

Example This statement sets the variable nextMarker equal to the next marker in the Score:
`nextMarker = marker(1)`

See also go, frame() (function), frameLabel, label(), labelList

the markerList

Syntax	the markerList
Description	Global property; contains a Lingo property list of the markers in the Score. The list is of the format: frameNumber: "markerName" This property can be tested but not set.
Example	This statement displays the list of markers in the Message window: put the markerlist -- [1: "Opening Credits", 15: "Main Menu", 26: "Closing Credits"] marker()

mask

Syntax	member(<i>whichQuickTimeMember</i>).mask
Description	the mask of member <i>whichQuickTimeMember</i>
Example	Cast member property; specifies a black-and-white (1-bit) cast member to be used as a mask for media rendered direct to Stage with media appearing in the areas where the mask's pixels are black. The mask property lets you benefit from the performance advantages of a direct-to-Stage digital video while playing a QuickTime movie in a nonrectangular area. The mask property has no effect on non-direct-to-Stage cast members. Director always aligns the registration point of the mask cast member with the upper left of the QuickTime movie sprite. Be sure to reset the registration point of a bitmap to the upper left corner, as it defaults to the center. The registration point of the QuickTime member cannot be reset from the upper left corner. The mask cast member can't be moved and is not affected by the center and crop properties of its associated cast member. For best results, set a QuickTime cast member's mask property before any of its sprites appear on the Stage in the on beginSprite event handler. Setting or changing the mask property while the cast member is on the Stage can have unpredictable results (for example, the mask may appear as a freeze frame of the digital video at the moment the mask property took effect). Masking is an advanced feature; you may need to experiment to achieve your goal. This property can be tested and set. To remove a mask, set the mask property to 0. This frame script sets a mask for a QuickTime sprite before Director begins to draw the frame: on prepareFrame member("Peeping Tom").mask = member("Keyhole") end
See also	invertMask

max()

Syntax list.max()
max(*list*)
max(*value1, value2, value3, ...*)

Description Function; returns the highest value in the specified list or the highest of a given series of values.

The max function also works with ASCII characters, similar to the way < and > operators work with strings.

Example This handler assigns the variable Winner the maximum value in the list Bids, which consists of [#Castle:600, #Schmitz:750, #Wang:230]. The result is then inserted into the content of the field cast member Congratulations.

```
on findWinner Bids
    Winner = Bids.max()
    member("Congratulations").text = \
        "You have won, with a bid of $" & Winner &"!"
```

maxInteger

Syntax the maxInteger

Description System property; returns the largest whole number that is supported by the system. On most personal computers, this is 2,147,483,647 (2 to the thirty-first power, minus 1).

This property can be useful for initializing boundary variables before a loop or for limit testing.

To use numbers larger than the range of addressable integers, use floating-point numbers instead. They aren't processed as quickly as integers, but they support a greater range of values.

Example This statement generates a table, in the Message window, of the maximum decimal value that can be represented by a certain number of binary digits:

```
on showMaxValues
    b = 31
    v = the maxInteger
    repeat while v > 0
        put b && "-" && v
        b = b-1
        v = v/2
    end repeat
end
```

mci

Syntax mci "string"

Description Command; for Windows only, passes the strings specified by *string* to the Windows Media Control Interface (MCI) for control of multimedia extensions.

Note: Microsoft no longer recommends using the 16-bit MCI interface. Consider using third-party Xtras for this functionality instead.

Example This statement makes the command play cdaudio from 200 to 600 track 7 play only when the movie plays back in Windows:

```
mci "play cdaudio from 200 to 600 track 7"
```

me

Syntax me

Description Special variable; used within parent scripts and behaviors to refer to the current object that is an instance of the parent script or the behavior or a variable that contains the memory address of the object.

The term has no predefined meaning in Lingo. The term me is used by convention.

To see an example of me used in a completed movie, see the Parent Scripts movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement sets the object myBird1 to the script named Bird. The me keyword accepts the parameter script Bird and is used to return that parameter.

```
myBird1 = new(script "Bird")
```

This is the on new handler of the Bird script:

```
on new me  
    return me  
end
```

Example The following two sets of handlers make up a parent script. The first set uses me to refer to the child object. The second set uses the variable myAddress to refer to the child object. In all other respects, the parent scripts are the same.

This is the first set:

```
property myData  
  
on new me, theData  
    myData = theData  
    return me  
end
```

```
on stepFrame me  
    ProcessData me  
end
```

This is the second set:

```
property myData  
  
on new myAddress, theData  
    myData = theData  
    return myAddress  
end
```

```
on stepFrame myAddress  
    ProcessData myAddress  
end
```

See also [new\(\)](#), [ancestor](#)

media

Syntax `member(whichCastMember).media`

the media of member *whichCastMember*

Description Cast member property; identifies the specified cast member as a set of numbers. Because setting the media member property can use large amounts of memory, this property is best used during authoring only.

You can use the media member property to copy the content of one cast member into another cast member by setting the second cast member's media value to the media value for the first cast member.

For a film loop cast member, the media member property specifies a selection of frames and channels in the Score.

To swap media in a projector, it's more efficient to set the member sprite property.

Example This statement copies the content of the cast member Sunrise into the cast member Dawn by setting the media member property value for Dawn to the media member property value for Sunrise:

```
member("Dawn").media = member("Sunrise").media
```

See also [type \(cast member property\)](#), [media](#)

mediaReady

Syntax

```
member(whichCastMember).mediaReady  
the mediaReady of member whichCastMember  
sprite(whichSpriteNumber).mediaReady  
the mediaReady of sprite whichSpriteNumber
```

Description

Cast member or sprite property; determines whether the contents of the sprite, the specified cast member, or a movie or cast file, or linked cast member is downloaded from the Internet and is available on the local disk (TRUE) or is not available (FALSE).

This property is useful only when streaming a movie or cast file. Movie streaming is activated by setting the Movie:Playback properties in the Modify menu to Play While Downloading Movie (default setting).

For a demonstration of the mediaReady member function, see the sample movie "Streaming Shockwave" in Director Help.

This property can be tested but not set.

Example

This statement changes cast members when the desired cast member is downloaded and available locally:

```
if member("background").mediaReady = TRUE then  
    sprite(2).memberNum = 10  
    -- 10 is the number of cast member "background"  
end if
```

See also

[frameReady\(\)](#)

member (keyword)

Syntax

```
member whichCastMember  
member whichCastMember of castLib whichCast  
member(whichCastMember, whichCastLib)
```

Description

Keyword; indicates that the object specified by *whichCastMember* is a cast member. If *whichCastMember* is a string, it is used as the cast member name. If *whichCastMember* is an integer, it is used as the cast member number.

When playing back a movie as an applet, refer to cast members by number rather than by name to improve the applet's performance.

The member keyword is a specific reference to both a castLib and a member within it if used alone:

```
put sprite(12).member  
-- (member 3 of castLib 2)
```

This property differs from the memberNum property of a sprite, which is always an integer designating position in a castLib but does not specify the castLib:

```
put sprite(12).memberNum  
-- 3
```

The number of a member is also an absolute reference to a particular member in a particular castLib:

```
put sprite(12).member.number  
-- 131075
```

Example The following statement sets the hilite property of the button cast member named Enter Bid to TRUE:

```
member("Enter Bid").hilite = TRUE
```

Example This statement puts the name of sound cast member 132 into the variable soundName:

```
put member(132, "Viva Las Vegas").name
```

Example This statement checks the type of member Jefferson Portrait in the castLib Presidents:

```
memberType = member("Jefferson Portrait", "Presidents").type
```

Example This statement determines whether cast member 9 has a name assigned:

```
if member(9).name = EMPTY then exit
```

Example You can check for the existence of a member by testing for its number:

```
memberCheck = member("Epiphany").number  
if memberCheck = -1 then alert "Sorry, that member doesn't exist"
```

Example Alternatively, you can check for the existence of a member by testing for its type:

```
memberCheck = member("Epiphany").type  
if memberCheck = #empty then alert "Sorry, that member doesn't exist"
```

See also memberNum

member (sound property)

Syntax sound(*channel/Num*).member
the member of sound(*channel/Num*)

Description This read-only property is the sound cast member currently playing in sound channel *channel/Num*. This property returns null if no sound is being played.

Example This statement displays the name of the member of the sound playing in sound channel 2 in the message window.

```
put sound(2).member  
-- (member 4 of castLib 1)
```

See also getPlaylist(), queue()

member (sprite property)

Syntax

`sprite(whichSprite).member`

the member of sprite *whichSprite*

Description

Sprite property; specifies a sprite's cast member and cast.

The member sprite property differs from the memberNum sprite property, which specifies only the sprite's number to identify its location in the cast but doesn't specify the cast itself. The member sprite property also differs from mouseMember and the obsolete castNum sprite properties, neither of which specifies the sprite's cast.

When assigning a sprite's member property, use one of the following formats:

- Specify the full member and cast description (`sprite(x).member = member(A, B)`).
- Specify the cast member name (`sprite(x).member = member ("MELODY")`).
- Specify the unique integer that includes all cast libraries and corresponds to the mouseMember function (`sprite(x).member = 132`).

If you use only the cast member name, Director finds the first cast member that has that name in all current casts. If the name is duplicated in two casts, only the first name is used.

To specify a cast member by number when there are multiple casts, use the memberNum sprite property, which changes the member's position in its cast without affecting the sprite's cast (set the memberNum of sprite x to 132).

You can determine the memberNum sprite property from the member sprite property by using the phrase the number of the member of sprite x. You can also retrieve other cast member properties by using phrases such as the name of the member of sprite x or the rect of the member of sprite x.

The cast member assigned to a sprite channel is only one of that sprite's properties; other properties vary by the type of media element in that channel in the Score. For example, if you replace a bitmap with an unfilled shape by setting the member sprite property, the shape sprite's lineSize sprite property doesn't automatically change, and you probably won't see the shape.

Similar sprite property mismatches can occur if you change the member of a field sprite to a video. Although you can change all sprite properties through the type sprite property, it's generally more useful and predictable to replace cast members with similar cast members. For example, replace bitmap sprites with bitmap cast members.

This property can be tested and set.

Example

This statement assigns cast member 3 of cast number 4 to sprite 15:

`sprite(15).member = member(3, 4)`

Example The following handler uses the mouseMember function with the sprite.member property to find if the mouse is over a particular sprite:

```
on exitFrame
    MM = the mouseMember
    target = sprite(1).member
    if target = MM then put "above the hotspot"
        go the frame
    end
```

See also [castLibNum](#), [memberNum](#)

memberNum

Syntax `sprite(whichSprite).memberNum`

the memberNum of sprite *whichSprite*

Description Sprite property; identifies the position of the cast member (but doesn't identify the castLib) associated with the specified sprite *whichSprite*. Its value is the cast member number only; the cast member's cast is not specified.

The memberNum property is useful for switching cast members assigned to a sprite so long as the cast members are within the same cast. To switch among cast members in different casts, use the member sprite property. For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet.

This property also is useful for exchanging cast members when a sprite is clicked to simulate the reversed image that appears when a standard button is clicked. You can also make some action in the movie depend on which cast member is assigned to a sprite.

When you set this property within a script while the playback head is not moving, be sure to use the updateStage command to redraw the Stage.

This property can be tested and set.

Example The following statement switches the cast member assigned to sprite 3 to cast member number 35 in the same cast:

```
sprite(3).memberNum = 35
```

Example This statement assigns the cast member Narrator to sprite 10 by setting memberNum sprite property to Narrator's cast number. Narrator is in the same cast as the sprite's current cast member.

```
sprite(10).memberNum = member("Narrator").number
```

Example This handler swaps bitmaps when a button is clicked or rolled off. It assumes that the artwork for the Down button immediately follows the artwork for the Up button in the same cast.

```
on mouseDown
    upButton = sprite(the clickOn).memberNum
    downButton = upButton + 1
    repeat while the stillDown
        if rollover(the clickOn) then
            sprite(the clickOn).memberNum = downButton
        else
            sprite(the clickOn).memberNum = upButton
        end if
        updateStage
    end repeat
    if rollover (the clickOn) then put "The button was activated"
end
```

See also [castLib](#), [member \(sprite property\)](#), [number \(cast member property\)](#), [member \(keyword\)](#)

members

See [number of members](#)

memorySize

Syntax the memorySize

Description System property; returns the total amount of memory allocated to the program, whether in use or free memory. This property is useful for checking minimum memory requirements. The value is given in bytes.

In Windows, the value is the total physical memory available; on the Macintosh, the value is the entire partition assigned to the application.

Example This statement checks whether the computer allocates less than 500K of memory and if it does, displays an alert:

```
if the memorySize < 500 * 1024 then alert "There is not enough memory to run this movie."
```

See also [freeBlock\(\)](#), [freeBytes\(\)](#), [ramNeeded\(\)](#), [size](#)

menu

Syntax

menu: menuName

itemName | script
itemName | script

...

or

menu: menuName

itemName | script

itemName | script

...

[more menus]

Description

Keyword; in conjunction with the installMenu command, specifies the actual content of custom menus. Field cast members contain menu definitions; refer to them by the cast member name or number.

The menu keyword is followed immediately by a colon, a space, and the name of the menu. In subsequent lines, specify the menu items for that menu. You can set a script to execute when the user chooses an item by placing the script after the vertical bar symbol (|). A new menu is defined by the subsequent occurrence of the menu keyword.

Note: Menus are not available in Shockwave.

On the Macintosh, you can use special characters to define custom menus. These special characters are case sensitive. For example, to make a menu item bold, the letter *B* must be uppercase.

Special symbols should follow the item name and precede the vertical bar symbol (|). You can also use more than one special character to define a menu item. Using U, for example, sets the style to Bold and Underline.

Avoid special character formatting for cross-platform movies because not all Windows computers support it.

Symbol	Example	Description
@	menu: @	*On the Macintosh, creates the Apple symbol and enables Macintosh menu bar items when you define an Apple menu.
!Ã	!ÃEasy Select	*On the Macintosh, checks the menu with a check mark (Option+v).
<B	Bold<B	*On the Macintosh, sets the menu item's style to Bold.
<I	Italic<I	*On the Macintosh, sets the style to Italic.
<U	Underline<U	*On the Macintosh, sets the style to Underline.

Symbol	Example	Description
<O	Outline<O	*On the Macintosh, sets the style to Outline.
<S	Shadow<S	*On the Macintosh, sets the style to Shadow.
	Open/O go to frame "Open"	Associates a script with the menu item.
/	Quit/Q	Defines a command-key equivalent.
(Save(Disables the menu item.
(-	(-	Creates a disabled line in the menu.

* identifies formatting tags that work only on the Macintosh.

Example This is the text of a field cast member named CustomMenu2 which can be used to specify the content of a custom File menu. To install this menu, use “installMenu member(“CustomMenu2”)” while the movie is running. The Convert menu item runs the custom handler convertThis.

```
menu: File
Open/O | go to frame "Open"
Close/W | go to frame "Close"
Convert/C | convertThis
(
)
Quit/Q | go to frame "Quit"
```

See also installMenu, name (menu property), name (menu item property), number (menu items), checkMark, enabled, script

milliseconds

Syntax the milliseconds

Description System property; returns the current time in milliseconds (1/1000 of a second). Counting begins from the time the computer is started.

Example This statement converts milliseconds to seconds and minutes by dividing the number of milliseconds by 1000 and dividing that result by 60, and then sets the variable currentMinutes to the result:

```
currentSeconds = milliseconds/1000
currentMinutes = currentseconds/60
```

The resolution accuracy of the count is machine and operating system dependent.

Example	This handler counts the milliseconds and posts an alert if you've been working too long. on idle if the milliseconds > 1000 * 60 * 60 * 4 then alert "Take a break" end if end
See also	ticks, time(), timer

min

Syntax	list.min min(<i>list</i>) min (<i>a1, a2, a3...</i>)
Description	Function; specifies the minimum value in the list specified by <i>list</i> .
Example	This handler assigns the variable vLowest the minimum value in the list bids, which consists of [#Castle:600, #Shields:750, #Wang:230]. The result is then inserted in the content of the field cast member Sorry: on findLowest bids vLowest = bids.min() member("Sorry").text = \ "We're sorry, your bid of \$" & vLowest && "is not a winner!" end
See also	max()

missingFonts

Syntax	member(<i>textCastMember</i>).missingFonts
Description	Text cast member property; this property contains a list of the names of the fonts that are referenced in the text, but not currently available on the system.
	This allows the developer to determine during run time if a particular font is available or not.
	This property can be tested but not set.

See also substituteFont

mod

Syntax

integerExpression1 mod integerExpression2

Description

Math operator; performs the arithmetic modulus operation on two integer expressions. In this operation, *integerExpression1* is divided by *integerExpression2*.

The resulting value of the entire expression is the integer remainder of the division. It always has the sign of *integerExpression1*.

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

Example

This statement divides 7 by 4 and then displays the remainder in the Message window:

```
put 7 mod 4
```

The result is 3.

Example

This handler sets the ink effect of all odd-numbered sprites to copy, which is the ink effect specified by the number 0. First the handler checks whether the sprite in the variable *mySprite* is an odd-numbered sprite by dividing the sprite number by 2 and then checking whether the remainder is 1. If the remainder is 1, the result for an odd-numbered number, the handler sets the ink effect to copy.

```
on setInk
    repeat with mySprite = 1 to the lastChannel
        if (mySprite mod 2) = 1 then
            sprite(mySprite).ink = 0
        else
            sprite(mySprite).ink = 8
        end if
    end repeat
end setInk
```

Example

This handler regularly cycles a sprite's cast member among a number of bitmaps:

```
on exitFrame
    global gCounter
    -- These are sample values for bitmap cast member numbers
    theBitmaps = [2,3,4,5,6,7]
    -- Specify which sprite channel is affected
    theChannel = 1
    -- This cycles through the list
    gCounter = 1 + (gCounter mod theBitmaps.count)
    sprite(theChannel).memberNum = theBitmaps[gCounter]
    go the frame
end
```

modal

Syntax

window "window".modal

the modal of window "window"

Description

Window property; specifies whether movies can respond to events that occur outside the window specified by *window*.

- When the modal window property is TRUE, movies cannot respond to events outside the window.
- When the modal window property is FALSE, movies can respond to events outside the window.

Setting the modal window property to TRUE lets you make a specific movie in a window the only movie that the user can interact with.

Be aware that this property works even in the authoring environment. If you set the modal window property to TRUE, you will not be able to interact with Director's windows either.

You can always close a window that is modal by using Control+Alt+period (Windows) or Command+period (Macintosh).

Example

This statement lets movies respond to events outside of the Tool Panel window:

```
window("Tool Panel").modal = FALSE
```

modified

Syntax

member(*whichCastMember*).modified

the modified of member *whichCastMember*

Description

Cast member property; indicates whether the cast member specified by *whichCastMember* has been modified since it was read from the movie file.

- When the modified member property is TRUE (1), the cast member has been modified since it was read from the movie file.
- When the modified member property is FALSE (0), the cast member has not been modified since it was read from the movie file.

The modified member function always returns FALSE for a cast member in a movie that plays back as an applet.

Example

This statement tests whether the cast member Introduction has been modified since it was read from the movie file:

```
return member("Introduction").modified
```

modifiedBy

Syntax *member.modifiedBy*

the modifiedBy of *member*

Description Cast member property; records the name of the user who last edited the cast member, taken from the user name information provided during Director installation. You can change this information in Director's General Preferences dialog box.

This property can be tested but not set. It is useful for tracking and coordinating Director projects with more than one author, and may also be viewed in the Property Inspector's Member tab.

Example This statement displays the name of the person who last modified cast member 1:

```
put member(1).modifiedBy  
-- "Sam Sein"
```

See also comments, creationDate, modifiedDate

modifiedDate

Syntax *member.modifiedDate*

the modifiedDate of *member*

Description Cast member property; indicates the date and time that the cast member was last changed, using the system time on the authoring computer. This property is useful for tracking and coordinating Director projects.

This property can be tested but not set. It can also be viewed in the Property Inspector's Member tab and the Cast window list view.

Example This statement displays the date of the last change to cast member 1:

```
put member(1).modifiedDate  
-- date( 1999, 12, 8 )
```

See also comments, creationDate, modifiedBy

mostRecentCuePoint

Syntax	<pre>sprite(<i>whichSprite</i>).mostRecentCuePoint</pre> <p>the mostRecentCuePoint of sprite <i>whichSprite</i></p> <pre>sound(<i>channel/Num</i>).mostRecentCuePoint</pre> <p>the mostRecentCuePoint of sound <i>channel/Num</i></p>
Description	<p>Cast member, sound channel, and sprite property; for sound cast members, QuickTime digital video, and Xtras that support cue points, indicates the number that identifies the most recent cue point passed in the sprite or sound. The value is the cue point's ordinal number. If no cue points have been passed, the value is 0.</p> <p>For sound channels, the return value is for the sound cast member currently playing in the sound channel.</p> <p>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.</p>
Example	<p>This statement tells the Message window to display the number for the most recent cue point passed in the sprite in sprite channel 1:</p> <pre>put sprite(1).mostRecentCuePoint</pre>
Example	<p>This statement returns the ordinal number of the most recently passed cue point in the currently playing sound in sound channel 2:</p> <pre>put sound(2).mostRecentCuePoint</pre>
See also	<p><code>cuePointNames</code>, <code>isPastCuePoint()</code></p>

motionQuality

Syntax	<pre>sprite(<i>whichQTVRSprite</i>).motionQuality</pre> <p><code>motionQuality</code> of sprite <i>whichQTVRSprite</i></p>
Description	<p>QuickTime VR sprite property; the codec quality used when the user clicks and drags the QuickTime VR sprite. The property's value can be <code>#minQuality</code>, <code>#maxQuality</code>, or <code>#normalQuality</code>.</p> <p>This property can be tested and set.</p>
Example	<p>This sets the motionQuality of sprite 1 to <code>#minQuality</code>.</p> <pre>sprite(1).motionQuality = #minQuality</pre>

mouseChar

Syntax the mouseChar

Description System property; used for field sprites, contains the number of the character that is under the cursor when the property is called. The count is from the beginning of the field. If the mouse is not over a field or is in the gutter of a field, the result is -1.

The value of the mouseChar property can change in a handler or repeat loop. If a handler or repeat loop uses this property multiple times, it's usually a good idea to call the property once and assign its value to a local variable.

Example This statement determines whether the cursor is over a field sprite and changes the content of the field cast member Instructions to "Please point to a character." when it is:

```
if the mouseChar = -1 then  
    member("Instructions").text = "Please point to a character."
```

Example This statement assigns the character under the cursor in the specified field to the variable currentChar:

```
currentChar = member(the mouseMember).char[the mouseChar]
```

Example This handler highlights the character under the cursor when the sprite contains a text cast member:

```
property spriteNum  
  
on mouseWithin me  
    if sprite(spriteNum).member.type = #field then  
        MC = the mousechar  
        if MC < 1 then exit -- if over a border, final line, etc  
        hilite char MC of field sprite(spriteNum).member  
        else alert "Sorry, 'hilite' and 'mouseChar' are for fields."  
    end
```

See also mouseItem, mouseLine, mouseWord, char...of, number (characters)

on mouseDown (event handler)

Syntax

```
on mouseDown  
    statement(s)  
end
```

Description

System message and event handler; contains statements that run when the mouse button is pressed.

When the mouse button is pressed, Lingo searches the following locations, in order, for an on mouseDown handler: primary event handler, sprite script, cast member script, frame script, and movie script. Lingo stops searching when it reaches the first location that has an on mouseDown handler, unless the handler includes the pass command to explicitly pass the mouseDown message on to the next location.

To have the same response throughout the movie when pressing the mouse button, set mouseDownScript or put a mouseDown handler in a Movie script.

The on mouseDown event handler is a good place to put Lingo that flashes images, triggers sound effects, or makes sprites move when the user presses the mouse button.

Where you place an on mouseDown handler can affect when it runs.

- To apply the handler to a specific sprite, put it in a sprite script.
- To apply the handler to a cast member in general, put it in a cast member script.
- To apply the handler to an entire frame, put it in a frame script.
- To apply the handler throughout the entire movie, put it in a movie script.

You can override an on mouseDown handler by placing an alternative on mouseDown handler in a location that Lingo checks before it gets to the handler you want to override. For example, you can override an on mouseDown handler assigned to a cast member by placing an on mouseDown handler in a sprite script.

If used in a behavior, this event is passed the sprite script or frame script reference me.

Example

This handler checks whether the user clicks anywhere on the Stage and sends the playback head to another frame if a click occurs:

```
on mouseDown  
    if the clickOn = 0 then go to frame "AddSum"  
end
```

Example

This handler, assigned to a sprite script, plays a sound when the sprite is clicked:

```
on mouseDown  
    puppetSound "Crickets"  
end
```

See also

clickOn, mouseDownScript, mouseUpScript

the mouseDown (system property)

Syntax the mouseDown

Description System property; indicates whether the mouse button is currently being pressed (TRUE) or not (FALSE).

The Director player for Java doesn't update the mouseDown property when Lingo is in a repeat loop. If you try to test mouseDown inside a repeat loop in an applet, the applet hangs.

Example This handler causes the movie to beep until the user clicks the mouse:

```
on enterFrame
    repeat while the mouseDown = FALSE
        beep
    end repeat
end
```

Example This statement instructs Lingo to exit the repeat loop or handler it is in when the user clicks the mouse:

```
if the mouseDown then exit
```

See also mouseH, the mouseUp (system property), mouseV, on mouseDown (event handler), on mouseUp (event handler)

mouseDownScript

Syntax the mouseDownScript

Description System property; specifies the Lingo that is executed when the mouse button is pressed. The Lingo is written as a string, surrounded by quotation marks and can be a simple statement or a calling script for a handler. The default value is EMPTY, which means that the mouseDownScript property has no Lingo assigned to it.

When the mouse button is pressed and the mouseDownScript property is defined, Lingo executes the instructions specified for the mouseDownScript property first. No other on mouseDown handlers are executed, unless the instructions include the pass command so that the mouseDown message can be passed to other objects in the movie.

Setting the mouseDownScript property performs the same function as the when keyDown then command in earlier versions of Director.

To turn off the instructions you've specified for the mouseDownScript property, use the statement set the mouseDownScript to empty.

This property can be tested and set.

- Example** In this statement, when the user clicks the mouse button, the playback head always branches to the next marker in the movie:
- ```
the mouseDownScript = "go next"
```
- Example** In this statement, when the user clicks anywhere on the Stage, the computer beeps:
- ```
the mouseDownScript = "if the clickOn = 0 then beep"
```
- Example** This statement sets mouseDownScript to the custom handler *myCustomHandler*. A Lingo custom handler must be enclosed in quotation marks when used with the mouseDownScript property.
- ```
the mouseDownScript = "myCustomHandler"
```
- See also** stopEvent, mouseUpScript, on mouseDown (event handler), on mouseUp (event handler)

## on mouseEnter

- Syntax**
- ```
on mouseEnter
  statement(s)
end
```
- Description** System message and event handler; contains statements that run when the mouse pointer first contacts the active area of the sprite. The mouse button does not have to be pressed.
- If the sprite is a bitmap cast member with matte ink applied, the active area is the portion of the image that is displayed; otherwise, the active area is the sprite's bounding rectangle.
- If used in a behavior, this event is passed the sprite script or frame script reference me.
- Example** This statement is a simple button behavior that switches the bitmap of the button when the mouse rolls over and then off the button:
- ```
property spriteNum
on mouseEnter me
 -- Determine current cast member and switch to next in cast
 currentMember = sprite(spritenum).member.number
 sprite(spritenum).member = currentMember + 1
end
on mouseLeave me
 -- Determine current cast member and switch to previous in cast
 currentMember = sprite(spritenum).member.number
 sprite(spritenum).member = currentMember - 1
end
```
- See also** on mouseLeave, on mouseWithin

## mouseH

**Syntax**      the mouseH

          mouseH()

**Description**      System property and function; indicates the horizontal position of the mouse pointer. The value of mouseH is the number of pixels the cursor is positioned from the left edge of the Stage.

The mouseH function is useful for moving sprites to the horizontal position of the mouse pointer and checking whether the pointer is within a region of the Stage. Using the mouseH and mouseV functions together, you can determine the cursor's exact location.

The Director player for Java doesn't update the mouseH function when Lingo is in a repeat loop.

This function can be tested but not set.

**Example**      This handler moves sprite 10 to the mouse pointer location and updates the Stage when the user clicks the mouse button:

```
on mouseDown
 sprite(10).locH = the mouseH
 sprite(10).locV = the mouseV
 updateStage
end
```

**Example**      This statement tests whether the cursor is more than 10 pixels to the right or left of a starting point and if it is, sets the variable Far to TRUE:

```
if abs(mouseH() - startH) > 10 then
 draggedEnough = TRUE
```

**See also**      locH, locV, mouseV, mouseLoc

## mouseltem

**Syntax**      the mouseltem

**Description**      System property; contains the number of the item under the pointer when the function is called and the cursor is over a field sprite. (An item is any sequence of characters delimited by the current delimiter as set by the itemDelimiter property.) Counting starts at the beginning of the field. If the mouse is not over a field, the result is -1.

The value of the mouseltem property can change in a handler or repeat loop. If a handler or repeat loop relies on the initial value of mouseltem when the handler or repeat loop begins, call the property once and assign its value to a local variable.

**Example** This statement determines whether the cursor is over a field sprite and changes the content of the field cast member Instructions to “Please point to an item.” when it is not:

```
if the mouseItem = -1 then
 member("Instructions") = "Please point to an item."
end if
```

**Example** This statement assigns the item under the cursor in the specified field to the variable currentItem:

```
currentItem = member(mouseMember).item(mouseItem)
```

**Example** This handler highlights the item under the cursor when the mouse button is pressed:

```
on mouseDown
 thisField = sprite(the clickOn).member
 if the mouseItem < 1 then exit
 lastItem = 0
 repeat while the stillDown
 MI = the mouseItem
 if MI < 1 then next repeat
 if MI <> lastItem then
 thisField.item[MI].hilite()
 lastItem = MI
 end if
 end repeat
end
```

**See also** item...of, mouseChar, mouseLine, mouseWord, number (items), itemDelimiter

## on mouseLeave

**Syntax** on mouseLeave  
    *statement(s)*  
end

**Description** System message and event handler; contains statements that run when the mouse leaves the active area of the sprite. The mouse button does not have to be pressed.

If the sprite is a bitmap cast member with the matte ink applied, the active area is the portion of the image that is displayed; otherwise, the active area is the sprite’s bounding rectangle.

If used in a behavior, this event is passed the sprite script or frame script reference me.

**Example** This statement shows a simple button behavior that switches the bitmap of the button when the mouse rolls over and then back off the button:

```
property spriteNum
on mouseEnter me
 -- Determine current cast member and switch to next in cast
 currentMember = sprite(spriteNum).member.number
 sprite(spriteNum).member = currentMember + 1
end
on mouseLeave me
 -- Determine current cast member and switch to previous in cast
 currentMember = sprite(spriteNum).member.number
 sprite(spriteNum).member = currentMember - 1
end
```

**See also** on mouseEnter, on mouseWithin

## mouseLevel

**Syntax** sprite(*whichQuickTimeSprite*).mouseLevel

the mouseLevel of sprite *whichQuickTimeSprite*

**Description** QuickTime sprite property; controls how Director passes mouse clicks on a QuickTime sprite to QuickTime. The ability to pass mouse clicks within the sprite's bounding rectangle can be useful for interactive media such as QuickTime VR. The mouseLevel sprite property can have these values:

- #controller—Passes clicks only on the movie controller to QuickTime. Director responds only to mouse clicks that occur outside the controller. This is the standard behavior for QuickTime sprites other than QuickTime VR.
- #all—Passes all mouse clicks within the sprite's bounding rectangle to QuickTime. No clicks pass to other Lingo handlers.
- #none—Does not pass any mouse clicks to QuickTime. Director responds to all mouse clicks.
- #shared—Passes all mouse clicks within a QuickTime VR sprite's bounding rectangle to QuickTime and then passes these events to Lingo handlers. This is the default value for QuickTime VR.

This property can be tested and set.

**Example** This frame script checks to see if the name of the QuickTime sprite in channel 5 contains the string "QTVR." If it does, this script sets mouseLevel to #all; otherwise, it sets mouseLevel to #none.

```
on prepareFrame
 if sprite(5).member.name contains "QTVR" then
 sprite(5).mouseLevel = #all
 else
 sprite(5).mouseLevel = #none
 end if
end
```

## mouseLine

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | the mouseLine                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Description</b> | System property; contains the number of the line under the pointer when the property is called and the cursor is over a field sprite. Counting starts at the beginning of the field; a line is defined by Return delimiter, not by the wrapping at the edge of the field. When the mouse is not over a field sprite, the result is -1. The value of the mouseLine property can change in a handler or repeat loop. If a handler or repeat loop uses this property multiple times, it's usually a good idea to call the property once and assign its value to a local variable. |
| <b>Example</b>     | 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:<br><pre>if the mouseLine = -1 then     member("Instructions").text = "Please point to a line."</pre>                                                                                                                                                                                                                                                                                            |
| <b>Example</b>     | This statement assigns the contents of the line under the cursor in the specified field to the variable currentLine:<br><pre>currentLine = member(the mouseMember).line[the mouseLine]</pre>                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Example</b>     | This handler highlights the line under the cursor when the mouse button is pressed:<br><pre>on mouseDown     thisField = sprite(the clickOn).member     if the mouseLine &lt; 1 then exit     lastLine = 0     repeat while the stillDown         ML = the mouseLine         if ML &lt; 1 then next repeat         if ML &lt; &gt; lastLine then             thisField.line[ML].hilite()             lastLine = ML         end if     end repeat end</pre>                                                                                                                     |
| <b>See also</b>    | mouseChar, mouseItem, mouseWord, line...of, number (lines)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

## mouseLoc

|                    |                                                                                                                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | the mouseLoc                                                                                                                                                                      |
| <b>Description</b> | Function; returns the current position of the mouse as a point(). The point location is given as two coordinates, with the horizontal location first, then the vertical location. |
| <b>See also</b>    | mouseH, mouseV                                                                                                                                                                    |

## mouseMember

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | the mouseMember                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Description</b> | System property; returns the cast member assigned to the sprite that is under the pointer when the property is called. When the pointer is not over a sprite, it returns the result VOID.<br><br>The mouseMember property replaces mouseCast, used in earlier versions of Director. You can use this function to make a movie perform specific actions when the pointer rolls over a sprite and the sprite uses a certain cast member.<br><br>The value of the mouseMember property can change frequently. To use this property multiple times in a handler with a consistent value, assign the mouseMember value to a local variable when the handler starts and use the variable.<br><br>For casts other than cast 1, mouseMember returns a value that does not distinguish between the cast member and the cast number. To distinguish the cast member and the cast number, use the expression member (the mouseMember); if the user doesn't click a sprite, however, this expression generates a script error. |
| <b>Example</b>     | This statement checks whether the cast member Off Limits is the cast member assigned to the sprite under the cursor and displays an alert if it is. This example shows how you can specify an action based on the cast member assigned to the sprite.<br><br>if the mouseMember = member "Off Limits" then alert "Stay away from there!"                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Example</b>     | This statement assigns the cast member of the sprite under the cursor to the variable lastMember:<br><br>lastMember = the mouseMember                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>See also</b>    | member (sprite property), memberNum, clickLoc, mouseChar, mouseItem, mouseLine, mouseWord, rollOver(), number (cast member property)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

## mouseOverButton

|                    |                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | sprite <i>whichFlashSprite</i> .mouseOverButton<br><br>the mouseOverButton of sprite <i>whichFlashSprite</i>                                                                                                                                                                                                                                                                                         |
| <b>Description</b> | Flash sprite property; indicates whether the mouse pointer is over a button in a Flash movie sprite specified by the <i>whichFlashSprite</i> parameter (TRUE), or whether the mouse pointer is outside the bounds of the sprite or the mouse pointer is within the bounds of the sprite but over a nonbutton object, such as the background (FALSE).<br><br>This property can be tested but not set. |

**Example** This frame script checks to see if the mouse pointer is over a navigation button in the Flash movie in sprite 3. If the mouse pointer is over the button, the script updates a text field with an appropriate message; otherwise, the script clears the message.

```
on enterFrame
 case sprite(3).mouseOverButton of
 TRUE:
 member("Message Line").text = "Click here to go to the next page."
 FALSE:
 member("Message Line").text = " "
 end case
 updatestage
end
```

## on mouseUp (event handler)

**Syntax**

```
on mouseUp
 statement(s)
end
```

**Description** System message and event handler; contains statements that are activated when the mouse button is released.

When the mouse button is released, Lingo searches the following locations, in order, for an on mouseUp handler: primary event handler, sprite script, cast member script, frame script, and movie script. Lingo stops searching when it reaches the first location that has an on mouseUp handler, unless the handler includes the pass command to explicitly pass the mouseUp message on to the next location.

To create the same response throughout the movie when the user releases the mouse button, set the mouseUpScript.

An on mouseUp event handler is a good place to put Lingo that changes the appearance of objects—such as buttons—after they are clicked. You can do this by switching the cast member assigned to the sprite after the sprite is clicked and the mouse button is released.

Where you place an on mouseUp handler can affect when it runs, as follows:

- To apply the handler to a specific sprite, put it in a sprite script.
- To apply the handler to a cast member in general, put it in a cast member script.
- To apply the handler to an entire frame, put it in a frame script.
- To apply the handler throughout the entire movie, put it in a movie script.

You can override an on mouseUp handler by placing an alternative on mouseUp handler in a location that Lingo checks before it gets to the handler you want to override. For example, you can override an on mouseUp handler assigned to a cast member by placing an on mouseUp handler in a sprite script.

If used in a behavior, this event is passed the sprite script or frame script reference me.

**Example** This handler, assigned to sprite 10, switches the cast member assigned to sprite 10 when the user releases the mouse button after clicking the sprite:

```
on mouseUp
 sprite(10).member = member "Dimmed"
end
```

**See also** on mouseDown (event handler)

## the mouseUp (system property)

**Syntax** the mouseUp

**Description** System property; indicates whether the mouse button is released (TRUE) or is being pressed (FALSE).

The Director player for Java doesn't update the mouseUp property when Lingo is in a repeat loop.

**Example** This handler causes the movie to run as long as the user presses the mouse button. The playback head stops when the user releases the mouse button.

```
on exitFrame me
 if the mouseUp then
 go the frame
 end if
end
```

**Example** This statement instructs Lingo to exit the repeat loop or handler it is in when the user releases the mouse button:

```
if the mouseUp then exit
```

**See also** the mouseDown (system property), mouseH, mouseV, the mouseUp (system property)

## on mouseUpOutside

**Syntax**      on mouseUpOutside me

```
 statement(s)
end
```

**Description**      System message and event handler; sent when the user presses the mouse button on a sprite but releases it (away from) the sprite.

**Example**      This statement plays a sound when the user clicks the mouse over a sprite and then releases it outside the bounding rectangle of the sprite:

```
on mouseUpOutside me
 puppetSound "Professor Long Hair"
end
```

**See also**      on mouseEnter, on mouseLeave, on mouseWithin

## mouseUpScript

**Syntax**      the mouseUpScript

**Description**      System property; determines the Lingo that is executed when the mouse button is released. The Lingo is written as a string, surrounded by quotation marks, and can be a simple statement or a calling script for a handler.

When the mouse button is released and the mouseUpScript property is defined, Lingo executes the instructions specified for the mouseUpScript property first. Unless the instructions include the pass command so that the mouseUp message can be passed on to other objects in the movie, no other on mouseUp handlers are executed.

When the instructions you've specified for the mouseUpScript property are no longer appropriate, turn them off by using the statement set the mouseUpScript to empty.

Setting the mouseUpScript property accomplishes the same thing as using the when mouseUp then command that appeared in earlier versions of Director.

This property can be tested and set. The default value is EMPTY.

**Example**      When this statement is in effect and the movie is paused, the movie always continues whenever the user releases the mouse button:

```
the mouseUpScript = "go to the frame +1"
```

**Example**      With this statement, when the user releases the mouse button after clicking anywhere on the Stage, the movie beeps:

```
the mouseUpScript = "if the clickOn = 0 then beep"
```

**Example** This statement sets mouseUpScript to the custom handler *myCustomHandler*. A Lingo custom handler must be enclosed in quotation marks when used with the mouseUpScript property.

```
the mouseUpScript = "myCustomHandler"
```

**See also** stopEvent, mouseDownScript, on mouseDown (event handler), on mouseUp (event handler)

## mouseV

**Syntax** the mouseV

```
mouseV()
```

**Description** System property; indicates the vertical position of the mouse cursor, in pixels, from the top of the Stage. The value increases as the cursor moves down and decreases as the cursor moves up.

The mouseV property is useful for moving sprites to the vertical position of the mouse cursor and checking whether the cursor is within a region of the Stage. Using the mouseH and mouseV properties together, you can identify the cursor's exact location.

The Director player for Java doesn't update the mouseV property when Lingo is in a repeat loop.

This property can be tested but not set.

**Example** This handler moves sprite 1 to the mouse pointer location and updates the Stage when the user clicks the mouse button:

```
on mouseDown
 sprite(1).locH = the mouseH
 sprite(1).locV = the mouseV
 updateStage
end
```

**Example** This statement tests whether the pointer is more than 10 pixels above or below a starting point and if it is, sets the variable vFar to TRUE:

```
if abs(the mouseV - startV) > 10 then draggedEnough = TRUE
```

**See also** mouseH, locH, locV, mouseLoc

## on mouseWithin

**Syntax**

```
on mouseWithin
 statement(s)
end
```

**Description** System message and event handler; contains statements that run when the mouse is within the active area of the sprite. The mouse button does not have to be pressed.

If the sprite is a bitmap cast member with the matte ink applied, the active area is the portion of the image that is displayed; otherwise, the sprite's bounding rectangle is the active area.

If used in a behavior, this event is passed the sprite script or frame script reference me.

**Example** This statement displays the mouse location when the mouse is over a sprite:

```
on mouseWithin
 member("Display").text = string(the mouseH)
end mouseWithin
```

**See also** on mouseEnter, on mouseLeave

## mouseWord

**Syntax**

```
the mouseWord
```

**Description** System property; contains the number of the word under the pointer when the property is called and when the pointer is over a field sprite. Counting starts from the beginning of the field. When the mouse is not over a field, the result is -1.

The value of the mouseWord property can change in a handler or repeat loop. If a handler or repeat loop uses this property multiple times, it's usually a good idea to call the function once and assign its value to a local variable.

**Example** This statement determines whether the pointer is over a field sprite and changes the content of the field cast member Instructions to "Please point to a word." when it is not:

```
if the mouseWord = -1 then
 member("Instructions").text = "Please point to a word."
else
 member("Instructions").text = "Thank you."
end if
```

**Example** This statement assigns the number of the word under the pointer in the specified field to the variable currentWord:

```
currentWord = member(the mouseMember).word[the mouseWord]
```

**Example** This handler highlights the word under the pointer when the mouse button is pressed:

```
on mouseDown
 thisField = sprite(the clickOn).member
 if the mouseWord < 1 then exit
 lastWord = 0
 repeat while the stillDown
 MW = the mouseWord
 if MW < 1 then next repeat
 if MW <> lastWord then
 thisField.word[MW].hilite()
 lastWord = MW
 end if
 end repeat
end
```

**See also** [mouseChar](#), [mouseltem](#), [mouseLine](#), [number \(words\)](#), [word...of](#)

## moveableSprite

**Syntax** `sprite(whichSprite).moveableSprite`

the moveableSprite of sprite *whichSprite*

**Description** Sprite property; indicates whether a sprite can be moved by the user (TRUE) or not (FALSE).

You can make a sprite moveable by using the Moveable option in the Score. However, to control whether a sprite is moveable and to turn this condition on and off as needed, use Lingo. For example, to let users drag sprites one at a time and then make the sprites unmoveable after they are positioned, turn the moveableSprite sprite property on and off at the appropriate times.

**Note:** For more customized control such as snapping back to the origin or animating while dragging, create a behavior to manage the additional functionality.

This property can be tested and set.

**Example** This handler makes sprites in channel 5 moveable:

```
on spriteMove
 sprite(5).moveableSprite = TRUE
end
```

**Example** This statement checks whether a sprite is moveable and if it isn't, displays a message:

```
if sprite(13).moveableSprite = FALSE then
 member("Notice").text = "You can't drag this item by using the mouse."
```

**See also** [mouseLoc](#)

## move member

**Syntax**

```
member(whichCastMember).move()
member(whichCastMember).move(member whichLocation)
move member whichCastMember {,member whichLocation}
```

**Description**

Command; moves the cast member specified by *whichCastMember* to the first empty location in the Cast window (if no parameter is set) or to the location specified by *whichLocation*.

For best results, use this command during authoring, not at run time, because the move is usually saved with the file. The actual location of a cast member does not affect most presentations during playback for an end user. To switch the content of a sprite or change the display during run time, set the member of the sprite.

**Example**

This statement moves cast member Shrine to the first empty location in the Cast window:

```
member("shrine").move()
```

**Example**

This statement moves cast member Shrine to location 20 in the Bitmaps Cast window:

```
member("Shrine").move(20, "Bitmaps")
```

## moveToBack

**Syntax**

```
window("whichWindow ").MoveToFront()
moveToFront window whichWindow
```

**Description**

Command; moves the window specified by *whichWindow* behind all other windows.

**Example**

These statements move the first window in *windowList* behind all other windows:

```
myWindow = the windowList[1]
moveToFront myWindow
```

**Example**

If you know the name of the window you want to move, use the syntax:

```
window("Demo Window").moveToFront()
```

## moveToFront

**Syntax**      `window("whichWindow").MoveToFront()`

`moveToFront window whichWindow`

**Description**     Command; moves the window specified by *whichWindow* in front of all other windows.

**Example**      These statements move the first window in *windowList* in front of all other windows:

```
myWindow = the windowList[1]
moveToFront myWindow
```

**Example**      If the you know the name of the window you want to move to the front, use the syntax:

```
window("Demo Window").moveToFront()
```

## moveVertex()

**Syntax**      `member(memberRef). MoveVertex(vertexIndex, xChange, yChange)`

`moveVertex(member memberRef, vertexIndex, xChange, yChange)`

**Description**     Function; moves the vertex of a vector shape cast member to another location.

The horizontal and vertical coordinates for the move are relative to the current position of the vertex point. The location of the vertex point is relative to the origin of the vector shape member.

Changing the location of a vertex affects the shape in the same way as dragging the vertex in an editor.

**Example**      This statement shifts the first vertex point in the vector shape Archie 25 pixels to the right and 10 pixels down from its current position:

```
member("Archie").moveVertex(1, 25, 10)
```

**See also**     `addVertex`, `deleteVertex()`, `moveVertexHandle()`, `originMode`, `vertexList`

## moveVertexHandle()

**Syntax**

```
moveVertexHandle(member memberRef, vertexIndex, handleIndex, xChange, yChange)
```

**Description**

Function; moves the vertex handle of a vector shape cast member to another location.

The horizontal and vertical coordinates for the move are relative to the current position of the vertex handle. The location of the vertex handle is relative to the vertex point it controls.

Changing the location of a control handle affects the shape in the same way as dragging the vertex in the editor.

**Example**

This statement shifts the first control handle of the second vertex point in the vector shape Archie 15 pixels to the right and 5 pixels up:

```
MoveVertexHandle(member "Archie", 2, 1, 15, -5)
```

**See also**

[addVertex](#), [deleteVertex\(\)](#), [originMode](#), [vertexList](#)

## on moveWindow

**Syntax**

```
on moveWindow
```

```
 statement(s)
```

```
end
```

**Description**

System message and event handler; contains statements that run when a window is moved, such as by dragging a movie to a new location on the Stage, and is a good place to put Lingo that you want executed every time a movie's window changes location.

**Example**

This handler displays a message in the Message window when the window a movie is playing in moves:

```
on moveWindow
 put "Just moved window containing"&&the movieName
end
```

**See also**

[activeWindow](#), [movieName](#), [windowList](#)

## movie

This property is obsolete. Use [movieName](#).

## movieAboutInfo

**Syntax** the movieAboutInfo

**Description** Movie property; a string entered during authoring in the Movie Properties dialog box. This property is provided to allow for enhancements in future versions of Shockwave.

This property can be set but not tested.

**See also** movieCopyrightInfo

## movieCopyrightInfo

**Syntax** the movieCopyrightInfo

**Description** Movie property; enters a string during authoring in the Movie Properties dialog box. This property is provided to allow for enhancements in future versions of Shockwave.

This property can be tested but not set.

**Example** This statement displays the copyright information in a text cast member:

```
member("Display").text = "Copyright"&&the movieCopyrightInfo
```

**See also** movieAboutInfo

## movieFileSize

**Syntax** the movieFileSize

**Description** Movie property; returns the number of unused bytes in the current movie caused by changes to the cast members and castLibs within a movie. The Save and Compact and Save As commands rewrite the file to delete this free space.

When the movie has no unused space, the movieFileSize function returns 0.

**Example** This statement displays the number of unused bytes that are in the current movie:  
put the movieFileSize

## movieFileSize

**Syntax** the movieFileSize

**Description** Movie property; returns the number of bytes in the current movie saved on disk. This is the same number returned when selecting File Properties in Windows or Get Info in the Macintosh Finder.

**Example** This statement displays the number of bytes in the current movie:  
put the movieFileSize

## movieFileVersion

|                    |                                                                                                                                        |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | the movieFileVersion                                                                                                                   |
| <b>Description</b> | Movie property; indicates the version of Director in which the movie was last saved. The result is a string.                           |
| <b>Example</b>     | This statement displays the version of Director that last saved the current movie:<br><pre>put the movieFileVersion<br/>-- "800"</pre> |

## movielImageCompression

|                    |                                                                                                                                                                                                                                                                                                                   |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | the movielImageCompression                                                                                                                                                                                                                                                                                        |
| <b>Description</b> | Movie property; indicates the type of compression that Director applies to internal (non-linked) bitmap members when saving a movie in Shockwave format. This property can be set only during authoring and has no affect until the movie is saved in Shockwave format. Its value can be either of these symbols: |
| <hr/>              |                                                                                                                                                                                                                                                                                                                   |
| <b>Value</b>       | <b>Meaning</b>                                                                                                                                                                                                                                                                                                    |
| #standard          | Use Director's standard internal compression format                                                                                                                                                                                                                                                               |
| #jpeg              | Use JPEG compression (see imageCompression)                                                                                                                                                                                                                                                                       |

You normally set this property in Director's Publish Settings dialog box.

You can choose to override this setting for specific bitmap cast members by setting their imageCompression and imageQuality cast member properties.

**See also** imageCompression, imageQuality, movielImageQuality

## movielImageQuality

|                    |                                                                                                                                                                                                                                                                                         |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | the movielImageQuality                                                                                                                                                                                                                                                                  |
| <b>Description</b> | Movie property; indicates the level of compression to use when the movielImageCompression property is set to #jpeg. The range of acceptable values is 0–100. Zero yields the lowest image quality and highest compression; 100 yields the highest image quality and lowest compression. |
|                    | You can only set this property during authoring and it has no affect until the movie is saved in Shockwave format.                                                                                                                                                                      |
|                    | Individual members may override this movie property by using the member property imageCompression.                                                                                                                                                                                      |
| <b>See also</b>    | imageCompression, imageQuality, movielImageCompression                                                                                                                                                                                                                                  |

## movieName

|                    |                                                                                                                                                                                     |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | the movieName                                                                                                                                                                       |
| <b>Description</b> | Movie property; indicates the simple name of the current movie.<br>In the Director authoring environment, a new movie that has not been saved has an empty string as this property. |
| <b>Example</b>     | This statement displays the name of the current movie in the Message window:<br><pre>put the movieName</pre>                                                                        |
| <b>See also</b>    | moviePath                                                                                                                                                                           |

## moviePath

|                    |                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | the moviePath                                                                                                                                                                                                                                                                                                                                                             |
| <b>Description</b> | Movie property; indicates the pathname of the folder in which the current movie is located.<br>For pathnames that work on both Windows and Macintosh computers, use the @ pathname operator.<br>To see an example of moviePath used in a completed movie, see the Read and Write Text movie in the Learning\Lingo Examples folder inside the Director application folder. |
| <b>Example</b>     | This statement displays the pathname for the folder containing the current movie:<br><pre>put the moviePath</pre>                                                                                                                                                                                                                                                         |
| <b>Example</b>     | This statement plays the sound file Crash.aif stored in the Sounds subfolder of the current movie's folder. Note the path delimiter used, indicating a Windows environment:<br><pre>sound playFile 1, the moviePath&amp;"Sounds/crash.aif"</pre>                                                                                                                          |
| <b>See also</b>    | @ (pathname), movieName                                                                                                                                                                                                                                                                                                                                                   |

## movieRate

**Syntax**

`sprite(whichSprite).movieRate`

the movieRate of sprite *whichSprite*

**Description**

Digital video sprite property; controls the rate at which a digital video in a specific channel plays. The movie rate is a value specifying the playback of the digital video. A value of 1 specifies normal forward play, -1 specifies reverse, and 0 specifies stop. Higher and lower values are possible. For example, a value of 0.5 causes the digital video to play slower than normal. However, frames may be dropped when the movieRate sprite property exceeds 1. The severity of frame dropping depends on factors such as the performance of the computer the movie is playing on and whether the digital video sprite is stretched.

This property can be tested and set.

To see an example of movieRate used in a completed movie, see the QT and Flash movie in the Learning\Lingo Examples folder inside the Director application folder.

**Example**

This statement sets the rate for a digital video in sprite channel 9 to normal playback speed:

```
sprite(9).movieRate = 1
```

This statement causes the digital video in sprite channel 9 to play in reverse:

```
sprite(9).movieRate = -1
```

**See also**

`duration`, `movieTime`

## movieTime

**Syntax**

`sprite(whichSprite).movieTime`

the movieTime of sprite *whichSprite*

**Description**

Digital video sprite property; determines the current time of a digital video movie playing in the channel specified by *whichSprite*. The movieTime value is measured in ticks.

This property can be tested and set.

To see an example of movieTime used in a completed movie, see the QT and Flash movie in the Learning\Lingo Examples folder inside the Director application folder.

**Example**

This statement displays the current time of the QuickTime movie in channel 9 in the Message window:

```
put sprite(9).movieTime
```

|                 |                                                                                                               |
|-----------------|---------------------------------------------------------------------------------------------------------------|
| <b>Example</b>  | This statement sets the current time of the QuickTime movie in channel 9 to the value in the variable Poster: |
|                 | sprite(9).movieTime = Poster                                                                                  |
| <b>See also</b> | duration                                                                                                      |

## movieXtraList

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | the movieXtraList                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Description</b> | Movie property; displays a linear property list of all Xtras in the Movies/Xtras dialog box that have been added to the movie.                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|                    | <ul style="list-style-type: none"> <li>• #name—Specifies the file name of the Xtra on the current platform. It is possible to have a list without a #name entry, such as when the Xtra exists only on one platform.</li> <li>• #packagefiles—Set only when the Xtra is marked for downloading. The value of this property is another list containing a property list for each file in the download package for the current platform. The properties in this subproperty list are #name and #version, which contain the same information as found in the XtraList.</li> </ul> |
|                    | Two possible properties can appear in movieXtraList:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Example</b>     | This statement displays output from movieXtraList in the Message window:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|                    | <pre>put the movieXtraList -- [[#name: "Mix Services"], [#name: "Sound Import Export"], [#name: "SWA Streaming \ PPC Xtra"], [#name: "TextXtra PPC"], [#name: "Font Xtra PPC"], [#name: "Flash Asset \ Options PPC"], [#name: "Font Asset PPC"], [#name: "Flash Asset PPC", \ #packagefiles: [[#fileName: "Flash Asset PPC", #version: "1.0.3"]]]]</pre>                                                                                                                                                                                                                     |
| <b>See also</b>    | xtraList                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

## multiSound

|                    |                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | the multiSound                                                                                                                                                              |
| <b>Description</b> | System property; specifies whether the system supports more than one sound channel and requires a Windows computer to have a multichannel sound card (TRUE) or not (FALSE). |
| <b>Example</b>     | This statement plays the sound file Music in sound channel 2 if the computer supports more than one sound channel:                                                          |
|                    | if the multiSound then sound playFile 2, "Music.wav"                                                                                                                        |

## name (cast property)

|                    |                                                                                                                                                                                                                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | castLib ( <i>whichCast</i> ).name<br>the name of castLib <i>whichCast</i>                                                                                                                                                                                                                                                  |
| <b>Description</b> | Cast member property; returns the name of the specified cast.<br>This property can be tested and set.                                                                                                                                                                                                                      |
| <b>Example</b>     | This code iterates through all the castLibs in a movie and displays their names in the Message window:<br><pre>totalCastLibs = the number of castLibs repeat with currentCastLib = 1 to totalCastLibs     put "Castlib"&amp;&amp;currentCastLib&amp;&amp;"is named"&amp;&amp;castLib(currentCastLib).name end repeat</pre> |
| <b>See also</b>    | && (concatenator)                                                                                                                                                                                                                                                                                                          |

## name (cast member property)

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | member( <i>whichCastMember</i> ).name<br>the name of member <i>whichCastMember</i>                                                                                                                                                                                                                                                                                                                                   |
| <b>Description</b> | Cast member property; determines the name of the specified cast member. The argument <i>whichCastMember</i> is a string when used as the cast member name or an integer when used as the cast member number.<br>The name is a descriptive string assigned by the developer. Setting this property is equivalent to entering a name in the Cast Member Properties dialog box.<br>This property can be tested and set. |
| <b>Example</b>     | This statement changes the name of the cast member named On to Off:<br><pre>member("On").name = "Off"</pre>                                                                                                                                                                                                                                                                                                          |
| <b>Example</b>     | This statement sets the name of cast member 15 to Background Sound:<br><pre>member(15).name = "Background Sound"</pre>                                                                                                                                                                                                                                                                                               |
| <b>Example</b>     | 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:<br><pre>itsName = member(i + 1).name</pre>                                                                                                                                                                                                                             |
| <b>See also</b>    | number (cast member property)                                                                                                                                                                                                                                                                                                                                                                                        |

## name (menu property)

|                    |                                                                                                                                                                                                                                     |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | the name of menu( <i>whichMenu</i> )<br>the name of menu <i>whichMenu</i>                                                                                                                                                           |
| <b>Description</b> | Menu property; returns a string containing the name of the specified menu number.<br><br>This property can be tested but not set. Use the installMenu command to set up a custom menu bar.                                          |
|                    | <b>Note:</b> Menus are not available in Shockwave.                                                                                                                                                                                  |
| <b>Example</b>     | This statement assigns the name of menu number 1 to the variable firstMenu:<br><br>firstMenu = menu(1).name                                                                                                                         |
| <b>Example</b>     | The following handler returns a list of menu names, one per line:<br><br>on menuList<br>theList = []<br>repeat with i = 1 to the number of menus<br>theList[i] = the name of menu i<br>end repeat<br>return theList<br>end menuList |
| <b>See also</b>    | number (menus), name (menu item property)                                                                                                                                                                                           |

## name (menu item property)

|                    |                                                                                                                                                                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | the name of menulItem( <i>whichItem</i> ) of menu( <i>whichMenu</i> )<br>the name of menulItem <i>whichItem</i> of menu <i>whichMenu</i>                                                                                                                                                                                        |
| <b>Description</b> | Menu property; determines the text that appears in the menu item specified by <i>whichItem</i> in the menu specified by <i>whichMenu</i> . The <i>whichItem</i> argument is either a menu item name or a menu item number; <i>whichMenu</i> is either a menu name or a menu number.<br><br>This property can be tested and set. |
|                    | <b>Note:</b> Menus are not available in Shockwave.                                                                                                                                                                                                                                                                              |
| <b>Example</b>     | This statement sets the variable itemName to the name of the eighth item in the Edit menu:<br><br>set itemName = the name of menulItem(8) of menu("Edit")                                                                                                                                                                       |
| <b>Example</b>     | This statement causes a specific file name to follow the word <i>Open</i> in the File menu:<br><br>the name of menulItem("Open") of menu("fileMenu") = "Open" && fileName                                                                                                                                                       |
| <b>See also</b>    | name (menu property), number (menu items)                                                                                                                                                                                                                                                                                       |

## name (window property)

|                    |                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>window (whichWindow).name</code><br>the name of window <i>whichWindow</i>                                                                                                                                              |
| <b>Description</b> | Window property; determines the name of the specified window in <code>windowList</code> . (The title window property determines the title that appears in a window's title bar.)<br><br>This property can be tested and set. |
| <b>Example</b>     | This statement changes the name of the window Yesterday to Today:<br><br><code>window("Yesterday").name = "Today"</code>                                                                                                     |

## name (system property)

|                    |                                                                                                                                                                                                                                |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>xtra (whichXtra).name</code><br>the name of xtra <i>whichXtra</i>                                                                                                                                                        |
| <b>Description</b> | System property; indicates the name of the specified Lingo Xtra. Xtras that provide support services or other functions not available to Lingo will not support this property.<br><br>This property can be tested but not set. |
| <b>Example</b>     | This statement displays the name of the first Xtra in the Message window:<br><br><code>put xtra(1).name</code>                                                                                                                 |

## name (timeout property)

|                    |                                                                                                                                                                                                                                                                   |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>timeoutObject.name</code>                                                                                                                                                                                                                                   |
| <b>Description</b> | This timeout property is the name of the timeout object as defined when the object is created. The <code>new()</code> command is used to create timeout objects.                                                                                                  |
| <b>Example</b>     | This timeout handler opens an alert with the name of the timeout that sent the event:<br><br><code>on handleTimeout timeoutObject<br/>    alert "Timeout:" &amp;&amp; timeoutObject.name<br/>end</code>                                                           |
| <b>See also</b>    | <code>forget</code> <code>window</code> , <code>new()</code> , <code>period</code> , <code>persistent</code> , <code>target</code> , <code>time</code> (timeout object property), <code>timeout()</code> , <code>timeoutHandler</code> , <code>timeoutList</code> |

## NAN

|                    |                                                                                                         |
|--------------------|---------------------------------------------------------------------------------------------------------|
| <b>Description</b> | Lingo return value; indicates that a specified Lingo expression is not a number.                        |
|                    | This statement attempts to display the square root of -1, which is not a number, in the Message window: |
|                    | <pre>put (-1).sqrt<br/>-- NAN</pre>                                                                     |

**See also** INF

## netAbort

|                    |                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>netAbort(<i>URL</i>)</code><br><code>netAbort(<i>netID</i>)</code>                                                                                                                                                                                     |
| <b>Description</b> | Command; cancels a network operation without waiting for a result.                                                                                                                                                                                           |
|                    | Using a network ID is the most efficient way to stop a network operation. The ID is returned when you use a network function such as <code>getNetText()</code> or <code>postNetText()</code> .                                                               |
|                    | In some cases, when a network ID is not available, you can use a URL to stop the transmission of data for that URL. The URL must be identical to that used to begin the network operation. If the data transmission is complete, this command has no effect. |
| <b>Example</b>     | This statement passes a network ID to <code>netAbort</code> to cancel a particular network operation:<br><br><code>on mouseUp<br/>    netAbort(myNetID)<br/>end</code>                                                                                       |
| <b>See also</b>    | <code>getNetText()</code> , <code>postNetText</code>                                                                                                                                                                                                         |

## netDone()

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>netDone()</code><br><code>netDone(<i>netID</i>)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Description</b> | Function; indicates whether a background loading operation (such as <code>getNetText</code> , <code>preloadNetThing</code> , <code>gotoNetMovie</code> , <code>gotoNetPage</code> , or <code>putNetText</code> ) is finished or was terminated by a browser error (TRUE, default) or is still in progress (FALSE). <ul style="list-style-type: none"><li>• Use <code>netDone()</code> to test the last network operation.</li><li>• Use <code>netDone(<i>netID</i>)</code> to test the network operation identified by <i>netID</i>.</li></ul> The <code>netDone</code> function returns 0 when a background loading operation is in progress. |
| <b>Example</b>     | This handler uses the <code>netDone</code> function to test whether the last network operation has finished. If the operation is finished, text returned by <code>netTextResult</code> is displayed in the field cast member <code>Display Text</code> . <pre>on exitFrame     if netDone() = 1 then         member("Display Text").text = netTextResult()     end if end</pre>                                                                                                                                                                                                                                                                |
| <b>Example</b>     | This handler uses a specific network ID as an argument for <code>netDone</code> to check the status of a specific network operation: <pre>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</pre>                                                                                                                                                                                                                                                                                                              |
| <b>See also</b>    | <code>getNetText()</code> , <code>netTextResult()</code> , <code>gotoNetMovie</code> , <code>preloadNetThing()</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

## netError()

|                    |                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>netError()</code><br><code>netError(<i>netID</i>)</code>                                                                                                                                                                                                                                                                                                                |
| <b>Description</b> | Function; determines whether an error has occurred in a network operation and, if so, returns an error number corresponding to an error message. If the operation was successful, this function returns a code indicating that everything is okay. If no background loading operation has started, or if the operation is in progress, this function returns an empty string. |

- Use `netError()` to test the last network operation.
- Use `netError(netID)` to test the network operation specified by `netID`.

Several possible error codes may be returned:

---

|      |                                                                                                                                  |
|------|----------------------------------------------------------------------------------------------------------------------------------|
| 0    | Everything is okay.                                                                                                              |
| 4    | Bad MOA class. The required network or nonnetwork Xtras are improperly installed or not installed at all.                        |
| 5    | Bad MOA Interface. See 4.                                                                                                        |
| 6    | Bad URL or Bad MOA class. The required network or nonnetwork Xtras are improperly installed or not installed at all.             |
| 20   | Internal error. Returned by <code>netError()</code> in the Netscape browser if the browser detected a network or internal error. |
| 4146 | Connection could not be established with the remote host.                                                                        |
| 4149 | Data supplied by the server was in an unexpected format.                                                                         |
| 4150 | Unexpected early closing of connection.                                                                                          |
| 4154 | Operation could not be completed due to timeout.                                                                                 |
| 4155 | Not enough memory available to complete the transaction.                                                                         |
| 4156 | Protocol reply to request indicates an error in the reply.                                                                       |
| 4157 | Transaction failed to be authenticated.                                                                                          |
| 4159 | Invalid URL.                                                                                                                     |
| 4164 | Could not create a socket.                                                                                                       |
| 4165 | Requested object could not be found (URL may be incorrect).                                                                      |
| 4166 | Generic proxy failure.                                                                                                           |
| 4167 | Transfer was intentionally interrupted by client.                                                                                |
| 4242 | Download stopped by <code>netAbort(url)</code> .                                                                                 |
| 4836 | Download stopped for an unknown reason, possibly a network error, or the download was abandoned.                                 |

---

When a movie plays back as an applet, this function always returns a string. The string either has a length of 0 or consists of text that describes an error. The string's content comes from Java and can vary on different operating systems or browsers. The text may not be translated into the local language.

**Example** This statement passes a network ID to netError to check the error status of a particular network operation:

```
on exitFrame
 global mynetID
 if netError(mynetID)<>"OK" then beep
end
```

## netLastModDate()

**Syntax** netLastModDate()

**Description** Function; returns the date last modified from the HTTP header for the specified item. The string is in Universal Time (GMT) format: *Ddd, nn Mmm yyyy hh:mm:ss GMT* (for example, Thu, 30 Jan 1997 12:00:00 AM GMT). There are variations where days or months are spelled completely. The string is always in English.

The netLastModDate function can be called only after netDone and netError report that the operation is complete and successful. After the next operation starts, the Director movie or projector discards the results of the previous operation to conserve memory.

The actual date string is pulled directly from the HTTP header in the form provided by the server. However, this string is not always provided, and in that case netLastModDate returns EMPTY.

When a movie plays back as an applet, this function's date format may differ from the date format that Shockwave uses; the date is in the format that the Java function DateAsString returns. The format may also vary on systems that use different languages.

**Example** These statements check the date of a file downloaded from the Internet:

```
if netDone() then
 theDate = netLastModDate()
 if theDate.char[6..11] <> "Jan 30" then
 alert "The file is outdated."
 end if
end if
```

**See also** netDone(), netError()

## netMIME()

**Syntax** netMIME()

**Description** Function; provides the MIME type of the Internet file that the last network operation returned (the most recently downloaded HTTP or FTP item).

The netMIME function can be called only after netDone and netError report that the operation is complete and successful. After the next operation starts, the Director movie or projector discards the results of the previous operation to conserve memory.

**Example** This handler checks the MIME type of an item downloaded from the Internet and responds accordingly:

```
on checkNetOperation theURL
 if netDone (theURL) then
 set myMimeType = netMIME()
 case myMimeType of
 "image/jpeg": go frame "jpeg info"
 "image/gif": go frame "gif info"
 "application/x-director": goToNetMovie theURL
 "text/html": goToNetPage theURL
 otherwise: alert "Please choose a different item."
 end case
 else
 go the frame
 end if
end
```

**See also** netDone(), netError(), getNetText(), postNetText, preloadNetThing()

## netPresent

**Syntax** netPresent()

the netPresent

**Description** System property; determines whether the Xtras needed to access the Internet are available but does not report whether an Internet connection is currently active.

If the Net Support Xtras are not available, netPresent will function properly, but netPresent() will cause a script error

**Example** This statement sends an alert if the Xtras are not available:

```
if not (the netPresent) then
 alert "Sorry, the Network Support Xtras could not be found."
end if
```

## netStatus

**Syntax**    netStatus *msgString*

**Description**    Command; displays the specified string in the status area of the browser window.  
The netStatus command doesn't work in projectors.

**Example**    This statement would place the string "This is a test" in the status area of the browser the movie is running in:

```
on exitFrame
 netStatus "This is a test"
end
```

## netTextResult()

**Syntax**    netTextResult(*netID*)

netTextResult()

**Description**    Function; returns the text obtained by the specified network operation. If no net ID is specified, netTextResult returns the result of the last network operation.

If the specified network operation was getNetText(), the text is the text of the file on the network.

If the specified network operation was postNetText, the result is the server's response.

After the next operation starts, Director discards the results of the previous operation to conserve memory.

When a movie plays back as an applet, this function returns valid results for the last 10 requests. When a movie plays back as a Shockwave movie, this function returns valid results for only the most recent getNetText() operation.

**Example**    This handler uses the "netDone and netError" functions to test whether the last network operation finished successfully. If the operation is finished, text returned by netTextResult is displayed in the field cast member Display Text.

```
global gNetID

on exitFrame
 if (netDone(gNetID) = TRUE) and (netError(gNetID) = "OK") then
 member("Display Text").text = netTextResult()
 end if
end
```

**See also**    netDone(), netError(), postNetText

## netThrottleTicks

**Syntax** the netThrottleTicks

**Description** System property; in the Macintosh authoring environment, allows you to control the frequency of servicing to a network operation.

The default value is 15. The higher the value is set, the smoother the movie playback and animation is, but less time is spent servicing any network activity. A low setting allows more time to be spent on network operations, but will adversely affect playback and animation performance.

This property only affects the authoring environment and projectors on the Macintosh. It is ignored on Windows or Shockwave on the Mac.

## new()

**Syntax** new(type)

new(type, castLib whichCast)

new(type, member whichCastMember of castLib whichCast)

variableName = new(parentScript arg1, arg2, ...)

new(script parentScriptName, value1, value2, ...)

timout("name").new(timeoutPeriod, #timeoutHandler, {, targetObject})

new(xtra "xtraName")

**Description** Function; creates a new cast member, child object, timeout object, or Xtra instance and allows you to assign of individual property values to child objects.

The Director player for Java supports this function only for the creation of child objects. When a movie plays back as an applet, you can't use the new function to create cast members.

For cast members, the type parameter sets the cast member's type. Possible predefined values correspond to the existing cast member types: #bitmap, #field, and so on. The new function can also create Xtra cast member types, which can be identified by any name that the author chooses.

It's also possible to create a new color cursor cast member using the Custom Cursor Xtra. Use new(#cursor) and set the properties of the resulting cast member to make them available for use.

The optional whichCastMember and whichCast parameters specify the cast member slot and Cast window where the new cast member is stored. When no cast member slot is specified, the first empty slot is used. The new function returns the cast member slot.

When the argument for the new function is a parent script, the new function creates a child object. The parent script should include an on new handler that sets the child object's initial state or property values and returns the me reference to the child object.

The child object has all the handlers of the parent script. The child object also has the same property variable names that are declared in the parent script, but each child object has its own values for these properties.

Because a child object is a value, it can be assigned to variables, placed in lists, and passed as a parameter.

As with other variables, you can use the put command to display information about a child object in the Message window.

When new() is used to create a timeout object, the timeoutPeriod sets the number of milliseconds between timeout events sent by the timeout object. The #timeoutHandler is a symbol that identifies the handler that will be called when each timeout event occurs. The targetObject identifies the name of the child object that contains the #timeoutHandler. If no targetObject is given, the #timeoutHandler is assumed to be in a movie script.

When a timeout object is created, it enables its targetObject to receive the system events prepareMovie, startMovie, stopMovie, prepareFrame, and exitFrame. To take advantage of this, the targetObject must contain handlers for these events.

The events do not need to be passed in order for the rest of the movie to have access to them.

To see an example of new() used in a completed movie, see the Parent Scripts, and Read and Write Text movies in the Learning\Lingo Examples folder inside the Director application folder.

**Example** To create a new bitmap cast member in the first available slot, you use this syntax:

```
set newMember = new(#bitmap)
```

After the line has been executed, newMember will contain the member reference to the cast member just created:

```
put newMember
-- (member 1 of castLib 1)
```

**Example** This startMovie script creates a new Flash cast member using the new command, sets the newly created cast member's linked property so that the cast member's assets are stored in an external file, and then sets the cast member's pathname property to the location of a Flash movie on the World Wide Web:

```
on startMovie
 flashCastMember = new(#flash)
 member(flashCastMember).pathName = "http://www.someURL.com/myFlash.swf"
end
```

**Example** When the movie starts, this handler creates a new animated color cursor cast member and stores its cast member number in a variable called customCursor. This variable is used to set the castMemberList property of the newly created cursor and to switch to the new cursor.

```
on startmovie
 customCursor = new(#cursor)
 member(customCursor).castMemberList = [member 1, member 2, member 3]
 cursor (member(customCursor))
end
```

**Example** These statements from a parent script include the on new handler to create a child object. The parent script is a script cast member named Bird, which contains these handlers.

```
on new me, nameForBird
 return me
end

on fly me
 put "I am flying"
end
```

**Example** The first statement in the example creates a child object the above script in the preceding example, and places it in a variable named myBird. The second statement makes the bird fly by calling the fly handler in the Bird parent script:

```
myBird = script("Bird").new()
myBird.fly()
```

**Example** This statement uses a new Bird parent script, which contains the property variable speed:

```
property speed

on new me, initSpeed
 speed = initSpeed
 return me
end

on fly me
 put "I am flying at " & speed & "mph"
end
```

**Example** These statements create two child objects called myBird1 and myBird2. They are given different starting speeds: 15 and 25, respectively. When the fly handler is called for each child object, the speed of the object is displayed in the Message window.

```
myBird1 = script("Bird").new(15)
myBird2 = script("Bird").new(25)
myBird1.fly()
myBird2.fly()
```

This message appears in the Message window:

```
-- "I am flying at 15 mph"
-- "I am flying at 25 mph"
```

**Example** This statement creates a new timeout object called intervalTimer that will send a timeout event to the on minuteBeep handler in the child object playerOne every 60 seconds:

```
timeout("intervalTimer").new(60000, #minuteBeep, playerOne)
```

**See also** on stepFrame, actorList, ancestor, me, type (cast member property), timeout()

## newCurve()

**Syntax**  
`vectorMember.newCurve(positionInVertexList)`  
`newCurve(vectorMember, positionInVertexList)`

**Description** Function; adds a #newCurve symbol to the vertexList of *vectorCastMember*, which adds a new shape to the vector shape. The #newCurve symbol is added at *positionInVertexList*. You can break apart an existing shape by calling newCurve() with a position in the middle of a series of vertices.

**Example** This Lingo adds a new curve to cast member 2 at the third position in the cast member's vertexList. The second line of the example replaces the contents of curve 2 with the contents of curve 3.

```
member(2).newCurve(3)
member(2).curve[2]=member(2).curve[3]
curve, vertexList
```

## next

**Syntax** `next`

**Description** Keyword; refers to the next marker in the movie and is equivalent to the phrase the marker (+ 1).

**Example** This statement sends the playback head to the next marker in the movie:  
`go next`

**Example** This handler moves the movie to the next marker in the Score when the right arrow key is pressed and to the previous marker when the left arrow key is pressed:

```
on keyUp
 if the keyCode = 124 then go next
 if the keyCode = 123 then go previous
end keyUp
```

**See also** `loop` (keyword), `go previous`

## next repeat

**Syntax** next repeat

**Description** Keyword; sends Lingo to the next step in a repeat loop in a script. This function differs from that of the exit repeat keyword.

**Example** This repeat loop displays only odd numbers in the Message window:

```
repeat with i = 1 to 10
 if (i mod 2) = 0 then next repeat
 put i
end repeat
```

## node

**Syntax** sprite(*whichQTVRSprite*).node

the node of sprite *whichQTVRSprite*

**Description** QuickTime VR sprite property; the current node ID displayed by the sprite.

This property can be tested and set.

## nodeEnterCallback

**Syntax** sprite(*whichQTVRSprite*).nodeEnterCallback

the nodeEnterCallback of sprite *whichQTVRSprite*

**Description** QuickTime VR sprite property; contains the name of the handler that runs after the QuickTime VR movie switches to a new active node on the Stage. The message has two arguments: the me parameter and the ID of the node that is being displayed.

The QuickTime VR sprite receives the message first.

To clear the callback, set this property to 0.

To avoid a performance penalty, set a callback property only when necessary.

This property can be tested and set.

## nodeExitCallback

**Syntax**    `sprite(whichQTVRSprite).nodeExitCallback`

the nodeExitCallback of sprite *whichQTVRSprite*

**Description**    QuickTime VR sprite property; contains the name of the handler that runs when the QuickTime VR movie is about to switch to a new active node on the Stage. The message has three arguments: the me parameter, the ID of the node that the movie is about to leave, and the ID of the node that the movie is about to switch to.

The value that the handler returns determines whether the movie goes on to the next node. If the handler returns #continue, the QuickTime VR sprite continues with a normal node transition. If the handler returns #cancel, the transition doesn't occur and the movie stays in the original node.

Set this property to 0 to clear the callback.

The QuickTime VR sprite receives the message first.

To avoid a performance penalty, set a callback property only when necessary.

This property can be tested and set.

## nodeType

**Syntax**    `sprite(whichQTVRSprite).nodeType`

nodeType of sprite *whichQTVRSprite*

**Description**    QuickTime VR sprite property; gives the type of node that is currently on the Stage for the specified sprite. Possible values are #object, #panorama, or #unknown. (#unknown is the value for a sprite that isn't a QuickTime VR sprite.)

This property can be tested but not set.

## not

**Syntax**    `not logicalExpression`

**Description**    Operator; performs a logical negation on a logical expression. This is the equivalent of making a TRUE value FALSE, and making a FALSE value TRUE. It is useful when testing to see if a certain known condition is not the case.

This logical operator has a precedence level of 5.

**Example**    This statement determines whether 1 is not less than 2:

```
put not (1 < 2)
```

Because 1 is less than 2, the result is 0, which indicates that the expression is FALSE.

|                 |                                                                                                                                                                           |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Example</b>  | This statement determines whether 1 is not greater than 2:<br>put not (1 > 2)                                                                                             |
|                 | Because 1 is not greater than 2, the result is 1, which indicates that the expression is TRUE.                                                                            |
| <b>Example</b>  | This handler sets the checkMark menu item property for Bold in the Style menu to the opposite of its current setting:                                                     |
|                 | <pre>on resetMenuItem     the checkMark of menuItem("Bold") of menu("Style") = \         not (the checkMark of menuItem("Bold") of menu("Style")) end resetMenuItem</pre> |
| <b>See also</b> | and, or                                                                                                                                                                   |

## nothing

|                    |                                                                                                                                                                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | nothing                                                                                                                                                                                                                                                                                                             |
| <b>Description</b> | Command; does nothing. This command is useful for making the logic of an if...then statement more obvious. A nested if...then...else statement that contains no explicit command for the else clause may require else nothing, so that Lingo does not interpret the else clause as part of the preceding if clause. |
| <b>Example</b>     | The nested if...then...else statement in this handler uses the nothing command to satisfy the statement's else clause:                                                                                                                                                                                              |
|                    | <pre>on mouseDown     if the clickOn = 1 then         if sprite(1).moveableSprite = TRUE then             member("Notice").text = "Drag the ball"         else nothing         else member("Notice").text = "Click again"         end if     end if</pre>                                                           |
| <b>Example</b>     | This handler instructs the movie to do nothing so long as the mouse button is being pressed:                                                                                                                                                                                                                        |
|                    | <pre>on mouseDown     repeat while the stillDown         nothing     end repeat end mouseDown</pre>                                                                                                                                                                                                                 |
| <b>See also</b>    | if                                                                                                                                                                                                                                                                                                                  |

## nudge

**Syntax**  
sprite(*whichQTVRSprite*).nudge(*#direction*)  
nudge(*sprite whichQTVRSprite*, *#direction*)

**Description** QuickTime VR command; nudges the view perspective of the specified QuickTime VR sprite in the direction specified by *#direction*. Possible values for *#direction* are *#down*, *#downLeft*, *#downRight*, *#left*, *#right*, *#up*, *#upLeft*, and *#upRight*. Nudging to the right causes the image of the sprite to move to the left.

The nudge command has no return value.

**Example** This handler causes the perspective of the QTVR sprite to move to the left as long as the mouse is held down on the sprite.

```
on mouseDown me
 repeat while the stillDown
 sprite(1).nudge(#left)
 end repeat
end
```

## number (cast property)

**Syntax** the number of castLib *whichCast*

**Description** Cast property; indicates the number of the specified cast. For example, 2 is the castLib number for Cast 2.

This property can be tested but not set.

**Example** This repeat loop uses the Message window to display the number of cast members that are in each of the movie's casts:

```
repeat with n = 1 to the number of castLibs
 put castLib(n).name && "contains" && the number of \
 members of castLib(n) && "cast members."
end repeat
```

## number (cast member property)

### Syntax

member(*whichCastMember*).number

the number of member *whichCastMember*

### Description

Cast member property; indicates the cast number of the cast member specified by *whichCastMember*: either a name, if *whichCastMember* is a string, or a number, if *whichCastMember* is an integer.

The property is a unique identifier for the cast member that is a single integer describing its location in and position in the castLib.

This property can be tested but not set.

**Note:** When using the first syntax of member(*whichCastMember*).number, an error is generated if the cast member does not exist. When unsure of the existence of the member, use the alternate syntax to avoid the error.

### Example

This statement assigns the cast number of the cast member Power Switch to the variable *whichCastMember*:

```
whichCastMember = member("Power Switch").number
```

### Example

This statement assigns the cast member Red Balloon to sprite 1:

```
sprite(1).member = member("Red Balloon").number
```

### Example

This verifies that a cast member actually exists before trying to switch the cast member in the sprite:

```
property spriteNum
```

```
on mouseUp me
 if (member("Mike's face").number > 0) then
 sprite(spriteNum).member = "Mike's face"
 end if
end
```

### See also

member (sprite property), memberNum, number of members

## number (characters)

**Syntax**

the number of chars in *chunkExpression*

**Description**

Chunk expression; returns a count of the characters in a chunk expression.

Chunk expressions are any character (including spaces and control characters such as tabs and carriage returns), word, item, or line in any container of characters. Containers include field cast members and variables that hold strings, and specified characters, words, items, lines, and ranges in containers.

**Note:** The count() function provides a more efficient alternative for determining the number of characters in a chunk expression.

**Example**

This statement displays the number of characters in the string “Macromedia, the Multimedia Company” in the Message window:

```
put the number of chars in "Macromedia, the Multimedia Company"
```

The result is 34.

**Example**

This statement sets the variable charCounter to the number of characters in the word i located in the string Names:

```
charCounter = the number of chars in member("Names").word[i]
```

You can accomplish the same thing with text cast members using the syntax:

```
charCounter = member("Names").word[i].char.count
```

**See also**

length(), char...of, count(), number (items), number (lines), number (words)

## number (items)

**Syntax**

the number of items in *chunkExpression*

**Description**

Chunk expression; returns a count of the items in a chunk expression. An item chunk is any sequence of characters delimited by commas.

Chunk expressions are any character, word, item, or line in any container of characters. Containers include fields (field cast members) and variables that hold strings, and specified characters, words, items, lines, and ranges in containers.

**Note:** The count() function provides a more efficient alternative for determining the number of items in a chunk expression.

**Example**

This statement displays the number of items in the string “Macromedia, the Multimedia Company” in the Message window:

```
put the number of items in "Macromedia, the Multimedia Company"
```

The result is 2.

**Example** This statement sets the variable itemCounter to the number of items in the field Names:

```
itemCounter = the number of items in member("Names").text
```

You can accomplish the same thing with text cast members using the syntax:

```
itemCounter = member("Names").item.count
```

**See also** item...of, count(), number (characters), number (lines), number (words)

## number (lines)

**Syntax** the number of lines in *chunkExpression*

**Description** Chunk expression; returns a count of the lines in a chunk expression. (Lines refers to lines delimited by carriage returns, not lines formed by line wrapping.)

Chunk expressions are any character, word, item, or line in any container of characters. Containers include field cast members and variables that hold strings, and specified characters, words, items, lines, and ranges in containers.

**Note:** The count() function provides a more efficient alternative for determining the number of lines in a chunk expression.

**Example** This statement displays the number of lines in the string "Macromedia, the Multimedia Company" in the Message window:

```
put the number of lines in "Macromedia, the Multimedia Company"
```

The result is 1.

**Example** This statement sets the variable lineCounter to the number of lines in the field Names:

```
lineCounter = the number of lines in member("Names").text
```

You can accomplish the same thing with text cast members with the syntax:

```
lineCounter = member("Names").line.count
```

**See also** item...of, count(), number (characters), number (items), number (words)

## number (menus)

**Syntax** the number of menus

**Description** Menu property; indicates the number of menus installed in the current movie. This menu property can be tested but not set. Use the installMenu command to set up a custom menu bar.

**Note:** Menus are not available in Shockwave

**Example** This statement determines whether any custom menus are installed in the movie and, if no menus are already installed, installs the menu Menubar:

```
if the number of menus = 0 then installMenu "Menubar"
```

**Example** This statement displays in the Message window the number of menus that are in the current movie:

```
put the number of menus
```

**See also** installMenu, number (menu items)

## number (menu items)

**Syntax** the number of menultems of menu *whichMenu*

**Description** Menu property; indicates the number of menu items in the custom menu specified by *whichMenu*. The *whichMenu* parameter can be a menu name or menu number.

This menu property can be tested but not set. Use the installMenu command to set up a custom menu bar.

**Note:** Menus are not available in Shockwave

**Example** This statement sets the variable fileItems to the number of menu items in the custom File menu:

```
fileItems = the number of menultems of menu "File"
```

**Example** This statement sets the variable itemCount to the number of menu items in the custom menu whose menu number is equal to the variable i:

```
itemCount = the number of menultems of menu i
```

**See also** installMenu, number (menus)

## number (system property)

|                    |                                                                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | the number of castLibs                                                                                                                                                                                                                                                                                                       |
| <b>Description</b> | System property; returns the number of casts that are in the current movie.<br>This property can be tested but not set.                                                                                                                                                                                                      |
| <b>Example</b>     | This repeat loop uses the Message window to display the number of cast members that are in each of the movie's casts:<br><pre>repeat with n = 1 to the number of castLibs     put castLib(n).name &amp;&amp; "contains" &amp;&amp; the number of \         members of castLib(n) &amp;&amp; "cast members." end repeat</pre> |

## number (words)

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | the number of words in <i>chunkExpression</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Description</b> | Chunk expression; returns the number of words in the chunk expression specified by <i>chunkExpression</i> .<br>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.<br>To accomplish this functionality with text cast members, see count.<br><br><b>Note:</b> The count() function provides a more efficient alternative for determining the number of words in a chunk expression. |
| <b>Example</b>     | This statement displays in the Message window the number of words in the string "Macromedia, the multimedia company":<br><pre>put the number of words in "Macromedia, the multimedia company"</pre> The result is 4.                                                                                                                                                                                                                                                                                                                                                     |
| <b>Example</b>     | This handler reverses the order of words in the string specified by the argument wordList:<br><pre>on reverse wordList     theList = EMPTY     repeat with i = 1 to the number of words in wordList         put word i of wordList &amp; " " before theList     end repeat     delete theList.char[thelist.char.count]     return theList end</pre>                                                                                                                                                                                                                      |
| <b>See also</b>    | count(), number (characters), number (items), number (lines), word...of                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

## number of members

**Syntax** the number of members of castLib *whichCast*

**Description** Cast member property; indicates the number of the last cast member in the specified cast.

This property can be tested but not set.

**Example** This statement displays in the Message window the type of each cast member in the cast Central Casting. The number of members of castLib property is used to determine how many times the loop repeats.

```
repeat with i = 1 to the number of members of castLib("Central Casting")
 put "Cast member" && i && "is a" && member(i, "Central Casting").type
end repeat
```

## number of xtras

**Syntax** the number of xtras

**Description** System property; returns the number of scripting Xtras available to the movie. The Xtras may be either those opened by the openXlib command or those present in the standard Xtras folder.

This property can be tested but not set.

**Example** This statement displays in the Message window the number of scripting Xtras that are available to the movie:

```
put the number of xtras
```

## numChannels

**Syntax** member(*whichCastMember*).numChannels

the numChannels of member *whichCastMember*

**Description** Shockwave Audio (SWA) cast member property; returns the number of channels within the specified SWA streaming cast member. The value can be either 1 for monaural or 2 for stereo.

This property is available only after the SWA streaming cast member begins playing or after the file has been preloaded using the preLoadBuffer command.

This property can be tested but not set.

**Example** This example assigns the number of sound channels of the SWA streaming cast member Duke Ellington to the field cast member Channel Display:

```
myVariable = member("Duke Ellington").numChannels
if myVariable = 1 then
 member("Channel Display").text = "Mono"
else
 member("Channel Display").text = "Stereo"
end if
```

## numToChar()

### Syntax

```
numToChar(integerExpression)
```

**Description** Function; displays a string containing the single character whose ASCII number is the value of *integerExpression*. This function is useful for interpreting data from outside sources that are presented as numbers rather than as characters.

ASCII values up to 127 are standard on all computers. Values of 128 or greater refer to different characters on different computers.

**Example** This statement displays in the Message window the character whose ASCII number is 65:

```
put numToChar(65)
```

The result is the letter *A*.

**Example** This handler removes any nonalphanumeric characters from any arbitrary string and returns only capital letters:

```
on ForceUppercase input
 output = EMPTY
 num = length(input)
 repeat with i = 1 to num
 theASCII = charToNum(input.char[i])
 if theASCII = min(max(96, theASCII), 123) then
 theASCII = theASCII - 32
 if theASCII = min(max(63, theASCII), 91) then
 put numToChar(theASCII) after output
 end if
 end if
 end repeat
 return output
end
```

**See also** [charToNum\(\)](#)

## obeyScoreRotation

### Syntax

member(*flashMember*).obeyScoreRotation

### Description

Flash cast member property; set to TRUE or FALSE to determine if a Flash movie sprite uses the rotation information from the Score, or the older rotation property of Flash assets.

This property is automatically set to FALSE for all movies created in Director prior to version 7 in order to preserve old functionality of using the member rotation property for all sprites containing that Flash member.

New assets created in version 7 or later will have this property automatically set to TRUE.

If set to TRUE, the rotation property of the member is ignored and the Score rotation settings are obeyed instead.

### Example

The following sprite script sets the obeyScoreRotation property of cast member "dalmation" to 1 (TRUE), then rotates the sprite which contains the cast member 180 degrees.

```
on mouseUp me
 member("dalmation").obeyScoreRotation = 1
 sprite(1).rotation = sprite(1).rotation + 180
end
```

This property can be tested and set.

### See also

rotation

## objectP()

### Syntax

objectP(*expression*)

### Description

Function; indicates whether the expression specified by *expression* is an object produced by a parent script, Xtra, or window (TRUE) or not (FALSE).

The *P* in objectP stands for *predicate*.

It is good practice to use objectP to determine which items are already in use when you create objects by parent scripts or Xtra instances.

To see an example of objectP() used in a completed movie, see the Read and Write Text movie in the Learning\Lingo Examples folder inside the Director application folder.

|                 |                                                                                                                                                                                                                                                       |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Example</b>  | This statement checks whether the global variable gDataBase has an object assigned to it and, if not, assigns one. This check is commonly used when you perform initializations at the beginning of a movie or section that you don't want to repeat. |
|                 | <pre>if objectP(gDataBase) then     nothing else     gDataBase = script("Database Controller").new() end if</pre>                                                                                                                                     |
| <b>See also</b> | floatP(), ilk(), integerP(), stringP(), symbolP()                                                                                                                                                                                                     |

## of

The word of is part of many Lingo properties, such as foreColor, number, name, and so on.

## offset() (string function)

**Syntax** offset(*stringExpression1*, *stringExpression2*)

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | Function; returns an integer indicating the position of the first character of <i>stringExpression1</i> in <i>stringExpression2</i> . This function returns 0 if <i>stringExpression1</i> is not found in <i>stringExpression2</i> . Lingo counts spaces as characters in both strings.<br><br>On the Macintosh, the string comparison is not sensitive to case or diacritical marks. For example, Lingo considers <i>a</i> and <i>À</i> to be the same character on the Macintosh. |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                |                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------|
| <b>Example</b> | This statement displays in the Message window the beginning position of the string “media” within the string “Macromedia”: |
|                | <pre>put offset("media", "Macromedia")</pre>                                                                               |

The result is 6.

|                |                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------|
| <b>Example</b> | This statement displays in the Message window the beginning position of the string “Micro” within the string “Macromedia”: |
|                | <pre>put offset("Micro", "Macromedia")</pre>                                                                               |

The result is 0, because “Macromedia” doesn’t contain the string “Micro”.

**Example** This handler finds all instances of the string represented by `stringToFind` within the string represented by `input` and replaces them with the string represented by `stringToInsert`.

```
on SearchAndReplace input, stringToFind, stringToInsert
 output = ""
 findLen = stringToFind.length - 1
 repeat while input contains stringToFind
 currOffset = offset(stringToFind, input)
 output = output & input.char [1..currOffset]
 delete the last char of output
 output = output & stringToInsert
 delete input.char [1.. (currOffset + findLen)]
 end repeat
 set output = output & input
 return output
end
```

**See also** `chars()`, `length()`, `contains`, `starts`

## offset() (rectangle function)

**Syntax** `rectangle.offset(horizontalChange, verticalChange)`  
`offset (rectangle, horizontalChange, verticalChange)`

**Description** Function; yields a rectangle that is offset from the rectangle specified by `rectangle`. The horizontal offset is the value specified by `horizontalChange`; the vertical offset is the value specified by `verticalChange`.

- When `horizontalChange` is greater than 0, the offset is toward the right of the Stage; when `horizontalChange` is less than 0, the offset is toward the left of the Stage.
- When `verticalChange` is greater than 0, the offset is toward the top of the Stage; when `verticalChange` is less than 0, the offset is toward the bottom of the Stage.

The values for `verticalChange` and `horizontalChange` are in pixels.

**Example** This handler moves sprite 1 five pixels to the right and five pixels down.

```
on diagonalMove
 newRect=sprite(1).rect.offset(5, 5)
 sprite(1).rect=newRect
end
```

## on

**Syntax**

```
on handlerName {argument1}, {arg2}, {arg3} ...
 statement(s)
end handlerName
```

- Description** Keyword; indicates the beginning of a handler, a collection of Lingo statements that you can execute using the handler name. A handler can accept arguments as input values and returns a value as a function result.
- Handlers can be defined in behaviors, movie scripts, and cast member scripts. A handler in a cast member script can be called only by other handlers in the same script. A handler in a movie script can be called from anywhere.
- You can use the same handler in more than one movie by putting the handler's script in a shared cast.

## open

- Syntax**
- ```
open {whichDocument with} whichApplication
```
- Description** Command; launches the application specified by the string *whichApplication*. Use *whichDocument* to specify a document that the application opens when it is launched. When either is in a different folder than the current movie, you must specify the full pathname to the file or files.
- The computer must have enough memory to run both Director and other applications at the same time.
- This is a very simple command for opening an application or a document within an application. For more control, look at options available in third-party Xtras.
- Example** This statement checks whether the computer is a Macintosh and then if it is, opens the application SimpleText:
- ```
if the platform contains "Mac" then open "SimpleText"
```
- Example** This statement opens the SimpleText application, which is in the folder Applications on the drive HD, and the document named Storyboards:
- ```
open "Storyboards" with "HD:Applications:SimpleText"
```
- See also** openXlib

openResFile

This is obsolete. Use recordFont.

open window

Syntax

```
window(whichWindow).open()
```

open window *whichWindow*

Description

Window command; opens the window object or movie file specified by *whichWindow* and brings it to the front of the Stage. If no movie is assigned to the window, the Open File dialog box appears.

- If you replace *whichWindow* with a movie's file name, the window uses the file name as the window.
- If you replace *whichWindow* with a window name, the window takes that name. However, you must then assign a movie to the window by using set the fileName of window.

To open a window that uses a movie from a URL, it's a good idea to use the downloadNetThing command to download the movie's file to a local disk first and then use the file on the disk. This minimizes problems with waiting for the movie to download.

For local media, the movie is not loaded into memory until the open movie command is executed. This can create a noticeable delay if you don't use preloadMovie to load at least the first frame of the movie prior to issuing the open window command.

Note: Opening a movie in a window is currently not supported in playback using a browser.

Example

This statement opens the window Control Panel and brings it to the front:

```
window("Control Panel").open()
```

See also

close window, downloadNetThing, preLoadMovie

on openWindow

Syntax

```
on openWindow
```

statement(s)

end

Description

System message and event handler; contains statements that run when Director opens a window and is a good place to put Lingo that you want executed every time the movie's window opens.

Example

This handler plays the sound file Hurray when the window that the movie is playing in opens:

```
on openWindow
    puppetSound 2, "Hurray"
end
```

openXlib

Syntax openXlib *whichFile*

Description Command; opens the Xlibrary file specified by the string expression *whichFile*. If the file is not in the folder containing the current movie, *whichFile* must include the pathname.

It is good practice to close any file you have opened as soon as you are finished using it. The openXlib command has no effect on an open file.

The openXlib command doesn't support URLs as file references.

Xlibrary files contain Xtras. Unlike openResFile, openXlib makes these Xtras known to Director.

In Windows, the .dll extension is optional.

Note: This command is not supported in Shockwave.

Example This statement opens the Xlibrary file Video Disc Xlibrary:

```
openXlib "Video Disc Xlibrary"
```

Example This statement opens the Xlibrary file Xtras, which is in a different folder than the current movie:

```
openXlib "My Drive>New Stuff>Transporter Xtras"
```

See also closeXlib, interface(), showXlib

optionDown

Syntax the optionDown

Description System property; determines whether the user is pressing the Alt key (Windows) or the Option key (Macintosh) (TRUE) or not (FALSE).

In Windows, optionDown doesn't work in projectors if Alt is pressed without another nonmodifier key. Avoid using optionDown if you intend to distribute a movie as a Windows projector and need to detect only the modifier key press; use controlDown or shiftDown instead.

On the Macintosh, pressing the Option key changes the key value, so use keyCode instead.

For a movie playing back with the Director player for Java, this function returns TRUE only if a second key is pressed at the same time as the Alt or Option key. If the Alt or Option key is pressed by itself, optionDown returns FALSE.

The Director player for Java supports key combinations with the Alt or Option key. However, the browser receives the keys before the movie plays and responds to and intercepts any key combinations that are also browser keyboard shortcuts.

Example This handler checks whether the user is pressing the Alt or the Option key and if so, calls the handler named doOptionKey:

```
on keyDown  
    if (the optionDown) then doOptionKey(key)  
end keyDown
```

See also controlDown, commandDown, key(), keyCode(), shiftDown

or

Syntax *logicalExpression1* or *logicalExpression2*

Description Operator; performs a logical OR operation on two or more logical expressions to determine whether any expression is TRUE.

This is a logical operator with a precedence level of 4.

Example This statement indicates in the Message window whether at least one of the expressions $1 < 2$ and $1 > 2$ is TRUE:

```
put (1 < 2) or (1 > 2)
```

Because the first expression is TRUE, the result is 1, which is the numerical equivalent of TRUE.

Example This statement checks whether the content of the field cast member named State is either AK or HI and displays an alert if it is:

```
if member("State").text = "AK" or member("State").text = "HI" then  
    alert "You're off the map!"  
end if"
```

See also and, not

organizationName

Syntax the organizationName

Description Movie property; contains the company name entered during installation of Director.

This property is available in the authoring environment only. It can be used in a movie in a window tool that is personalized to show the user's information.

Example This handler would be located in a movie script of a movie in a window (MIAW). It places the user's name and serial number into a display field when the window is opened:

```
on prepareMovie
    displayString = the userName
    put RETURN & the organizationName after displayString
    put RETURN & the serialNumber after displayString
    member("User Info").text = displayString
end
```

See also [serialNumber](#), [userName](#), [window](#)

originalFont

Syntax `member(whichFontMember).originalFont`

the originalFont of member *whichFontMember*

Description Font cast member property; returns the exact name of the original font that was imported when the given cast member was created.

Example This statement displays the name of the font that was imported when cast member 11 was created:

```
put member(11).originalFont
-- "Monaco"
```

See also [recordFont](#), [bitmapSizes](#), [characterSet](#)

originH

Syntax `sprite(whichVectorOrFlashSprite).originH`

the originH of sprite *whichVectorOrFlashSprite*

`member(whichVectorOrFlashMember).originH`

the originH of member *whichVectorOrFlashMember*

Description Cast member and sprite property; controls the horizontal coordinate of a Flash movie or vector shape's origin point, in pixels. The value can be a floating-point value.

The origin point is the coordinate in a Flash movie or vector shape around which scaling and rotation occurs. The origin point can be set with floating-point precision using the separate `originH` and `originV` properties, or it can be set with integer precision using the single `originPoint` property.

You can set the `originH` property only if the `originMode` is set to `#point`.

This property can be tested and set. The default value is 0.

Note: This property must be set to the default value if the scaleMode property is set to #autoSize, or the sprite does not display correctly.

Example This sprite script uses the originMode property to set up a Flash movie sprite so its origin point can be set to a specific point. It then sets the horizontal and vertical origin points.

```
on beginSprite me
    sprite(spriteNum of me).originMode = #point
    sprite(spriteNum of me).originH = 100
    sprite(spriteNum of me).originV = 80
end
```

See also originV, originMode, originPoint, scaleMode

originMode

Syntax `sprite(whichFlashOrVectorShapeSprite).originMode`
the originMode of sprite *whichFlashOrVectorShapeSprite*
`member(whichFlashOrVectorShapeMember).originMode`
the originMode of member *whichFlashOrVectorShapeMember*

Description Cast member property and sprite property; sets the origin point around which scaling and rotation occurs, as follows:

- #center (default)—The origin point is at the center of the Flash movie.
- #topleft—The origin point is at the top left of the Flash movie.
- #point—The origin point is at a point specified by the originPoint, originH, and originV properties.

This property can be tested and set.

Note: This property must be set to the default value if the scaleMode property is set to #autoSize, or the sprite will not display correctly.

Example This sprite script uses the originMode property to set up a Flash movie sprite so its origin point can be set to a specific point. It then sets the horizontal and vertical origin points.

```
on beginSprite me
    sprite(spriteNum of me).originMode = #point
    sprite(spriteNum of me).originH = 100
    sprite(spriteNum of me).originV = 80
end
```

See also originH, originV, originPoint, scaleMode

originPoint

Syntax

```
sprite whichVectorOrFlashSprite.originPoint  
the originPoint of sprite whichVectorOrFlashSprite  
member(whichVectorOrFlashMember).originPoint  
the originPoint of member whichVectorOrFlashMember
```

Description

Cast member and sprite property; controls the origin point around which scaling and rotation occurs of a Flash movie or vector shape.

The `originPoint` property is specified as a Director point value: for example, `point(100,200)`. Setting a Flash movie or vector shape's origin point with the `originPoint` property is the same as setting the `originH` and `originV` properties separately. For example, setting the `originPoint` property to `point(50,75)` is the same as setting the `originH` property to 50 and the `originV` property to 75.

Director point values specified for the `originPoint` property are restricted to integers, whereas `originH` and `originV` can be specified with floating-point numbers. When you test the `originPoint` property, the point values are truncated to integers. As a rule of thumb, use the `originH` and `originV` properties for precision; use the `originPoint` property for speed and convenience.

You can set the `originPoint` property only if the `originMode` property is set to `#point`.

This property can be tested and set. The default value is 0.

Note: This property must be set to the default value if the `scaleMode` property is set to `#autoSize`, or the sprite will not display correctly.

Example

This sprite script uses the `originMode` property to set up a Flash movie sprite so its origin point can be set to a specific point. It then sets the origin points.

```
on beginSprite me  
    sprite(me.spriteNum).scaleMode = #showAll  
    sprite(me.spriteNum).originMode = #point  
    sprite(me.spriteNum).originPoint = point(100, 80)  
end
```

See also

`originH`, `originV`, `scaleMode`

originV

Syntax

```
sprite(whichVectorOrFlashSprite).originV  
the originV of sprite whichVectorOrFlashSprite  
member(whichVectorOrFlashMember).originV  
the originV of member whichVectorOrFlashMember
```

Description

Cast member and sprite property; controls the vertical coordinate of a Flash movie or vector shape's origin point around which scaling and rotation occurs, in pixels. The value can be a floating-point value.

The origin point can be set with floating-point precision using the separate `originH` and `originV` properties, or it can be set with integer precision using the single `originPoint` property.

You can set the `originV` property only if the `originMode` property is set to `#point`.

This property can be tested and set. The default value is 0.

Note: This property must be set to the default value if the `scaleMode` property is set to `#autoSize`, or the sprite does not display correctly.

Example

This sprite script uses the `originMode` property to set up a Flash movie sprite so its origin point can be set to a specific point. It then sets the horizontal and vertical origin points.

```
on beginSprite me  
    sprite(me.spriteNum).scaleMode = #showAll  
    sprite(me.spriteNum).originMode = #point  
    sprite(me.spriteNum).originH = 100  
    sprite(me.spriteNum).originV = 80  
end
```

See also

`originH`, `originPoint`, `scaleMode`

otherwise

Syntax	otherwise <i>statement(s)</i>
Description	Keyword; precedes instructions that Lingo performs when none of the earlier conditions in a case statement are met. This keyword can be used to alert users of out-of-bound input or invalid type, and can be very helpful in debugging during development.
Example	The following handler tests which key the user pressed most recently and responds accordingly: <ul style="list-style-type: none">• If the user pressed A, B, or C, the movie performs the corresponding action following the of keyword.• If the user pressed any other key, the movie executes the statement that follows the otherwise keyword. In this case, the statement is a simple alert. <pre>on keyDown case (the key) of "A": go to frame "Apple" "B", "C": puppetTransition 99 go to frame "Oranges" otherwise: alert "That is not a valid key." end case end keyDown</pre>

pageHeight

Syntax	member(<i>whichCastMember</i>).pageHeight the pageHeight of member <i>whichCastMember</i>
Description	Field cast member property; returns the height, in pixels, of the area of the field cast member that is visible on the Stage. This property can be tested but not set.
Example	This statement returns the height of the visible portion of the field cast member Today's News: put member("Today's News").pageHeight"

palette

Syntax

`member(whichCastMember).palette`

the palette of member *whichCastMember*

Description

Cast member property; for bitmap cast members only, determines which palette is associated with the cast member specified by *whichCastMember*.

This property can be tested and set.

Example

This statement displays the palette assigned to the cast member Leaves in the Message window:

```
put member("Leaves").palette"
```

paletteMapping

Syntax

the paletteMapping

Description

Movie property; determines whether the movie remaps (TRUE) or does not remap (FALSE, default) palettes for cast members whose palettes are different from the current movie palette. Its effect is similar to that of the Remap Palettes When Needed check box in the Movie Properties dialog box.

To display different bitmaps with different palettes simultaneously, set paletteMapping to TRUE. Director looks at each onscreen cast member's reference palette (the palette assigned in its Cast Member Properties dialog box) and, if it is different from the current palette, finds the closest match for each pixel in the new palette.

The colors of the nonmatching bitmap will be close to the original colors.

Remapping consumes processor time, and it's usually better to adjust the bitmap's palette in advance.

Remapping can also produce undesirable results. If the palette changes in the middle of a sprite span, the bitmap immediately remaps to the new palette and appears in the wrong colors. However, if anything refreshes the screen—a transition or a sprite moving across the Stage—then the affected rectangle on the screen appears in remapped colors.

Example

This statement tells the movie to remap the movie's palette whenever necessary:

```
set the paletteMapping = TRUE
```

paletteRef

Syntax

`member(whichCastMember). paletteRef`

the paletteRef

Description

Bitmap cast member property; determines the palette associated with a bitmap cast member. Built-in Director palettes are indicated by symbols (#systemMac, #rainbow, and so on). Palettes that are cast members are treated as cast member references. This behavior differs from that of the palette member property, which returns a positive number for cast palettes and negative numbers for built-in Director palettes.

This property can be tested and set.

Example

This statement assigns the Macintosh system palette to the bitmap cast member Shell:

```
member("Shell").paletteRef = #systemMac
```

pan (QTVR property)

Syntax

`pan of sprite whichQTVRSprite`

Description

QuickTime VR sprite property; the current pan of the QuickTime VR movie. The value is in degrees.

This property can be tested and set.

pan (sound property)

Syntax

`sound(channelNum).pan`

the pan of sound(*channelNum*)

Description

Property; indicates the left/right balance of the sound playing in sound channel *channelNum*. The range of values is from -100 to 100. -100 indicates only the left channel is heard. 100 indicate only the right channel is being heard. A value of 0 indicates even left/right balance, causing the sound source to appear to be centered. For mono sounds, pan affects which speaker (left or right) the sound plays through.

You can change the pan of a sound object at any time, but if the sound channel is currently performing a fade, the new pan setting doesn't take effect until the fade is complete.

To see an example of pan (sound property) used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This Lingo statement pans the sound in sound channel 2 from the left channel to the right channel:

```
repeat with x = -100 to 100
    sound(2).pan = x
end repeat
```

See also [fadeIn\(\)](#), [fadeOut\(\)](#), [fadeTo\(\)](#), [volume \(sound channel\)](#)

paragraph

Syntax *chunkExpression.paragraph[whichParagraph]*

chunkExpression.paragraph[firstParagraph..lastParagraph]

Description Text cast member property; this chunk expression allows access to different paragraphs within a text cast member.

The paragraph is delimited by a carriage return.

```
put member("AnimText").paragraph[3]
```

See also [line...of](#)

param()

Syntax *param(parameterPosition)*

Description Function; provides the value of a parameter passed to a handler. The expression *parameterPosition* represents the parameter's position in the arguments.

To avoid errors in a handler, this function can be used to determine the type of a particular parameter.

Example This handler accepts any number of arguments, adds all the numbers passed in as parameters, and then returns the sum:

```
on AddNumbers
    sum = 0
    repeat with currentParamNum = 1 to the paramCount
        sum = sum + param(currentParamNum)
    end repeat
    return sum
end
```

You would use it by passing in the values you wanted to add:

```
put AddNumbers(3, 4, 5, 6)
-- 18
put AddNumbers(5, 5)
-- 10
```

See also [getAt](#), [param\(\)](#), [paramCount\(\)](#), [return \(keyword\)](#)

paramCount()

Syntax the paramInt

Description Function; indicates the number of parameters sent to the current handler.

Example This statement sets the variable counter to the number of parameters that were sent to the current handler:

```
set counter = the paramInt
```

pass

Syntax pass

Description Command; passes an event message to the next location in the message hierarchy and enables execution of more than one handler for a given event.

The Director player for Java supports this command only within on keyDown and on keyUp handlers attached to editable sprites.

The pass command branches to the next location as soon as the command runs. Any Lingo that follows the pass command in the handler does not run.

By default, an event message stops at the first location containing a handler for the event, usually at the sprite level.

If you include the pass command in a handler, the event is passed to other objects in the hierarchy even though the handler would otherwise intercept the event.

Example This handler checks the keypresses being entered, and allows them to pass through to the editable text sprite if they are valid characters:

```
on keyDown me
    legalCharacters = "1234567890"
    if legalCharacters contains the key then
        pass
    else
        beep
    end if
end
```

See also stopEvent

pasteClipBoardInto

Syntax

`member(whichCastMember). pasteClipBoardInto()`

`pasteClipBoardInto member whichCastMember`

Description

Command; pastes the contents of the Clipboard into the cast member specified by *whichCastMember* and erases the exiting cast member. For example, pasting a bitmap into a field cast member makes the bitmap the cast member and erases the field cast member.

You can paste any item that is in a format that Director can use as a cast member. When you copy a string from another application, the string's formatting is not retained.

The `pasteClipBoardInto` command provides a convenient way to copy objects from other movies and from other applications into the Cast window. Because copied cast members must be stored in RAM, avoid using this command during playback in low memory situations.

Note: When you use this command in Shockwave, or in the authoring environment and projectors with the `safePlayer` property set to TRUE, a warning dialog will allow the user to cancel the paste operation.

Example

This statement pastes the Clipboard contents into the bitmap cast member Shrine:

`member("shrine").pasteClipBoardInto()`

See also

`safePlayer`

pathName (cast member property)

Syntax

`member(whichFlashMember).pathName`

the `pathName` of member *whichFlashMember*

Description

Cast member property; controls the location of an external file that stores the assets of a Flash movie cast member are stored. You can link a Flash movie to any path on a local or network drive or to a URL.

Setting the path of an unlinked cast member converts it to a linked cast member.

This property can be tested and set. The `pathName` property of an unlinked member is an empty string.

This property is the same as the `fileName` property for other member types, and you can use `fileName` instead of `pathName`.

Example This startMovie script creates a new Flash cast member using the new command, sets the newly created cast member's linked property so that the cast member's assets are stored in an external file, and then sets the cast member's pathName property to the location of a Flash movie on the World Wide Web:

```
on startMovie
    member(new(#flash)).pathName = \
    "http://www.someURL.com/myFlash.swf"
end
```

See also fileName (cast member property), linked

pathName (movie property)

This is obsolete. Use moviePath.

pattern

Syntax member(*whichCastMember*).pattern

the pattern of member *whichCastMember*

Description Cast member property; determines the pattern associated with the specified shape. Possible values are the numbers that correspond to the swatches in the Tools window's patterns palette. If the shape cast member is unfilled, the pattern is applied to the cast member's outer edge.

The Director player for Java can assign only the patterns for chips 1 and 15 in Director's patterns palette.

This property can be useful in Shockwave movies to change images by changing the tiling applied to a shape, allowing you to save memory required by larger bitmaps.

This property can be tested and set.

Example The following statements make the shape cast member Target Area a filled shape and assign it pattern 1, which is a solid color:

```
member("Target Area").filled = TRUE
member("Target Area").pattern = 1
```

Example This handler cycles through eight tiles, with each tile's number offset from the previous one, enabling you to create animation using smaller bitmaps:

```
on exitFrame
    currentPat = member("Background Shape").pattern
    nextPat = 57 + ((currentPat - 56) mod 8)
    member("Background Shape").pattern = nextPat
    go the frame
end
```

pause (movie playback)

This is obsolete. Use go to the frame.

pause() (sound playback)

Syntax sound(*channel/Num*).pause()

pause(sound(*channel/Num*))

Description This command suspends playback of the current sound in sound channel *channel/Num*. A subsequent play() command will resume playback.

Example This statement pauses playback of the sound cast member playing in sound channel 1.

sound(1).pause()

See also breakLoop(), isBusy(), play() (sound), playNext(), queue(), rewind(), status, stop() (sound)

pausedAtStart

Syntax member(*whichFlashOrDigitalVideoMember*).pausedAtStart

the pausedAtStart of member *whichFlashOrDigitalVideoMember*

Description Cast member property; controls whether the digital video or Flash movie plays when it appears on the Stage. If this property is TRUE, the digital video or Flash movie does not play when it appears. If this property is FALSE, it plays immediately when it appears.

For a digital video cast member, the property specifies whether the Paused at Start check box in the Digital Video Cast Member Properties dialog box is selected or not.

This property can be tested and set.

Example This statement turns on the Paused at Start check box in the Digital Video Cast Member Info dialog box for the QuickTime movie Rotating Chair:

member("Rotating Chair").pausedAtStart = TRUE

See also play

pause member

Syntax

```
member(whichCastMember). pause()  
pause member ("whichCastMember")
```

Description

Command; pauses the streaming of a Shockwave Audio (SWA) streaming cast member. When the sound is paused, the state member property equals 4. The portion of the sound that has already been downloaded and is available will continue to play until the cache runs out.

Example

This handler could be used for a Play or Pause button. If the sound is playing, the handler pauses the sound; otherwise, the handler plays the sound linked to the SWA streaming cast member soundSWA.

```
on mouseDown  
    whatState = member("soundSWA").state  
    if whatState = 3 then  
        member("soundSWA").pause()  
    else  
        member("soundSWA").play()  
    end if  
end
```

See also

play member, stop member

pause sprite

Syntax

```
sprite(whichGIFSpriteNumber). pause()  
pause(sprite whichGIFSpriteNumber)
```

Description

Command; causes an animated GIF sprite to pause in its playback and remain on the current frame.

Example

sprite(1).pause()

See also

resume sprite, rewind sprite

pauseState

Syntax

the pauseState

Description

Movie property; determines whether the pause command is currently pausing the movie (TRUE) or not (FALSE).

Because the pause command is obsolete, this property is not commonly used.

Example

This statement checks whether the movie is currently paused and, if it is paused, causes the movie to continue playing:

```
if the pauseState = TRUE then go the frame + 1
```

See also

pause (movie playback)

percentPlayed

Syntax

member(*whichCastMember*).percentPlayed

the percentPlayed of member *whichCastMember*

Description

Shockwave Audio (SWA) cast member property; returns the percentage of the specified SWA file that has actually played.

This property can be tested only after the SWA sound starts playing or has been preloaded by means of the preLoadBuffer command. This property cannot be set.

Example

This handler displays the percentage of the SWA streaming cast member Frank Sinatra that has played and puts the value in the field cast member Percent Played:

```
on exitFrame
    whatState = member("Frank Sinatra").state
    if whatState > 1 AND whatState < 9 then
        member("Percent Played").text = /
        string(member("Frank Sinatra").percentPlayed)
    end if
end
```

See also

percentStreamed

percentStreamed

Syntax

member(*whichCastMember*).percentStreamed

the percentStreamed of member *whichCastMember*

Description

Shockwave Audio (SWA) cast member property; for SWA streaming sounds, gets the percent of an SWA file already streamed from an HTTP or FTP server; or for Flash movie cast members, gets the percent of a Flash movie that has streamed into memory. This property is a value from 0 to 100%.

For SWA, this property differs from the percentPlayed property in that it includes the amount of the file that has been buffered but not yet played.

This property can be tested only after the SWA sound starts playing or has been preloaded by means of the preLoadBuffer command. This property cannot be set.

Example

This example displays the percentage of the SWA streaming cast member Ray Charles that has streamed and puts the value in a field:

```
on exitFrame
    whatState = member("Ray Charles").state
    if whatState > 1 AND whatState < 9 then
        member("Percent Streamed Display").text = \
        string(member("Ray Charles").percentStreamed)
    end if
end
```

Example This frame script keeps the playback head looping in the current frame so long as less than 60 percent of a Flash movie called Splash Screen has streamed into memory:

```
on exitFrame
    if member("Splash Screen").percentStreamed < 60 then
        go to the frame
    end if
end
```

See also [percentPlayed](#)

period

Syntax *timeoutObject.period*

Description Object property; the number of milliseconds between timeout events sent by the *timeOutObject* to the *timeout* handler.

This property can be tested and set.

Example This timeout handler decreases the timeout's period by one second each time it's invoked, until a minimum period of 2 seconds (2000 milliseconds) is reached:

```
on handleTimeout timeoutObject
    if timeoutObject.period > 2000 then
        timeoutObject.period = timeoutObject.period - 1000
    end if
end handleTimeout
```

See also [name](#) (timeout property), [persistent](#), [target](#), [time](#) (timeout object property), [timeout\(\)](#), [timeoutHandler](#), [timeoutList](#)

persistent

Syntax

timeoutObject.persistent

Description

Object property; determines whether the given *timeoutObject* is removed from the *timeoutList* when the current movie stops playing. If TRUE, *timeoutObject* remains active. If FALSE, the timeout object is deleted when the movie stops playing. The default value is FALSE.

Setting this property to TRUE allows a timeout object to continue generating timeout events in other movies. This is useful when one movie branches to another with the go to movie command.

Example

This prepareMovie handler creates a timeout object that will remain active after the declaring movie stops playing:

```
on prepareMovie
    -- Make a timeout object that sends an event every 60 minutes.
    timeout("reminder").new(1000 * 60 * 60, #handleReminder)
    timeout("reminder").persistent = TRUE
end
```

See also

[name](#) (*timeout* property), [period](#), [target](#), [time](#) (*timeout object* property), [timeout\(\)](#), [timeoutHandler](#), [timeoutList](#)

PI

Syntax

PI

Description

Constant; returns the value of pi (π), the ratio of a circle's circumference to its diameter, as a floating-point number. The value is rounded to the number of decimal places set by the *floatPrecision* property.

Example

This statement uses the PI constant as part of an equation for calculating the area of a circle:

```
set vArea = PI*power(vRadius,2)
```

picture (cast member property)

Syntax

`member(whichCastMember).picture`

the picture of member *whichCastMember*

Description

Cast member property; determines which image is associated with a bitmap, text, or PICT cast member. To update changes to a cast member's registration point or update changes to an image after relinking it using the `fileName` property, use the following statement:

`member(whichCastMember).picture = member(whichCastMember).picture`

where you replace *whichCastMember* with the name or number of the affected cast member.

Because changes to cast members are stored in RAM, this property is best used during authoring. Avoid setting it in projectors.

The property can be tested and set.

Example

This statement sets the variable named `pictHolder` to the image in the cast member named `Sunset`:

`pictHolder = member("Sunset").picture`

See also

`type` (sprite property)

picture (window property)

Syntax

`the stage.picture`

the picture of the stage

`window whichWindow.picture`

the picture of window *whichWindow*

Description

Window property; this property provides a way to get a picture of the current contents of a window (either the Stage window or a movie in a window).

You can apply the resulting bitmap data to an existing bitmap or use it to create a new one.

This property can be read and but not set.

Example

This statement grabs the current content of the Stage and places it into a bitmap cast member:

`member("Stage image").picture = (the stage).picture`

See also

`media`, `picture` (cast member property)

pictureP()

Syntax

pictureP(*pictureValue*)

Description

Function; reports whether the state of the picture member property for the specified cast member is TRUE (1) or FALSE (0).

Because pictureP doesn't directly check whether a picture is associated with a cast member, you must test for a picture by checking the cast member's picture member property.

Example

The first statement assigns the value of the picture member property for the cast member Shrine, which is a bitmap, to the variable *pictureValue*. The second statement checks whether Shrine is a picture by checking the value assigned to *pictureValue*.

```
set pictureValue to the picture of member "Shrine"  
put pictureP(pictureValue)
```

The result is 1, which is the numerical equivalent of TRUE.

platform

Syntax

the platform

Description

System property; indicates the platform type for which the projector was created.

This property can be tested but not set.

Possible values are the following:

Possible value	Corresponding platform
Macintosh,PowerPC	PowerPC Macintosh
Windows,32	Windows 95 or Windows NT

For forward compatibility and to allow for addition of values, it is better to test the platform by using contains.

When the movie plays back as a converted Java applet, this property's value indicates the browser and operating system in which the applet is playing. The property's value has the following syntax when the movie plays back as an applet:

Java javaVersion, browser, operatingSystem

The following are the possible values for this property's parameters:

- *javaVersion*: 1.0 or 1.1
- *browser*: IE, Netscape, or UnknownBrowser
- *operatingSystem*: Macintosh, Windows, or UnknownOS

For example, if an applet is playing in Microsoft Internet Explorer with Java 1.1 in Windows, platform has the value Java 1.1, IE, Windows.

Example This statement checks whether a projector was created for Windows 95 or Windows NT:

```
on exitFrame
    if the platform contains "Windows,32" then
        castLib("Win95 Art").name = "Interface"
    end if
end
```

See also [runMode](#)

play

Syntax `sprite(whichFlashSprite).play()`

`play [frame] whichFrame`

`play movie whichMovie`

`play frame whichFrame of movie whichMovie`

`play sprite whichFlashSprite`

Description Command; branches the playback head to the specified frame of the specified movie or starts a Flash movie sprite playing. For the former, the expression *whichFrame* can be either a string marker label or an integer frame number. The expression *whichMovie* must be a string that specifies a movie file. When the movie is in another folder, *whichMovie* must specify a path.

The play command is like the go to command, except that when the current sequence finishes playing, play automatically returns the playback head to the frame where play was called.

If play is issued from a frame script, the playback head returns to the next frame; if play is issued from a sprite script or handler, the playback head returns to the same frame. A play sequence ends when the playback head reaches the end of the movie or when the play done command is issued.

To play a movie from a URL, use downloadNetThing or preloadNetThing() to download the file to a local disk first, and then use play to play the movie on the local disk to minimize download time.

You can use the play command to play several movies from a single handler. The handler is suspended while each movie plays but resumes when each movie is finished. Contrast this with a series of go commands that, when called from a handler, play the first frame of each movie. The handler is not suspended while the movie plays but immediately continues executing.

When play is used to play a Flash movie sprite, the Flash movie plays from its current frame if it is stopped or from its first frame if it is already on the last frame.

Each play command needs a matching play done command to avoid using up memory if the original calling script isn't returned to. To avoid this memory consumption, you can use a global variable to record where the movie should return to.

Example This statement moves the playback head to the marker named blink:

```
play "blink"
```

Example This statement moves the playback head to the next marker:

```
play marker(1)
```

Example This statement moves the playback head to a separate movie:

```
play movie "My Drive:More Movies:" & newMovie
```

Example This frame script checks to see if the Flash movie sprite in channel 5 is playing, and if it is not, it starts the movie:

```
on enterFrame
    if not sprite(5).playing then
        sprite(5).play()
    end if
end
```

See also [downloadNetThing](#)

play() (sound)

Syntax

```
sound(channelNum).play()  
sound(channelNum).play(member (whichMember))  
sound(channelNum).play([#member: member(whichmember), {#startTime: milliseconds,  
#endTime: milliseconds, #loopCount: numberOfLoops, #loopStartTime: milliseconds,  
#loopEndTime: milliseconds, #preloadTime: milliseconds}])
```

Description

This function begins playing any sounds queued in *soundObject*, or queues and begins playing the given member.

Sound members take some time to load into RAM before they can begin playback. It's recommended that you queue sounds with *queue()* before you want to begin playing them and then use the first form of this function. The second two forms do not take advantage of the pre-loading accomplished with the *queue()* command.

By using an optional property list, you can specify exact playback settings for a sound. These properties may be optionally set:

Property	Description
#member	The sound cast member to queue. This property must be provided; all others are optional.
#startTime	The time within the sound at which playback begins, in milliseconds. The default is the beginning of the sound. See startTime.
#endTime	The time within the sound at which playback ends, in milliseconds. The default is the end of the sound. See endTime.
#loopCount	The number of times to play a loop defined with #loopStartTime and #loopEndTime. The default is 1. See loopCount.
#loopStartTime	The time within the sound to begin a loop , in milliseconds. See loopStartTime.
#loopEndTime	The time within the sound to end a loop, in milliseconds. See loopEndTime.
#preloadTime	The amount of the sound to buffer before playback, in milliseconds. See preloadTime.

To see an example of play() (sound) used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement plays cast member introMusic in sound channel 1:

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

Example This statement plays cast member creditsMusic in sound channel 2. Playback begins 4 seconds into the sound and ends 15 seconds into the sound. The section from 10.5 seconds to 14 seconds loops 6 times.

```
sound(2).play([#member: member("creditsMusic"), #startTime: 4000, \
#endTime: 15000, #loopCount: 6, #loopStartTime: 10500, #loopEndTime: 14000])
```

See also setPlaylist(), isBusy(), pause(), playNext(), preloadTime, queue(), rewind(), stop()

playBackMode

Syntax

```
sprite(whichFlashSprite).playBackMode  
the playBackMode of sprite whichFlashSprite  
member(whichGIFAnimMember).playBackMode  
the playBackMode of member whichGIFAnimMember
```

Description

Cast member and sprite property; controls the tempo of a Flash movie or animated GIF cast member with the following values:

- #normal (*default*)—Plays the Flash movie or GIF file as close to the original tempo as possible.
- #lockStep—Plays the Flash movie or GIF file frame for frame with the Director movie.
- #fixed—Plays the Flash movie or GIF file at the rate specified by the fixedRate property.

This property can be tested and set.

Example

This sprite script sets the frame rate of a Flash movie sprite to match the frame rate of the Director movie:

```
property spriteNum  
on beginSprite me  
    sprite(spriteNum).playBackMode = #lockStep  
end
```

See also

fixedRate

play done

Syntax

play done

Description

Command; ends the sequence started with the most recent play command. The play done command returns the playback head to where the calling sequence initiated the play command. If play is issued from a frame script, the playback head returns to the next frame; if play is issued from a sprite script, the playback head returns to the same frame.

Each play command needs a matching play done command to avoid using up memory if the original calling script isn't returned to. To avoid this memory consumption, you can use a global variable to record where the movie should return to.

The play done command has no effect in a movie that is playing in a window.

Example This handler returns the playback head to the frame of the movie that was playing before the current movie started:

```
on exitFrame  
    play done  
end
```

See also [play](#)

playing

Syntax `sprite(whichFlashSprite).playing`

the playing of sprite *whichFlashSprite*

Description Flash sprite property; indicates whether a Flash movie is playing (TRUE) or stopped (FALSE).

This property can be tested but not set.

Example This frame script checks to see if the Flash movie sprite in channel 5 is playing and, if it is not, starts the movie:

```
on enterFrame  
    if not sprite(5).playing then  
        sprite(5).play()  
    end if  
end
```

play member

Syntax `member(whichCastMember).play()`

play member *whichCastMember*

Description Command; begins playback of a Shockwave Audio (SWA) streaming cast member.

If the sound has not been preloaded by means of the `preLoadBuffer` command, the SWA sound preloads before playing begins. When the sound is playing, the state member property equals 3.

Be aware that Xtras to support this functionality must be included when playing back a streaming sound.

Example This handler begins the playback of the cast member Big Band:

```
on mouseDown  
    member("Big Band").play()  
end
```

See also [pause member](#), [stop member](#)

playNext()

Syntax

```
sound(channelNum).playNext()  
playNext(sound(channelNum))
```

Description

This command immediately interrupts playback of the current sound playing in the given sound channel and begins playing the next queued sound. If no more sounds are queued in the given channel, the sound simply stops playing.

Example

This statement plays the next queued sound in sound channel 2.

```
sound(2).playNext()
```

See also

pause() (sound playback), play() (sound), stop() (sound)

point()

Syntax

```
point(horizontal, vertical)
```

Description

Function and data type; yields a point that has the horizontal coordinate specified by *horizontal* and the vertical coordinate specified by *vertical*.

A point has a locH and a locV property. Point coordinates can be changed by arithmetic operations.

To see an example of point() used in a completed movie, see the Imaging and Vector Shapes movies in the Learning\Lingo Examples folder inside the Director application folder.

Example

This statement sets the variable lastLocation to the point (250, 400):

```
set lastLocation = point(250, 400)
```

Example

This statement adds 5 pixels to the horizontal coordinate of the point assigned to the variable myPoint:

```
myPoint.locH = myPoint.locH + 5
```

Example

These statements set a sprite's Stage coordinates to mouseH and mouseV plus 10 pixels. The two statements are equivalent.

```
sprite(the clickOn).loc = point(the mouseH, the mouseV) + point(10, 10)  
sprite(the clickOn).loc = the mouseLoc + 10
```

Example	This handler moves a named sprite to the location that the user clicks:
	end mouseDown
	on mouseDown
	-- Set these variables as needed for your own movie

```
theSprite = 1 -- Set the sprite that should move
steps = 40 -- Set the number of steps to get there
initialLoc = sprite(theSprite).loc
delta = (the clickLoc - initialLoc) / steps
repeat with i = 1 to steps
    sprite(theSprite).loc = initialLoc + (i * delta)
    updateStage
end repeat
end mouseDown
```

See also	mouseLoc, flashToStage(), rect(), stageToFlash()
-----------------	--

pointInHyperlink()

Syntax	sprite(<i>whichSpriteNumber</i>).pointInHyperlink(<i>point</i>) pointInHyperlink(<i>sprite whichSpriteNumber</i> , <i>point</i>)
---------------	---

Description	Text sprite function; returns a value (TRUE or FALSE) that indicates whether the specified point is within a hyperlink in the text sprite. Typically, the point used is the cursor position. This is useful for setting custom cursors.
--------------------	---

See also	cursor (command), mouseLoc
-----------------	----------------------------

pointToChar()

Syntax	pointToChar(sprite <i>spriteNumber</i> , <i>pointToTranslate</i>)
---------------	--

Description	Function; returns an integer representing the character position located within the text or field sprite <i>spriteNumber</i> at screen coordinate <i>pointToTranslate</i> , or returns -1 if the point is not within the text.
--------------------	--

This function can be used to determine the character under the cursor.

Example	This Lingo displays the number of the character being clicked, as well as the letter, in the Message window:
----------------	--

```
property spriteNum

on mouseDown me
    pointClicked = the mouseLoc
    currentMember = sprite(spriteNum).member
    charNum = sprite(spriteNum).pointToChar(pointClicked)
    actualChar = currentMember.char[charNum]
    put "Clicked character" && charNum & ", the letter" && actualChar
end
```

See also	mouseLoc, pointToWord(), pointToItem(), pointToLine(), pointToParagraph()
-----------------	---

pointToItem()

Syntax

```
sprite(whichSpriteNumber).pointToItem(pointToTranslate)  
pointToItem(sprite spriteNumber, pointToTranslate)
```

Description

Function; returns an integer representing the item position in the text or field *sprite spriteNumber* at screen coordinate *pointToTranslate*, or returns -1 if the point is not within the text. Items are separated by the *itemDelimiter* property, which is set to a comma by default.

This function can be used to determine the item under the cursor.

Example

This Lingo displays the number of the item being clicked, as well as the text of the item, in the Message window:

```
property spriteNum  
  
on mouseDown me  
    pointClicked = the mouseLoc  
    currentMember = sprite(spriteNum).member  
    itemNum = sprite(spriteNum).pointToItem(pointClicked)  
    itemText = currentMember.item[itemNum]  
    put "Clicked item" && itemNum & ", the text" && itemText  
end
```

See also

itemDelimiter, *mouseLoc*, *pointToChar()*, *pointToWord()*, *pointToItem()*, *pointToLine()*, *pointToParagraph()*

pointToLine()

Syntax

```
sprite(whichSpriteNumber).pointToLine(pointToTranslate)  
pointToLine(sprite spriteNumber, pointToTranslate)
```

Description

Function; returns an integer representing the line position in the text or field *sprite spriteNumber* at screen coordinate *pointToTranslate*, or returns -1 if the point is not within the text. Lines are separated by carriage returns in the text or field cast member.

This function can be used to determine the line under the cursor.

Example This Lingo displays the number of the line being clicked, as well as the text of the line, in the Message window:

```
property spriteNum  
  
on mouseDown me  
    pointClicked = the mouseLoc  
    currentMember = sprite(spriteNum).member  
    lineNum = sprite(spriteNum).pointToLine(pointClicked)  
    lineText = currentMember.line[lineNum]  
    put "Clicked line" && lineNum & ", the text" && lineText  
end
```

See also itemDelimiter, mouseLoc, pointToChar(), pointToWord(), pointToItem(), pointToLine(), pointToParagraph()

pointToParagraph()

Syntax
`sprite(whichSpriteNumber).pointToParagraph(pointToTranslate)`
`pointToParagraph(sprite spriteNumber, pointToTranslate)`

Description Function; returns an integer representing the paragraph number located within the text or field sprite *spriteNumber* at screen coordinate *pointToTranslate*, or returns -1 if the point is not within the text. Paragraphs are separated by carriage returns in a block of text.

This function can be used to determine the paragraph under the cursor.

Example This Lingo displays the number of the paragraph being clicked, as well as the text of the paragraph, in the message window:

```
property spriteNum  
  
on mouseDown me  
    pointClicked = the mouseLoc  
    currentMember = sprite(spriteNum).member  
    paragraphNum = sprite(spriteNum).pointToParagraph(pointClicked)  
    paragraphText = currentMember.paragraph[paragraphNum]  
    put "Clicked paragraph" && paragraphNum & ", the text" && paragraphText  
end
```

See also itemDelimiter, mouseLoc, pointToChar(), pointToWord(), pointToItem(), pointToLine()

pointToWord()

Syntax

```
sprite(whichSpriteNumber).pointToWord(pointToTranslate)  
pointToWord(sprite spriteNumber, pointToTranslate)
```

Description

Function; returns an integer representing the number of a word located within the text or field sprite *spriteNumber* at screen coordinate *pointToTranslate*, or returns -1 if the point is not within the text. Words are separated by spaces in a block of text.

This function can be used to determine the word under the cursor.

Example

This Lingo displays the number of the word being clicked, as well as the text of the word, in the Message window:

```
property spriteNum  
  
on mouseDown me  
    pointClicked = the mouseLoc  
    currentMember = sprite(spriteNum).member  
    wordNum = sprite(spriteNum).pointToWord(pointClicked)  
    wordText = currentMember.word[wordNum]  
    put "Clicked word" && wordNum & ", the text" && wordText  
end
```

See also

itemDelimiter, mouseLoc, pointToChar(), pointToItem(), pointToLine(), pointToParagraph()

posterFrame

Syntax

```
member(whichFlashMember).posterFrame  
the posterFrame of member whichFlashMember
```

Description

Flash cast member property; controls which frame of a Flash movie cast member is used for its thumbnail image. This property specifies an integer corresponding to a frame number in the Flash movie.

This property can be tested and set. The default value is 1.

Example

This handler accepts a reference to a Flash movie cast member and a frame number as parameters, and it then sets the thumbnail of the specified movie to the specified frame number:

```
on resetThumbnail whichFlashMovie, whichFrame  
    member(whichFlashMovie).posterFrame = whichFrame  
end
```

postNetText

Syntax

```
postNetText(url, propertyList {,serverOSString} {,serverCharSetString})  
postNetText(url, postText {,serverOSString} {,serverCharSetString})
```

Description

Command; sends a POST request to *url*, which is an HTTP URL, with *postText* as the data.

This command is similar to `getNetText()`. As with `getNetText()`, the server's response is returned by `netTextResult(netID)` once `netDone(netID)` becomes 1, and if `netError(netID)` is 0, or okay.

When a property list is used instead of a string, the information is sent in the same way a browser posts an HTML form, with METHOD=POST. This facilitates the construction and posting of form data within a Director title. Property names correspond to HTML form field names and property values to field values.

The property list can use either strings or symbols as the property names. If a symbol is used, it is automatically converted to a string without the # at the beginning. Similarly, a numeric value is converted to a string when used as the value of a property.

Note: If a program uses the alternate form—a string instead of property list—the string *postText* is sent to the server as an HTTP POST request using MIME type "text/plain." This will be convenient for some applications, but is not compatible with HTML forms posting.

The optional parameter *serverOSString* defaults to UNIX but may be set to Windows or Mac and translates any carriage returns in the *postText* argument into those used on the server to avoid confusion. For most applications, this setting is unnecessary because line breaks are usually not used in form responses.

The optional parameter *serverCharSetString* applies only if the user is running on a Shift-JIS (Japanese) system. Its possible settings are "JIS", "EUC", "ASCII", and "AUTO". Posted data is converted from Shift-JIS to the named character set. Returned data is handled exactly as by `getNetText()` (converted from the named character set to Shift-JIS). If you use "AUTO", the posted data from the local character set is not translated; the results sent back by the server are translated as they are for `getNetText()`. "ASCII" is the default if *serverCharSetString* is omitted. "ASCII" provides no translation for posting or results.

The optional arguments may be omitted without regard to position.

This command also has an additional advantage over `getNetText()`: a `postNetText()` query can be arbitrarily long, whereas the `getNetText()` query is limited to the length of a URL (1K or 4K, depending on the browser).

Note: If you use `postNetText` to post data to a domain different from the one the movie is playing from, the movie will display a security alert when playing back in Shockwave.

To see an example of postNetText used in a completed movie, see the Forms and Post movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement omits the *serverCharSetString* parameter:

```
netID = postNetText("www.mydomain.com\database.cgi", "Bill Jones", "Win")
```

Example This example generates a form from user-entry fields for first and last name, along with a Score. Note that both *serverOSString* and *serverCharSetString* have been omitted:

```
lastName = member("Last Name").text  
firstName = member("First Name").text  
totalScore = member("Current Score").text  
infoList = [{"FName":firstName, "LName":lastName, "Score":totalScore}]  
netID = postNetText("www.mydomain.com\userbase.cgi", infoList)
```

See also [getNetText\(\)](#), [netTextResult\(\)](#), [netDone\(\)](#), [netError\(\)](#)

power()

Syntax `power(base, exponent)`

Description Math function; calculates the value of the number specified by *base* to the exponent specified by *exponent*.

Example This statement sets the variable vResult to the value of 4 to the third power:

```
set vResult = power(4,3)
```

preLoad (command)

Syntax

```
preLoad  
preLoad toFrameNum  
preLoad fromFrame, toFrameNum
```

Description

Command; preloads cast members in the specified frame or range of frames into memory and stops when memory is full or when all of the specified cast members have been preloaded, as follows:

- When used without arguments, the command preloads all cast members used from the current frame to the last frame of a movie.
- When used with one argument, *toFrame*, the command preloads all cast members used in the range of frames from the current frame to the frame *toFrameNum*, as specified by the frame number or label name.
- When used with two arguments, *fromFrame* and *toFrameNum*, preloads all cast members used in the range of frames from the frame *fromFrame* to the frame *toFrameNum*, as specified by the frame number or label name.

The preLoad command also returns the number of the last frame successfully loaded. To obtain this value, use the result function.

Example

This statement preloads the cast members used from the current frame to the frame that has the next marker:

```
preLoad marker (1)
```

Example

This statement preloads the cast members used from frame 10 to frame 50:

```
preLoad 10, 50
```

See also

[preLoadMember](#)

preLoad (cast member property)

Syntax

```
member(whichCastMember).preLoad  
the preLoad of member whichCastMember
```

Description

Cast member property; determines whether the digital video cast member specified by *whichCastMember* can be preloaded into memory (TRUE) or not (FALSE, default). The TRUE status has the same effect as selecting Enable Preload in the Digital Video Cast Member Properties dialog box.

For Flash movie cast members, this property controls whether a Flash movie must load entirely into RAM before the first frame of a sprite is displayed (TRUE), or whether the movie can stream into memory as it plays (FALSE, default). This property works only for linked Flash movies whose assets are stored in an external file; it has no effect on members whose assets are stored in the cast. The streamMode and bufferSize properties determine how the cast member is streamed into memory.

This property can be tested and set.

Example This statement reports in the Message window whether the QuickTime movie Rotating Chair can be preloaded into memory:

```
put member("Rotating Chair").preload
```

Example This startMovie handler sets up a Flash movie cast member for streaming and then sets its bufferSize property:

```
on startMovie
    member("Flash Demo").preLoad = FALSE
    member("Flash Demo").bufferSize = 65536
end
```

See also bufferSize, streamMode

preLoadBuffer member

Syntax `member(whichCastMember).preLoadBuffer()`

`preLoadBuffer` member *whichCastMember*

Description Command; preloads part of a specified Shockwave Audio (SWA) file into memory. The amount preloaded is determined by the preLoadTime property. This command works only if the SWA cast member is stopped.

When the preLoadBuffer command succeeds, the state member property equals 2.

Most SWA cast member properties can be tested only after the preLoadBuffer command has completed successfully. These properties include: cuePointNames, cuePointTimes, currentTime, duration, percentPlayed, percentStreamed, bitRate, sampleRate, and numChannels.

Example This statement loads the cast member Mel Torme into memory:

```
member("Mel Torme").preLoadBuffer()
```

See also preLoadTime

preLoadEventAbort

Syntax	the preLoadEventAbort
Description	Movie property; specifies whether pressing keys or clicking the mouse can stop the preloading of cast members (TRUE) or not (FALSE, default). This property can be tested and set. Setting this property affects the current movie.
Example	This statement lets the user stop the preloading of cast members by pressing keys or clicking the mouse: set the preLoadEventAbort = TRUE
See also	preLoad (command), preLoadMember

preLoadMember

Syntax	preLoadMember member(<i>whichCastMember</i>).preLoad() preLoadMember <i>whichCastMember</i> member(<i>fromCastmember</i>).preLoad(<i>toCastMember</i>) preLoadMember <i>fromCastmember</i> , <i>toCastmember</i>
Description	Command; preloads cast members and stops when memory is full or when all of the specified cast members have been preloaded. The preLoadMember command returns the cast member number of the last cast member successfully loaded. To obtain this value, use the result function. When used without arguments, preLoadMember preloads all cast members in the movie. When used with the <i>whichCastMember</i> argument, preLoadMember preloads just that cast member. If <i>whichCastMember</i> is an integer, only the first cast is referenced. If <i>whichCastMember</i> is a string, the first member with the string as its name will be used. When used with the arguments <i>fromCastmember</i> and <i>toCastmember</i> , the preLoadMember command preloads all cast members in the range specified by the cast member numbers or names.
Example	This statement preloads cast member 20 of the first (internal) cast: member(20).preLoad()
Example	This statement preloads cast member Shrine and the 10 cast members after it in the Cast window: member("Shrine").preLoad(member("Shrine").number + 10)
Example	To preload a specific member of a specific cast, use the following syntax: member("John Boy", "Family Members").preLoad()

preLoadMode

Syntax `castLib(whichCast).preLoadMode`

the preLoadMode of castLib *whichCast*

Description Cast member property; determines the specified cast's preload mode and has the same effect as setting Load Cast in the Cast Properties dialog box. Possible values are the following:

- 0—Load cast when needed.
- 1—Load cast before frame 1.
- 2—Load cast after frame 1.

The default value for cast members is 0, when needed.

An on prepareMovie handler is usually a good place for Lingo that determines when cast members are loaded.

This property can be tested and set.

Example The following statement tells Director to load the members of the cast Buttons before the movie enters frame 1:

```
CastLib("Buttons").preLoadMode = 1
```

preLoadMovie

Syntax `preLoadMovie whichMovie`

Description Command; preloads the data and cast members associated with the first frame of the specified movie. Preloading a movie helps it start faster when it is started by a go to movie or play movie command.

To preload cast members from a URL, use preloadNetThing() to load the cast members directly into the cache, or use downloadNetThing to load a movie on a local disk from which you can load the movie into memory and minimize downloading time.

Example This statement preloads the movie Introduction, which is located in the same folder as the current movie:

```
preLoadMovie "Introduction"
```

preloadNetThing()

Syntax

preloadNetThing (*url*)

Description

Function; preloads a file from the Internet to the local cache so it can be used later without a download delay. Replace *url* with the name of any valid Internet file, such as a Director movie, graphic, or FTP server location. The return value is a network ID that you can use to monitor the progress of the operation.

The Director player for Java doesn't support this command because Java's security model doesn't allow writing to the local disk.

The preloadNetThing() function downloads the file while the current movie continues playing. Use netDone() to find out whether downloading is finished.

After an item is downloaded, it can be displayed immediately because it is taken from the local cache rather than from the network.

Although many network operations can be active at a time, running more than four concurrent operations usually slows down performance unacceptably.

Neither the cache size nor the Check Documents option in a browser's preferences affects the behavior of the preloadNetThing function.

The preloadNetThing() function does not parse a Director file's links. Thus, even if a Director file is linked to casts and graphic files, preloadNetThing() downloads only the Director file. You still must preload other linked objects separately.

Example

This statement uses preloadNetThing() and returns the network ID for the operation:

```
set mynetid = preloadNetThing("http://www.yourserver.com/menupage/mymovie.dir")
```

After downloading is complete, you can navigate to the movie using the same URL. The movie will be played from the cache instead of the URL, since it's been loaded in the cache.

See also

netDone()

preLoadRAM

Syntax	the preLoadRAM
Description	System property; specifies the amount of RAM that can be used for preloading a digital video. This property can be set and tested. This property is useful for managing memory, limiting digital video cast members to a certain amount of memory, so that other types of cast members can still be preloaded. When preLoadRAM is FALSE, all available memory can be used for preloading digital video cast members. However, it's not possible to reliably predict how much RAM a digital video will require once it is preloaded, because memory requirements are affected by the content of the movie, how much compression was performed, the number of keyframes, changing imagery, and so on. It is usually safe to preload the first couple of seconds of a video and then continue streaming from that point on. If you know the data rate of your movie, you can estimate the setting for preLoadRAM. For example, if your movie has a data rate of 300K per second, set preLoadRAM to 600K if you want to preload the first 2 seconds of the video file. This is only an estimate, but it works in most situations.
Example	This statement sets preLoadRAM to 600K, to preload the first 2 seconds of a movie with a data rate of 300K per second: set the preLoadRAM = 600
See also	loop (keyword), next

preLoadTime

Syntax	member(<i>whichCastMember</i>).preLoadTime the preLoadTime of member <i>whichCastMember</i> sound(<i>channelNum</i>).preLoadTime
Description	Cast member and sound channel property; for cast members, specifies the amount of the Shockwave Audio (SWA) streaming cast member to download, in seconds, before playback begins or when a preLoadBuffer command is used. The default value is 5 seconds. This property can be set only when the SWA streaming cast member is stopped. For sound channels, the value is for the given sound in the queue or the currently playing sound if none is specified.

Example This handler sets the preload download time for the SWA streaming cast member Louis Armstrong to 6 seconds. The actual preload occurs when a preLoadBuffer or play command is issued.

```
on mouseDown
    member("Louis Armstrong").stop()
    member("Louis Armstrong").preLoadTime = 6
end
```

Example This statement returns the preLoadTime of the currently playing sound in sound channel 1:

```
put sound(1).preLoadTime
```

See also preLoadBuffer member

on prepareFrame

Syntax on prepareFrame

```
    statement(s)
end
```

Description System message and event handler; contains statements that run immediately before the current frame is drawn.

Unlike beginSprite and endSprite events, a prepareFrame event is generated each time the playback head enters a frame.

The on prepareFrame handler is a useful place to change sprite properties before the sprite is drawn.

If used in a behavior, the on prepareFrame handler receives the reference me.

The go, play, and updateStage commands are disabled in an on prepareFrame handler.

Example This handler sets the locH property of the sprite that the behavior is attached to:

```
on prepareFrame me
    sprite(me.spriteNum).locH = the mouseH
end
```

See also on enterFrame

on prepareMovie

Syntax

```
on prepareMovie  
    statement(s)  
end
```

Description

System message and event handler; contains statements that run after the movie preloads cast members but before the movie does the following:

- Creates instances of behaviors attached to sprites in the first frame that plays.
- Prepares the first frame that plays, including drawing the frame, playing any sounds, and executing transitions and palette effects.

New global variables used for sprite behaviors in the first frame should be initialized in the on prepareMovie handler. Global variables already set by the previous movie do not need to be reset.

An on prepareMovie handler is a good place to put Lingo that creates global variables, initializes variables, plays a sound while the rest of the movie is loading into memory, or checks and adjusts computer conditions such as color depth.

The go, play, and updateStage commands are disabled in an on prepareMovie handler.

Example

This handler creates a global variable when the movie starts:

```
on prepareMovie  
    global currentScore  
    set currentScore = 0  
end
```

See also

on enterFrame, on startMovie

previous

See

go previous

printFrom

Syntax

```
printFrom fromFrame {,toFrame} {,reduction}
```

Description

Command; prints whatever is displayed on the Stage in each frame, whether or not the frame is selected, starting at the frame specified by *fromFrame*. Optionally, you can supply *toFrame* and a reduction value (100%, 50%, or 25%).

The frame being printed need not be currently displayed. This command always prints at 72 dots per inch (dpi), bitmaps everything on the screen (text will not be as smooth in some cases), prints in portrait (vertical) orientation, and ignores Page Setup settings. For more flexibility when printing from within Director, see PrintOMatic Lite Xtra, which is on the installation disk.

Example

This statement prints what is on the Stage in frame 1:

```
printFrom 1
```

This statement prints what is on the Stage in every frame from the marker Intro to the marker Tale. The reduction is 50%.

```
printFrom label("Intro"), label("Tale"), 50
```

property

Syntax

```
property {property1} {,property2} {,property3} {...}
```

Description

Keyword; declares the properties specified by *property1*, *property2*, and so on as property variables.

Declare property variables at the beginning of the parent script or behavior script. You can access them from outside the parent script or behavior script by using the the operator.

Note: The spriteNum property is available to all behaviors and simply needs to be declared to be accessed.

You can refer to a property within a parent script or behavior script without using the me keyword. However, to refer to a property of a parent script's ancestor, use the form me.property.

For behaviors, properties defined in one behavior script are available to other behaviors attached to the same sprite.

You can directly manipulate a child object's property from outside the object's parent scripts through syntax similar to that for manipulating other properties. For example, this statement sets the motionStyle property of a child object:

```
set the motionStyle of myBouncingObject to #frenetic
```

Use the count function to determine the number of properties within the parent script of a child object. Retrieve the name of these properties by using getPropAt. Add properties to an object by using setaProp().

To see an example of property used in a completed movie, see the Parent Scripts movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement lets each child object created from a single parent script have its own location and velocity setting:

```
property location, velocity
```

Example This parent script handler declares pMySpriteNum a property to make it available:

```
-- script Elder  
property pMyChannel
```

```
on new me, whichSprite  
    me.pMyChannel = whichSprite  
    return me  
end
```

The original behavior script sets up the ancestor and passes the spriteNum property to all behaviors:

```
property spriteNum  
property ancestor  
  
on beginSprite me  
    set ancestor = new(script "Elder", spriteNum)  
end
```

See also me, ancestor, spriteNum

proxyServer

Syntax

```
proxyServer serverType, "ipAddress", portNum  
proxyServer()
```

Description Command; sets the values of an FTP or HTTP proxy server, as follows:

- *serverType*—#ftp or #http
- *ipAddress*—A string containing the IP address
- *portNum*—The integer value of the port number

If you use the syntax proxyServer(), this element returns the settings of an FTP or HTTP proxy server.

Example This statement sets up an HTTP proxy server at IP address 197.65.208.157 using port 5:

```
proxyServer #http,"197.65.208.157",5
```

Example This statement returns the port number of an HTTP proxy server:

```
put proxyServer(#http,#port)
```

If no server type is specified, the function returns 1.

Example This statement returns the IP address string of an HTTP proxy server:

```
put proxyServer(#http)
```

Example This statement turns off an FTP proxy server:

```
proxyServer #ftp,#stop
```

ptToHotSpotID()

Syntax

```
ptToHotSpotID(whichQTVRSprite, point)
```

Description QuickTime VR function; returns the ID of the hotspot, if any, that is at the specified point. If there is no hotspot, the function returns 0.

puppet

Syntax

```
sprite(whichSprite).puppet
```

the puppet of sprite *whichSprite*

Description

Sprite property; determines whether the sprite channel specified by *whichSprite* is a puppet under Lingo control (TRUE) or not (FALSE, default).

- If a sprite channel is a puppet, any changes that Lingo makes to the channel's sprite properties remain in effect after the playback head leaves the sprite.
- If a sprite channel is not a puppet, any changes that Lingo makes to a sprite last for the life of the current sprite only.

While the playback head is in the same sprite, setting the sprite channel's puppet sprite property to FALSE resets the sprite's properties to those set in the Score.

Making the sprite channel a puppet lets you control many sprite properties, such as member, locH, and width, from Lingo after the playback head exits from the sprite.

Setting the puppet sprite property is equivalent to using the puppetSprite command. For example, the following statements are equivalent: set the puppet of sprite 1 to TRUE and puppetSprite 1, TRUE.

This property can be tested and set.

Example

This statement makes the sprite numbered *i* + 1 a puppet:

```
sprite(i + 1).puppet = TRUE
```

Example

This statement records whether sprite 5 is a puppet by assigning the value of the puppet sprite property to the variable. When sprite 5 is a puppet, isPuppet is set to TRUE. When sprite 5 is not a puppet, isPuppet is set to FALSE.

```
isPuppet = sprite(5).puppet
```

See also

puppetSprite

puppetPalette

Syntax

```
puppetPalette whichPalette {, speed} {,nFrames}
```

Description

Command; causes the palette channel to act as a puppet and lets Lingo override the palette setting in the palette channel of the Score and assign palettes to the movie.

The `puppetPalette` command sets the current palette to the palette cast member specified by `whichPalette`. If `whichPalette` evaluates to a string, it specifies the cast name of the palette. If `whichPalette` evaluates to an integer, it specifies the member number of the palette.

For best results, use the `puppetPalette` command before navigating to the frame on which the effect will occur so that Director can map to the desired palette before drawing the next frame.

You can fade in the palette by replacing `speed` with an integer from 1(slowest) to 60 (fastest). You can also fade in the palette over several frames by replacing `nFrames` with an integer for the number of frames.

A puppet palette remains in effect until you turn it off with the command `puppetPalette 0`. No subsequent palette changes in the Score are obeyed when the puppet palette is in effect.

Note: The browser controls the palette for the entire Web page. Thus, Shockwave and the Director player for Java always uses the browser's palette.

For the most reliable color when authoring a movie for playback as a Director player for Java, use the default palette for the authoring system.

Example This statement makes Rainbow the movie's palette:

```
puppetPalette "Rainbow"
```

Example This statement makes Grayscale the movie's palette. The transition to the Grayscale palette occurs over a time setting of 15 and between frames labeled Gray and Color.

```
puppetPalette "Grayscale", 15, label("Gray") - label("Color")
```

puppetSound

Syntax

```
puppetSound whichChannel, whichCastMember  
puppetSound whichCastMember  
puppetSound member whichCastMember  
puppetSound 0  
puppetSound whichChannel, 0
```

Description

Command; makes the sound channel a puppet, plays the sound cast member specified by *whichCastMember*, and lets Lingo override any sounds assigned in the Score's sound channels.

Specify a sound channel by replacing *whichChannel* with a channel number.

The sound starts playing after the playback head moves or the `updateStage` command is executed. Using 0 as the cast number argument stops the sound from playing. It also returns control of the sound channel to the Score.

Puppet sounds can be useful for playing a sound while a different movie is being loaded into memory.

The Director player for Java supports the following versions of the `puppetSound` command:

- `puppetSound whichChannel, whichCastMember, or puppetSound whichCastMember`—Plays a sound.
- `puppetSound 0 or puppetSound whichChannel, 0`—Stops a sound.

Example

This statement plays the sound Wind under control of Lingo:

```
puppetSound "Wind"
```

Example

This statement turns off the sound playing in channel 2:

```
puppetSound 2, 0
```

See also

`sound fadeln`, `sound fadeOut`, `sound playFile`, `sound stop`

puppetSprite

Syntax

`puppetSprite whichChannel, state`

Description

Command; determines whether the sprite channel specified by *whichSprite* is a puppet and under Lingo control (TRUE) or not a puppet and under the control of the Score (FALSE).

While the playback head is in the same sprite, turning off the sprite channel's puppetting using the command `puppetSprite whichSprite, FALSE` resets the sprite's properties to those in the Score.

The sprite channel's initial properties are whatever the channel's settings are when the `puppetSprite` command is executed. You can use Lingo to change sprite properties as follows:

- If a sprite channel is a puppet, any changes that Lingo makes to the channel's sprite properties remain in effect after the playback head exits the sprite.
- If a sprite channel is not a puppet, any changes that Lingo makes to a sprite last for the life of the current sprite only.

The channel must contain a sprite when you use the `puppetSprite` command.

Making the sprite channel a puppet lets you control many sprite properties—such as `memberNum`, `locH`, and `width`—from Lingo after the playback head exits the sprite.

Use the command `puppetSprite whichSprite, FALSE` to return control to the Score when you finish controlling a sprite channel from Lingo and to avoid unpredictable results that may occur when the playback head is in frames that aren't intended to be puppets.

Note: Version 6 of Director introduced autopuppeting, which made it unnecessary to explicitly puppet a sprite under most circumstances. Explicit control is still useful if you want to retain complete control over a channel's contents even after a sprite span has finished playing.

Example

This statement makes the sprite in channel 15 a puppet:

`puppetSprite 15, TRUE`

Example

This statement removes the puppet condition from the sprite in the channel numbered *i* + 1:

`puppetSprite i + 1, FALSE`

See also

`backColor`, `bottom`, `constraint`, `foreColor`, `height`, `ink`, `left`, `lineSize`, `locH`, `locV`, `memberNum`, `puppet`, `right`, `top`, `type` (sprite property), and `width`; `cursor` (command) and `puppetSprite`

puppetTempo

Syntax

`puppetTempo framesPerSecond`

Description

Command; causes the tempo channel to act as a puppet and sets the tempo to the number of frames specified by *framesPerSecond*. When the tempo channel is a puppet, Lingo can override the tempo setting in the Score and change the tempo assigned to the movie.

It's unnecessary to turn off the puppet tempo condition to make subsequent tempo changes in the Score take effect.

Note: Although it is theoretically possible to achieve frame rates up to 30,000 frames per second (fps) with the `puppetTempo` command, you could do this only with little animation and a very powerful machine.

Example

This statement sets the movie's tempo to 30 fps:

```
puppetTempo 30
```

Example

This statement increases the movie's old tempo by 10 fps:

```
puppetTempo oldTempo + 10
```

puppetTransition

Syntax

`puppetTransition member whichCastMember`

`puppetTransition whichTransition {,time} {,chunkSize} {,changeArea}`

Description

Command; performs the specified transition between the current frame and the next frame.

To use an Xtra transition cast member, use `puppetTransition member` followed by the cast member's name or number.

To use a built-in Director transition, replace *whichTransition* with a value in the following table. Replace *time* with the number of quarter seconds used to complete the transition. The minimum value is 0; the maximum is 120 (30 seconds). Replace *chunkSize* with the number of pixels in each chunk of the transition. The minimum value is 1; the maximum is 128. Smaller chunk sizes yield smoother transitions but are slower.

Code	Transition	Code	Transition
01	Wipe right	27	Random rows
02	Wipe left	28	Random columns
03	Wipe down	29	Cover down
04	Wipe up	30	Cover down, left

05	Center out, horizontal	31	Cover down, right
06	Edges in, horizontal	32	Cover left
07	Center out, vertical	33	Cover right
08	Edges in, vertical	34	Cover up
09	Center out, square	35	Cover up, left
10	Edges in, square	36	Cover up, right
11	Push left	37	Venetian blinds
12	Push right	38	Checkerboard
13	Push down	39	Strips on bottom, build left
14	Push up	40	Strips on bottom, build right
15	Reveal up	41	Strips on left, build down
16	Reveal up, right	42	Strips on left, build up
17	Reveal right	43	Strips on right, build down
18	Reveal down, right	44	Strips on right, build up
19	Reveal down	45	Strips on top, build left
20	Reveal down, left	46	Strips on top, build right
21	Reveal left	47	Zoom open
22	Reveal up, left	48	Zoom close
23	Dissolve, pixels fast*	49	Vertical blinds
24	Dissolve, boxy rectangles	50	Dissolve, bits fast*
25	Dissolve, boxy squares	51	Dissolve, pixels*
26	Dissolve, patterns	52	Dissolve, bits*

Transitions marked with an asterisk (*) do not work on monitors set to 32 bits.

There is no direct relationship between a low time value and a fast transition. The actual speed of the transition depends on the relation of *chunkSize* and *time*. For example, if *chunkSize* is 1 pixel, the transition takes longer no matter how low the time value, because the computer has to do a lot of work. To make transitions occur faster, use a larger chunk size, not a shorter time.

Replace *changeArea* with a value that determines whether the transition occurs only in the changing area (TRUE) or over the entire Stage (FALSE, default). The *changeArea* variable is an area within which sprites have changed.

Example	This statement performs a wipe right transition. Because no value is specified for <i>changeArea</i> , the transition occurs over the entire Stage, which is the default. puppetTransition 1
	This statement performs a wipe left transition that lasts 1 second, has a chunk size of 20, and occurs over the entire Stage: puppetTransition 2, 4, 20, FALSE

purgePriority

Syntax	member(<i>whichCastMember</i>).purgePriority the purgePriority of member <i>whichCastMember</i>
Description	Cast member property; specifies the purge priority of the cast member specified by <i>whichCastMember</i> . Cast members' purge priorities determine the priority that Director follows to choose which cast members to delete from memory when memory is full. The higher the purge priority, the more likely that the cast member will be deleted. The following purgePriority settings are available:
	<ul style="list-style-type: none"> • 0—Never • 1—Last • 2—Next • 3—Normal
	Normal, which is the default, lets Director purge cast members from memory at random. Next, Last, and Never allow some control over purging, but Last or Never may cause your movie to run out of memory if several cast members are set to these values. Setting purgePriority for cast members is useful for managing memory when the size of the movie's cast exceeds the available memory. As a general rule, you can minimize pauses while the movie loads cast members and reduce the number of times Director reloads a cast member by assigning a low purge priority to cast members that are used frequently in the course of the movie.

Example This statement sets the purge priority of cast member Background to 3, which makes it one of the first cast members to be purged when memory is needed:

```
member("Background").purgePriority = 3
```

put

Syntax put *expression*

Description Command; evaluates the expression specified by *expression* and displays the result in the Message window. This command can be used as a debugging tool by tracking the values of variables as a movie plays.

The Director player for Java displays the message from the put command in the browser's Java console window. Access to the console window depends on the browser.

Example This statement displays the time in the Message window:

```
put the time  
-- "9:10 AM"
```

Example This statement displays the value assigned to the variable *bid* in the Message window:

```
put bid  
-- "Johnson"
```

See also put...after, put...before, put...into

put...after

Syntax put *expression* after *chunkExpression*

Description Command; evaluates a Lingo expression, converts the value to a string, and inserts the resulting string after a specified chunk in a container, without replacing the container's contents. (If *chunkExpression* specifies a nonexistent target chunk, the string value is inserted as appropriate into the container.)

Chunk expressions refer to any character, word, item, or line in any container. Containers include field cast members; text cast members; variables that hold strings; and specified characters, words, items, lines, and ranges within containers.

Example This statement adds the string "fox dog cat" after the contents of the field cast member Animal List:

```
put "fox dog cat" after member "Animal List"
```

The same can be accomplished using this statement:

```
put "fox dog cat" after member("Animal List").line[1]
```

See also char...of, item...of, line...of, paragraph, word...of; put...before, put...into

put...before

Syntax

`put expression before chunkExpression`

Description

Command; evaluates a Lingo expression, converts the value to a string, and inserts the resulting string before a specified chunk in a container, without replacing the container's contents. (If *chunkExpression* specifies a nonexistent target chunk, the string value is inserted as appropriate into the container.)

Chunk expressions refer to any character, word, item, or line in any container. Containers include field cast members; text cast members; variables that hold strings; and specified characters, words, items, lines, and ranges in containers.

Example

This statement sets the variable `animalList` to the string “fox dog cat” and then inserts the word `elk` before the second word of the list:

```
put "fox dog cat" into animalList  
put "elk " before word 2 of animalList
```

The result is the string “fox elk dog cat”.

The same can be accomplished using this syntax:

```
put "fox dog cat" into animalList  
put "elk " before animalList.word[2]
```

See also

`char...of, item...of, line...of, paragraph, word...of; put...after, put...into`

put...into

Syntax

`put expression into chunkExpression`

Description

Command; evaluates a Lingo expression, converts the value to a string, and uses the resulting string to replace a specified chunk in a container. (If *chunkExpression* specifies a nonexistent target chunk, the string value is inserted as appropriate into the container.)

Chunk expressions refer to any character, word, item, or line in any container. Containers include field cast members; text cast members; variables that hold strings; and specified characters, words, items, lines, and ranges in containers.

When a movie plays back as an applet, the `put...into` command replaces all text within a container, not chunks of text.

To assign values to variables, use the `set` command.

Example

This statement changes the second line of the field cast member `Review Comments` to “Reviewed by Agnes Gooch”:

```
put "Reviewed by Agnes Gooch" into line 2 of member "Review Comments"
```

The same can be accomplished with a text cast member using this syntax:

```
put "Reviewed by Agnes Gooch" into member("Review Comments").line[2]
```

See also

`char...of, item...of, line...of, paragraph, word...of, put...after, put...before, set...to, set...=`

qtRegisterAccessKey

Syntax

```
qtRegisterAccessKey(categoryString, keyString)
```

Description

Command; allows registration of a key for encrypted QuickTime media.

The key is an application-level key, not a system-level key. After the application unregisters the key or shuts down, the media will no longer be accessible.

Note: For security reasons, there is no way to display a listing of all registered keys.

See also

[qtUnRegisterAccessKey](#)

qtUnRegisterAccessKey

Syntax

```
qtUnRegisterAccessKey(categoryString, keyString)
```

Description

Command; allows the key for encrypted QuickTime media to be unregistered.

The key is an application-level key, not a system-level key. After the application unregisters the key, only movies encrypted with this key continue to play. Other media will no longer be accessible.

See also

[qtRegisterAccessKey](#)

quad

Syntax

```
sprite(whichSpriteNumber).quad
```

Description

Sprite property; contains a list of four points, which are floating point values that describe the corner points of a sprite on the Stage. The points are organized in the following order: upper left, upper right, lower right, and lower left.

The points themselves can be manipulated to create perspective and other image distortions.

After you manipulate the quad of a sprite, you can reset it to the Score values by turning off the puppet of the sprite with `puppetSprite whichSpriteNumber, FALSE`. When the quad of a sprite is disabled, you cannot rotate or skew the sprite.

Example

This statement displays a typical list describing a sprite:

```
put sprite(1).quad  
-- [point(153.0000, 127.0000), point(231.0000, 127.0000), \  
point(231.0000, 242.0000), point(153.0000, 242.0000)]
```

Example When modifying this sprite property, be aware that you must reset the list of points after changing any of the values. This is because when you set a variable to the value of a property, you are placing a copy of the list, not the list itself, in the variable. To effect a change, use syntax like this:

```
-- Get the current property contents  
currQuadList = sprite(5).quad  
-- Add 50 pixels to the horizontal and vertical positions of the first point in the list  
currQuadList[1] = currQuadList[1] + point(50, 50)  
-- Reset the actual property to the newly computed position  
sprite(5).quad = currQuadList
```

See also rotation, skew

quality

Syntax `sprite(whichFlashSprite).quality`

the quality of sprite *whichFlashSprite*

`member(whichFlashMember).quality`

the quality of member *whichFlashMember*

Description Flash cast member and sprite property; controls whether Director uses anti-aliasing to render a Flash movie sprite, producing high-quality rendering but possibly slower movie playback. The quality property can have these values:

- `#autoHigh`—Director starts by rendering the sprite with anti-aliasing. If the actual frame rate falls below the movie's specified frame rate, Director turns off anti-aliasing. This setting gives precedence to playback speed over visual quality.
- `#autoLow`—Director starts by rendering the movie without anti-aliasing. If the Flash player determines that the computer processor can handle it, anti-aliasing is turned on. This setting gives precedence to visual quality whenever possible.
- `#high` (default)—The movie always plays with anti-aliasing.
- `#low`—The movie always plays without anti-aliasing.

The quality property can be tested and set.

Example This sprite script checks the color depth of the computer on which the movie is playing. If the color depth is set to 8 bits or less (256 colors), the script sets the quality of the sprite in channel 5 to `#low`.

```
on beginSprite me  
    if the colorDepth <= 8 then  
        sprite(1).quality = #low  
    end if  
end
```

queue()

Syntax

```
sound(channelNum).queue(member(whichMember))  
  
sound(channelNum).queue([#member: member(whichmember), {#startTime: milliseconds,  
#endTime: milliseconds, #loopCount: numberOfLoops, #loopStartTime: milliseconds,  
#loopEndTime: milliseconds, #preloadTime: milliseconds}])  
  
queue(sound(channelNum), member(whichMember))  
  
queue(soundObject, [#member: member(whichmember), {#startTime: milliseconds,  
#endTime: milliseconds, #loopCount: numberOfLoops, #loopStartTime: milliseconds,  
#loopEndTime: milliseconds, #preloadTime: milliseconds}])
```

Description

Function; adds the given sound cast member to the queue of sound channel *channelNum*.

Once a sound has been queued, it can be played immediately with the `play()` command. This is because Director preloads a certain amount of each sound that is queued, preventing any delay between the `play()` command and the start of playback. The default amount of sound that is preloaded is 1500 milliseconds. This parameter can be modified by passing a property list containing one or more parameters with the `queue()` command.

You can specify these properties may be specified with the `queue()` command:

Property	Description
#member	The sound cast member to queue. This property must be provided; all others are optional.
#startTime	The time within the sound at which playback begins, in milliseconds. The default is the beginning of the sound. See <code>startTime</code> .
#endTime	The time within the sound at which playback ends, in milliseconds. The default is the end of the sound. See <code>endTime</code> .
#loopCount	The number of times to play a loop defined with <code>#loopStartTime</code> and <code>#loopEndTime</code> . The default is 1. See <code>loopCount</code> .
#loopStartTime	The time within the sound to begin a loop, in milliseconds. See <code>loopStartTime</code> .
#loopEndTime	The time within the sound to end a loop, in milliseconds. See <code>loopEndTime</code> .
#preloadTime	The amount of the sound to buffer before playback, in milliseconds. See <code>preloadTime</code> .

These parameters can also be passed with the `setPlayList()` command.

To see an example of queue() used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This handler queues and plays two sounds. The first sound, cast member Chimes, is played in its entirety. The second sound, cast member introMusic, is played starting at its 3-second point, with a loop repeated 5 times from the 8-second point to the 8.9 second point, and stopping at the 10-second point.

```
on playMusic
    sound(2).queue([#member: member("Chimes")])
    sound(2).queue([#member: member("introMusic"), #startTime: 3000 \
        #endTime: 10000, #loopCount: 5, #loopStartTime: 8000, #loopEndTime: 8900])
    sound(2).play()
end
```

See also startTime, endTime, loopCount, loopStartTime, loopEndTime, preLoadTime, setPlaylist(), play() (sound)

quickTimeVersion()

Syntax quickTimeVersion()

Description Function; returns a floating-point value that identifies the current installed version of QuickTime and replaces the current QuickTimePresent function.

In Windows, if multiple versions of QuickTime 3.0 or later are installed, quickTimeVersion() returns the latest version number. If a version before QuickTime 3.0 is installed, quickTimeVersion() returns version number 2.1.2 regardless of the version installed.

Example This statement uses quickTimeVersion() to display in the Message window the version of QuickTime that is currently installed:

```
put quickTimeVersion()
```

quit

Syntax quit

Description Command; exits from Director or a projector to the Windows desktop or Macintosh Finder.

Example This statement tells the computer to exit to the Windows desktop or Macintosh Finder when the user presses Control+Q in Windows or Command+Q on the Macintosh:

```
if the key = "q" and the commandDown then quit
```

See also restart, shutDown

QUOTE

Syntax	QUOTE
Description	Constant; represents the quotation mark character and refers to the literal quotation mark character in a string, because the quotation mark character itself is used by Lingo scripts to delimit strings.
Example	<p>This statement inserts quotation mark characters in a string:</p> <pre>put "Can you spell" && QUOTE & "Macromedia" & QUOTE & "?"</pre> <p>The result is a set of quotation marks around the word <i>Macromedia</i>:</p> <p>Can you spell "Macromedia"?</p>

ramNeeded()

Syntax	ramNeeded (<i>firstFrame</i> , <i>lastFrame</i>)
Description	Function; determines the memory needed, in bytes, to display a range of frames. For example, you can test the size of frames containing 32-bit artwork: if ramNeeded() is larger than freeBytes(), then go to frames containing 8-bit artwork and divide by 1024 to convert bytes to kilobytes (K).
Example	<p>This statement sets the variable <i>frameSize</i> to the number of bytes needed to display frames 100 to 125 of the movie:</p> <pre>put ramNeeded (100, 125) into frameSize</pre>
Example	<p>This statement determines whether the memory needed to display frames 100 to 125 is more than the available memory, and if it is, branches to the section using cast members that have lower color depth:</p> <pre>if ramNeeded (100, 125) > the freeBytes then play frame "8-bit"</pre>
See also	freeBytes(), size

random()

Syntax

```
random(integerExpression)
```

Description

Function; returns a random integer in the range 1 to the value specified by *integerExpression*. This function can be used to vary values in a movie, such as to vary the path through a game, assign random numbers, or change the color or position of sprites.

To start a set of possible random numbers with a number other than 1, subtract the appropriate amount from the random() function. For example, the expression random(n + 1) - 1 uses a range from 0 to the number n.

Example

This statement assigns random values to the variable diceRoll:

```
set diceRoll = random(6) + random(6)
```

This statement randomly changes the foreground color of sprite 10:

```
sprite(10).forecolor = random(256) - 1
```

Example

This handler randomly chooses which of two movie segments to play:

```
on SelectScene
    if random(2) = 2 then
        play frame "11a"
    else
        play frame "11-b"
    end if
end
```

Example

The following statements produce results in a specific range.

This statement produces a random multiple of 5 in the range 5 to 100:

```
theScore = 5 * random(20)
```

This statement produces a random multiple of 5 in the range 0 to 100:

```
theScore = 5 * (random(21) - 1)
```

This statement generates integers between -10 and +10:

```
dirH = random(21) - 11
```

This statement produces a random two-point decimal value:

```
the floatPrecision = 2
theCents = random(100)/100.0 - .01
```

randomSeed

Syntax	the randomSeed
Description	System property; specifies the seed value used for generating random numbers accessed through the random() function.
	Using the same seed produces the same sequence of random numbers. This property can be useful for debugging during development. Using the ticks property is an easy way to produce a unique random seed since the ticks value is highly unlikely to be duplicated on subsequent uses.
	This property can be tested and set.
Example	This statement displays the random seed number in the Message window: <pre>put the randomSeed</pre>
See also	random(), ticks

rawNew()

Syntax	<pre>parentScript.rawNew()</pre> <pre>rawNew(parentScript)</pre>
Description	Function; creates a child object from a parent script without calling its on new handler. This allows a movie to create child objects without initializing the properties of those child objects. This is particularly useful when you want to create large numbers of child objects for later use. To initialize the properties of one of these raw child objects, call its on new handler.
Example	This statement creates a child object called RedCar from the parent script CarParentScript without initializing its properties: <pre>RedCar = script("CarParentScript").rawNew()</pre>
Example	This statement initializes the properties of the child object RedCar: <pre>RedCar.new()</pre> <pre>new(), script</pre>

recordFont

Syntax

```
recordFont(whichCastMember, font {[,.face]} {[,.bitmapSizes]} {,characterSubset} {,  
userFontName})
```

Description

Command; embeds a TrueType or Type 1 font as a cast member. Once embedded, these fonts are available to the author just like other fonts installed in the system.

You must create an empty font cast member with the new() command before using recordFont.

- *font*—Name of original font to be recorded.
- *face*—List of symbols indicating the face of the original font; possible values are #plain, #bold, #italic. If you do not provide a value for this argument, #plain is used.
- *bitmapSizes*—List of integers specifying the sizes for which bitmaps are to be recorded. This argument can be empty. If you omit this argument, no bitmaps are generated. These bitmaps typically look better at smaller point sizes (below 14 points) but take up more memory.
- *characterSubset*—String of characters to be encoded. Only the specified characters will be available in the font. If this argument is, all characters are encoded. If only certain characters are encoded but an unencoded character is used, that character is displayed as an empty box.
- *userFontName*—A string to use as the name of the newly recorded font cast member.

The command creates a Shock Font in *whichCastMember* using the font named in the *font* argument. The value returned from the command reports whether the operation was successful. Zero indicates success.

Example

This statement creates a simple Shock Font using only the two arguments for the cast member and the font to record:

```
myNewFontMember = new(#font)  
recordFont(myNewFontMember, "Lunar Lander")
```

Example

This statement specifies the bitmap sizes to be generated and the characters for which the font data should be created:

```
myNewFontMember = new(#font)  
recordfont(mynewmember,"lunar lander", [], [14, 18, 45], "Lunar Lander Game High \\\nScore First Last Name")
```

Note: Since recordFont resynthesizes the font data rather than using it directly, there are no legal restrictions on Shock Font distribution.

See also

new()

rect()

Syntax

rect(*left, top, right, bottom*)

rect(*point1, point2*)

Description

Function and data type; defines a rectangle.

The rect(*left, top, right, bottom*) format defines a rectangle whose sides are specified by *left*, *top*, *right*, and *bottom*. The *left* and *right* values specify numbers of pixels from the left edge of the Stage. The *top* and *bottom* values specify numbers of pixels from the top of the Stage.

The rect(*point1, point2*) format defines a rectangle that encloses the points specified by *point1* and *point2*.

You can refer to rectangle components by list syntax or property syntax. For example, the following two phrases are equivalent:

```
targetWidth = targetRect.right - targetRect.left  
targetWidth = targetRect[3] - targetRect[1]
```

You can perform arithmetic operations on rectangles. If you add a single value to a rectangle, Lingo adds it to each element in the rectangle.

To see an example of rect() used in a completed movie, see the Imaging movie in the LearningLingo Examples folder inside the Director application folder.

Example

This statement sets the variable newArea to a rectangle whose left side is at 100, top is at 150, right side is at 300, and bottom is at 400 pixels:

```
set newArea = rect(100, 150, 300, 400)
```

Example

This statement sets the variable newArea to the rectangle defined by the points firstPoint and secondPoint. The coordinates of firstPoint are (100, 150); the coordinates of secondPoint are (300, 400). Note that this statement creates the same rectangle as the one created in the previous example:

```
put rect(firstPoint, secondPoint)
```

Example

These statements add and subtract values for rectangles:

```
put rect(0,0,100,100) + rect(30, 55, 120, 95)  
-- rect(30, 55, 220, 195)  
put rect(0,0,100,100) - rect(30, 55, 120, 95)  
-- rect(-30, -55, -20, 5)
```

Example

This statement adds 80 to each coordinate in a rectangle:

```
put rect(60, 40, 120, 200) + 80  
-- rect(140, 120, 200, 280)
```

Example

This statement divides each coordinate in a rectangle by 3:

```
put rect(60, 40, 120, 200) / 3  
-- rect(20, 13, 40, 66)
```

See also

point(), quad

rect (image)

Syntax *imageObject.rect*

Description Read-only property; returns a rectangle describing the size of the given image object. The coordinates are given relative to the top left corner of the image. The left and top values of the rectangle are therefore always 0, and the right and bottom values are the width and height of the cast member.

Example This statement returns the rectangle of the 300 x 400 pixel member Sunrise in the message window:

```
put member("Sunrise").image.rect  
-- rect(0, 0, 300, 400)
```

Example This Lingo looks at the first 50 cast members and displays the rectangle and name of each cast member that is a bitmap:

```
on showAllRects  
repeat with x = 1 to 50  
  if member(x).type = #bitmap then  
    put member(x).image.rect && "-" && member(x).name  
  end if  
end repeat  
end
```

See also [height](#), [width](#)

rect (member)

Syntax *member(whichCastMember).rect*

the rect of member *whichCastMember*

Description Cast member property; specifies the left, top, right, and bottom coordinates, returned as a rectangle, for the rectangle of any graphic cast member, such as a bitmap, shape, movie, or digital video.

For a bitmap, the rect member property is measured from the upper left corner of the bitmap, instead of from the upper left corner of the easel in the Paint window.

For an Xtra cast member, the rect member property is a rectangle whose upper left corner is at (0,0).

The Director player for Java can't set the rect member property.

This property can be tested. It can be set for field cast members only.

Example This statement displays the coordinates of bitmap cast member 20:

```
put member(20).rect
```

Example This statement sets the coordinates of bitmap cast member Banner:

```
member("Banner").rect = rect(100, 150, 300, 400)
```

See also [rect\(\)](#), [rect \(sprite\)](#)

rect (sprite)

Syntax	sprite <i>whichSprite</i> .rect
	the rect of sprite <i>whichSprite</i>
Description	Sprite property; specifies the left, top, right, and bottom coordinates, as a rectangle, for the rectangle of any graphic sprite such as a bitmap, shape, movie, or digital video.
	This property can be tested and set.
Example	This statement displays the coordinates of bitmap sprite 20: <code>put sprite(20).rect</code>

rect (window)

Syntax	window <i>whichWindow</i> .rect
	the rect of window <i>whichWindow</i>
Description	Window property; specifies the left, top, right, and bottom coordinates, as a rectangle, of the window specified by <i>whichWindow</i> .
	If the size of the rectangle specified is less than that of the Stage where the movie was created, the movie is cropped in the window, not resized.
	To pan or scale the movie playing in the window, set the drawRect or sourceRect property of the window.
	This property can be tested and set.
Example	This statement displays the coordinates of the window Control Panel: <code>put window("Control Panel").rect</code>
See also	<code>drawRect</code> , <code>sourceRect</code>

ref

Syntax *chunkExpression.ref*

Description Text chunk expression property; this provides a convenient way to refer to a chunk expression within a text cast member.

Example For example, without references you would need statements like these:

```
member(whichTextMember).line[whichLine].word[firstWord..lastWord].font = "Palatino"  
member(whichTextMember).line[whichLine].word[firstWord..lastWord].fontSize = 36  
member(whichTextMember).line[whichLine].word[firstWord..lastWord].fontStyle = [#bold]
```

But with a ref property you can refer to the same chunk as follows:

```
myRef = member(whichTextMember).line[whichLine].word[firstWord..lastWord].ref
```

The variable myRef is now shorthand for the entire chunk expression. This allows something like the following:

```
put myRef.font  
-- "Palatino"
```

Or you can set a property of the chunk as follows:

```
myRef.fontSize = 18  
myRef.fontStyle = [#italic]
```

You can get access to the string referred to by the reference using the text property of the reference:

```
put myRef.text
```

This would result in the actual string data, and not information about the string.

regPoint

Syntax *member(whichCastMember).regPoint*

the regPoint of member *whichCastMember*

Description Cast member property; specifies the registration point of a cast member. The registration point is listed as the horizontal and vertical coordinates of a point in the form point (*horizontal, vertical*). Nonvisual members such as sounds do not have a useful regPoint property.

You can use the regPoint property to animate individual graphics in a film loop, changing the film loop's position in relation to other objects on the Stage.

You can also use regPoint to adjust the position of a mask being used on a sprite.

When a Flash movie cast member is first inserted into the cast, its registration point is its center and its centerRegPoint property is set to TRUE. If you subsequently use the regPoint property to reposition the registration point, the centerRegPoint property is automatically set to FALSE.

This property can be tested and set.

Example This statement displays the registration point of the bitmap cast member Desk in the Message window:

```
put member("Desk").regPoint
```

Example This statement changes the registration point of the bitmap cast member Desk to the values in the list:

```
member("Desk").regPoint = point(300, 400)
```

See also centerRegPoint, mask

regPointVertex

Syntax *vectorMember*.regPointVertex

the regPointVertex of *vectorMember*

Description Cast member property; indicates whether a vertex of *vectorCastMember* is used as the registration point for that cast member. If the value is zero, the registration point is determined normally, using the centerRegPoint and regPoint properties. If the value is nonzero, it indicates the position in the vertexList of the vertex being used as the registration point. The centerRegPoint is set to FALSE and the regPoint is set to the location of that vertex.

Example This statement makes the registration point for the vector shape cast member Squiggle correspond to the the location of the third vertex:

```
member("squiggle").regPointVertex=3
```

centerRegPoint, regPoint

relative

See @ (pathname)

repeat while

Syntax

```
repeat while testCondition
    statement(s)
end repeat
```

Description

Keyword; repeatedly executes *statement(s)* so long as the condition specified by *testCondition* is TRUE. This structure can be used in Lingo that continues to read strings until the end of a file is reached, checks items until the end of a list is reached, or repeatedly performs an action until the user presses or releases the mouse button.

While in a repeat loop, Lingo ignores other events. To check the current key in a repeat loop, use the `keyPressed` property.

Only one handler can run at a time. If Lingo stays in a repeat loop for a long time, other events stack up waiting to be evaluated. Therefore, repeat loops are best used for short, fast operations or when users are idle.

If you need to process something for several seconds or more, evaluate the function in a loop with some type of counter or test to track progress.

The Director player for Java doesn't detect mouse movements, update properties that indicate the mouse's position, or update the status of mouse button presses when Lingo is in a repeat loop.

If the stop condition is never reached or there is no exit from the repeat loop, you can force Director to stop by using Control+Alt+period (Windows) or Command+period (Macintosh).

Example

This handler starts the timer counting, resets the timer to 0, and then has the timer count up to 60 ticks:

```
on countTime
    startTimer
    repeat while the timer < 60
        -- waiting for time
    end repeat
end countTime
```

See also

`exit`, `exit repeat`, `repeat with`, `keyPressed()`

repeat with

Syntax

```
repeat with counter = start to finish
    statement(s)
end
```

Description

Keyword; executes the Lingo specified by *statement(s)* the number of times specified by *counter*. The value of *counter* is the difference between the value specified by *start* and the value specified by *finish*. The counter is incremented by 1 each time Lingo cycles through the repeat loop.

The repeat with structure is useful for repeatedly applying the same effect to a series of sprites or for calculating a series of numbers to some exponent.

While in a repeat loop, Lingo ignores other events. To check the current key in a repeat loop, use the `keyPressed` property.

Only one handler can run at a time. If Lingo stays in a repeat loop for a long time, other events stack up waiting to be evaluated. Therefore, repeat loops are best used for short, fast operations or when users are idle.

If you need to process something for several seconds or more, evaluate the function in a loop with some type of counter or test to track progress.

If the stop condition is never reached or there is no exit from the repeat loop, you can force Director to stop by using Control+Alt+period (Windows) or Command+period (Macintosh).

The Director player for Java doesn't detect mouse movements, update properties that indicate the mouse's position, or update the status of mouse button presses when Lingo is in a repeat loop.

Example

The following handler turns sprites 1 through 30 into puppets:

```
on puppetize
    repeat with channel = 1 to 30
        puppetSprite channel, TRUE
    end repeat
end puppetize
```

See also

[exit](#), [exit repeat](#), [repeat while](#)

repeat with...down to

Syntax

repeat with *variable* = *startValue* down to *endValue*

Description

Keyword; counts down by increments of 1 from *startValue* to *endValue*.

Only one handler can run at a time. If Lingo stays in a repeat loop for a long time, other events stack up waiting to be evaluated. Therefore, repeat loops are best used for short, fast operations or when you know the user won't be doing other things.

While in a repeat loop, Lingo ignores other events. To check the current key in a repeat loop, use the `keyPressed` property.

If you need to process something for several seconds or more, evaluate the function in a loop with some type of counter or test to track progress.

If the stop condition is never reached or there is no exit from the repeat loop, you can force Director to stop by using Control+Alt+period (Windows) or Command+period (Macintosh).

The Director player for Java doesn't detect mouse movements, update properties that indicate the mouse's position, or update the status of mouse button presses when Lingo is in a repeat loop.

Example

This handler contains a repeat loop that counts down from 20 to 15:

```
on CountDown
    repeat with i = 20 down to 15
        sprite(6).memberNum = 10 + i
        updateStage
    end repeat
end
```

repeat with...in list

Syntax

repeat with *variable* in *someList*

Description

Keyword; assigns successive values from the specified list to the variable.

While in a repeat loop, Lingo ignores other events except keypresses. To check the current key in a repeat loop, use the `keyPressed` property.

Only one handler can run at a time. If Lingo stays in a repeat loop for a long time, other events stack up waiting to be evaluated. Therefore, repeat loops are best used for short, fast operations or when users are idle.

If you need to process something for several seconds or more, evaluate the function in a loop with some type of counter or test to track progress.

If the stop condition is never reached or there is no exit from the repeat loop, you can force Director to stop by using Control+Alt+period (Windows) or Command+period (Macintosh).

The Director player for Java doesn't detect mouse movements, update properties that indicate the mouse's position, or update the status of mouse button presses when Lingo is in a repeat loop.

Example This statement displays four values in the Message window:

```
repeat with i in [1, 2, 3, 4]
    put i
end repeat
```

on resizeWindow

Syntax on resizeWindow
 statement(s)
end

Description System message and event handler; contains statements that run when a movie is running as a movie in a window (MIAW) and the user resizes the window by dragging the window's resize box or one of its edges.

An on resizeWindow event handler is a good place to put Lingo related to the window's dimensions, such as Lingo that positions sprites or crops digital video.

Example This handler moves sprite 3 to the coordinates stored in the variable centerPlace when the window that the movie is playing in is resized:

```
on resizeWindow centerPlace
    sprite(3).loc = centerPlace
end
```

See also drawRect, sourceRect

restart

Syntax restart

Description Command; closes all open applications and restarts the computer.

Example This statement restarts the computer when the user presses Command+R (Macintosh) or Control+R (Windows):

```
if the key = "r" and the commandDown then restart
```

See also quit, shutDown

result

Syntax the result

Description Function; displays the value of the return expression from the last handler executed.

The result function is useful for obtaining values from movies that are playing in windows and tracking Lingo's progress by displaying results of handlers in the Message window as the movie plays.

This function has no effect in the Director player for Java.

To return a result from a handler, assign the result to a variable and then check the variable's value. Use a statement such as set myVariable = *function()*, where *function()* is the name of a specific function.

Example The following handler returns a random roll for two dice:

```
on diceRoll  
    return random(6) + random(6)  
end
```

Example The two statements

```
diceRoll  
roll = the result
```

are equivalent to this statement:

```
set roll = diceRoll()
```

Note that set roll = diceRoll would not call the handler because there are no parentheses following diceRoll; diceRoll here is considered a variable reference.

See also return (keyword)

resume sprite

Syntax sprite(*whichGIFSpriteNumber*).resume()

```
resume(sprite whichGIFSpriteNumber)
```

Description Animated GIF command; causes the sprite to resume playing from the frame after the current frame if it's been paused. This command has no effect if the animated GIF sprite has not been paused.

See also pause sprite, rewind sprite

RETURN (constant)

Syntax	RETURN
Description	Constant; represents a carriage return.
Example	This statement causes a paused movie to continue when the user presses the carriage return: <pre>if (the key = RETURN) then go to the frame + 1</pre>
Example	This statement uses the RETURN character constant to insert a carriage return between two lines in an alert message: <pre>alert "Last line in the file." & RETURN & "Click OK to exit."</pre>
Example	In Windows, it is standard practice to place an additional line-feed character at the end of each line. This statement creates a two-character string named CRLF that provides the additional line feed: <pre>CRLF = RETURN & numToChar(10)</pre>

return (keyword)

Syntax	return <i>expression</i>
Description	Keyword; returns the value of <i>expression</i> and exits from the handler. The <i>expression</i> argument can be any Lingo value. When calling a handler that serves as a user-defined function and has a return value, you must use parentheses around the argument lists, even if there are no arguments, as in the diceRoll function handler discussed under the entry for the result function. The function of the return keyword is similar to that of the exit command, except that return also returns a value to whatever called the handler. The return command in a handler immediately exits from that handler, but it can return a value to the Lingo that called it. The use of return in object-oriented scripting can be difficult to understand. It's easier to start by using return to create functions and exit handlers. Later, you will see that the return me line in an on new handler gives you a way to pass back a reference to an object that was created so it can be assigned to a variable name. The return keyword isn't the same as the character constant RETURN, which indicates a carriage return. The function depends on the context. To retrieve a returned value, use parentheses after the handler name in the calling statement to indicate that the named handler is a function. To see an example of return (keyword) used in a completed movie, see the Parent Scripts movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This handler returns a random multiple of 5 between 5 and 100:

```
on getRandomScore
    theScore = 5 * random(20)
    return theScore
end getRandomScore
```

Call this handler with a statement similar to the following:

```
set thisScore to GetRandomScore()
```

In this example, the variable `thisScore` is assigned the return value from the function `GetRandomScore`. A parent script performs the same function: by returning the object reference, the variable name in the calling code provides a handle for subsequent references to that object.

See also [result](#), [RETURN \(constant\)](#)

rewind()

Syntax `sound(channelNum).rewind()`

```
rewind(sound(channelNum))
```

Description Function; interrupts the playback of the current sound in sound channel `channelNum` and restarts it at its `startTime`. If the sound is paused, it remains paused, with the `currentTime` set to the `startTime`.

Example This restarts playback of the sound cast member playing in sound channel 1 from the beginning.

```
sound(1).rewind()
```

See also [pause\(\) \(sound playback\)](#), [play\(\) \(sound\)](#), [playNext\(\)](#), [queue\(\)](#), [stop\(\) \(sound\)](#)

rewind sprite

Syntax `sprite(whichFlashOrGIFAnimSprite).rewind()`

```
rewind sprite whichFlashOrGIFAnimSprite
```

Description Command; returns a Flash or animated GIF movie sprite to frame 1 when the sprite is stopped or when it is playing.

Example This frame script checks whether the Flash movie sprite in the sprite the behavior was placed in is playing and, if so, continues to loop in the current frame. When the movie is finished, the sprite rewinds the movie (so the first frame of the movie appears on the Stage) and lets the playback head continue to the next frame.

```
property spriteNum  
  
on exitFrame  
    if sprite(spriteNum).playing then  
        go the frame  
    else  
        sprite(spriteNum).rewind()  
        updateStage  
    end if  
end
```

right

Syntax

`sprite(whichSprite).right`

the right of sprite *whichSprite*

Description Sprite property; indicates the distance, in pixels, of the specified sprite's right edge from the left edge of the Stage.

When a movie plays back as an applet, this property's value is relative to the upper left corner of the applet.

Sprite coordinates are expressed relative to the upper left corner of the Stage.

This property can be tested and set.

Example This statement calls the handler `offRightEdge` when the right edge of sprite 3 is past the right edge of the Stage:

```
if sprite(3).right > (the stageRight - the stageLeft) then offRightEdge
```

See also `bottom`, `height`, `left`, `locH`, `locV`, `top`, `width`

rightIndent

Syntax

`chunkExpression.rightIndent`

Description Text cast member property; contains the offset distance, in pixels, of the right margin of *chunkExpression* from the right side of the text cast member.

The value is an integer greater than or equal to 0.

This property can be tested and set.

See also `firstIndent`, `leftIndent`

on rightMouseDown (event handler)

Syntax

```
on rightMouseDown  
    statement(s)  
end
```

Description System message and event handler; in Windows, specifies statements that run when the right mouse button is pressed. On Macintosh computers, the statements run when the mouse button and Control key are pressed simultaneously and the emulateMultiButtonMouse property is set to TRUE; if this property is set to FALSE, this event handler has no effect on the Macintosh.

Example This handler opens the window Help when the user clicks the right mouse button in Windows:

```
on rightMouseDown  
    window("Help").open()  
end
```

rightMouseDown (system property)

Syntax

```
the rightMouseDown
```

Description System property; indicates whether the right mouse button (Windows) or the mouse button and Control key (Macintosh) are being pressed (TRUE) or not (FALSE).

On the Macintosh, rightMouseDown is TRUE only if the emulateMultiButtonMouse property is TRUE.

Example This statement checks whether the right mouse button in Windows is being pressed and plays the sound Oops in sound channel 2 if it is:

```
if the rightMouseDown then puppetSound 2, "Oops"
```

See also emulateMultiButtonMouse

on rightMouseUp (event handler)

Syntax

```
on rightMouseUp  
    statement(s)  
end
```

Description System message and event handler; in Windows, specifies statements that run when the right mouse button is released. On Macintosh computers, the statements run if the mouse button is released while the Control key is pressed and the emulateMultiButtonMouse property is set to TRUE; if this property is set to FALSE, this event handler has no effect on the Macintosh.

Example This handler opens the Help window when the user releases the right mouse button in Windows:

```
on rightMouseUp  
    window("Help").open()  
end
```

rightMouseUp (system property)

Syntax

```
the rightMouseUp
```

Description System property; indicates whether the right mouse button (Windows) or the mouse button and Control key (Macintosh) are currently not being pressed (TRUE) or are currently being pressed (FALSE).

On the Macintosh, rightMouseUp is TRUE only if the emulateMultiButtonMouse property is TRUE.

Example This statement checks whether the right mouse button in Windows is released and plays the sound Click Me if it is:

```
if the rightMouseUp then puppetSound 2, "Click Me"
```

See also emulateMultiButtonMouse

rollOver()

Syntax

rollOver(*whichSprite*)

the rollOver

Description

Function; indicates whether the cursor is currently over the bounding rectangle of the sprite specified by *whichSprite* (TRUE or 1) or not (FALSE or 0).

This function has two possible syntax formats:

- When rollOver isn't preceded by the, include parentheses.
- When rollOver is preceded by the, don't include parentheses.

The rollOver function is typically used in frame scripts and is useful for creating handlers that perform an action when the user places the cursor over a specific sprite. It can also simulate additional sprite channels by splitting the Stage into regions that each send the playback head to a different frame that subdivides the region for the available sprite channels.

If the user continues to roll the mouse, the value of rollOver can change while Lingo is running a handler. You can make sure that a handler uses a consistent rollover value by assigning rollOver to a variable when the handler starts.

When the cursor is over the location of a sprite that no longer appears in the Score in the current section, rollOver still occurs and reports the sprite as being there.

Avoid this problem by not performing rollovers over these locations or by moving the sprite above the menu bar before removing it.

Example

This statement changes the content of the field cast member Message to "This is the place." when the cursor is over sprite 6:

```
if rollover(6) then member("Message").text = "This is the place."
```

Example

This handler sends the playback head to different frames when the cursor is over certain sprites on the Stage. It first assigns the rollOver value to a variable. This lets the handler use the rollOver value that was in effect when the rollover started, regardless of whether the user continues to move the mouse.

```
on exitFrame
  set currentSprite = the rollover
  case currentSprite of
    1: go to frame "Left"
    2: go to frame "Middle"
    3: go to frame "Right"
  end case
end exitFrame
```

See also

mouseMember

romanLingo

Syntax	the romanLingo
Description	System property; specifies whether Lingo uses a single-byte (TRUE) or double-byte interpreter (FALSE). The Lingo interpreter is faster with single-byte character sets. Some versions of Macintosh system software—Japanese, for example—use a double-byte character set. U.S. system software uses a single-byte character set. Normally, romanLingo is set when Director is first started and is determined by the local version of the system software. If you are using a non-Roman script system but don't use any double-byte characters in your script, set this property to TRUE for faster execution of your Lingo scripts.
Example	This statement sets romanLingo to TRUE, which causes Lingo to use a single-byte character set: set the romanLingo to TRUE

rotation

Syntax	the rotation of member <i>whichQuickTimeMember</i> member(<i>whichQuickTimeMember</i>).rotation sprite(<i>whichSprite</i>).rotation the rotation of sprite <i>whichSprite</i>
Description	Cast member property and sprite property; controls the rotation of a QuickTime movie, animated GIF, Flash movie, or bitmap sprite within the sprite's bounding rectangle, without rotating that rectangle or the sprite's controller (in the case of QuickTime). In effect, the sprite's bounding rectangle acts as a window through which you can see the Flash or QuickTime movie. The bounding rectangles of bitmaps and animated GIFs change to accommodate the rotating image. Score rotation works for a Flash movie only if obeyScoreRotation is set to TRUE. A Flash movie rotates around its origin point as specified by its originMode property. A QuickTime movie rotates around the center of the bounding rectangle of the sprite. A bitmap rotates around the registration point of the image. For QuickTime media, if the sprite's crop property is set to TRUE, rotating the sprite frequently moves part of the image out of the viewable area; when the sprite's crop property is set to FALSE, the image is scaled to fit within the bounding rectangle (which may cause image distortion). You specify the rotation in degrees as a floating-point number.

The Score can retain information for rotating an image from +21,474,836.47° to -21,474,836.48°, allowing 59,652 full rotations in either direction.

When the rotation limit is reached (slightly past the 59,652th rotation), the rotation resets to +116.47° or -116.48°—not 0.00°. This is because +21,474,836.47° is equal to +116.47°, and -21,474,836.48° is equal to -116.48° (or +243.12°). To avoid this reset condition, when you use Lingo to perform continuous rotation, constrain the angles to ±360°.

This property can be tested and set. The default value is 0.

Example The following behavior causes a sprite to rotate continuously by 2° every time the playback head advances, limiting the angle to 360°:

```
property spriteNum  
  
on prepareFrame me  
    sprite(spriteNum).rotation = integer(sprite(spriteNum).rotation + 2) mod 360  
end
```

Example This frame script keeps the playback head looping in the current frame while it rotates a QuickTime sprite in channel 5 a full 360° in 16° increments. When the sprite has been rotated 360°, the playback head continues to the next frame.

```
on exitFrame  
    if sprite(5).rotation < 360 then  
        sprite(5).rotation = sprite(5).rotation + 16  
        go the frame  
    end if  
end
```

Example This handler accepts a sprite reference as a parameter and rotates a Flash movie sprite 360° in 10° increments:

```
on rotateMovie whichSprite  
    repeat with i = 1 to 36  
        sprite(whichSprite).rotation = i * 10  
        updatestage  
    end repeat  
end
```

See also flipH, flipV, obeyScoreRotation, originMode

RTF

Syntax	member(<i>whichMember</i>).RTF
Description	Cast member property; allows access to the text and tags that control the layout of the text within a text cast member containing text in rich text format.
	This property can be tested and set.
Example	This statement displays in the Message window the RTF formatting information embedded in the text cast member Resume: <pre>put member("Resume").RTF</pre>
See also	HTML, importFileInto

runMode

Syntax	the runMode
Description	Function; returns a string indicating the mode in which the movie is playing. Possible values are as follows: <ul style="list-style-type: none">• Author—The movie is running in Director.• Projector—The movie is running as a projector.• BrowserPlugin—The movie is running as a Shockwave plug-in or other scripting environment, such as LiveConnect or ActiveX.• Java Applet—The movie is playing back as a Java applet. The safest way to test for particular values in this property is to use the contains operator. This helps avoid errors and allows partial matches.
Example	This statement determines whether or not external parameters are available and obtains them if they are: <pre>if the runMode contains "Plugin" then -- decode the embed parameter if externalParamName(swURL) = swURL then put externalParamValue(swURL) into myVariable end if end if</pre>
See also	environment, platform

on runPropertyDialog

Syntax

```
on runPropertyDialog me, currentInitializerList
    statement(s)
end
```

Description

System message and event handler; contains Lingo that defines specific values for a behavior's parameters in the Parameters dialog box. The `runPropertyDialog` message is sent whenever the behavior is attached to a sprite, or when the user changes the initial property values of a sprite's behavior.

The current settings for a behavior's initial properties are passed to the handler as a property list. If the `on runPropertyDialog` handler is not defined within the behavior, Director runs a behavior customization dialog box based on the property list returned by the `on getPropertyDescriptionList` handler.

Example

This handler overrides the behavior's values set in the Parameters dialog box for the behavior. New values are contained in the list `currentInitializerList`. Normally, the Parameters dialog box allows the user to set the mass and gravitational constants. However, this handler assigns these parameters constant values without displaying a dialog box:

```
property mass
property gravitationalConstant
on runPropertyDialog me, currentInitializerList
    --force mass to 10
    setaProp currentInitializerList, #mass, 10
    -- force gravitationalConstant to 9.8
    setaProp currentInitializerList, #gravitationalConstant, 9.8
    return currentInitializerList
end
```

See also

[on getBehaviorDescription](#), [on getPropertyDescriptionList](#)

safePlayer

Syntax

the `safePlayer`

Description

System property; controls whether or not safety features in Director are turned on.

In a Shockwave movie, this property can be tested but not set. It is always TRUE in Shockwave.

In the authoring environment and in projectors, the default value is FALSE. This property may be tested, but it may only be set to TRUE. Once it has been set to TRUE, it cannot be set back to FALSE without restarting Director or the projector.

When safePlayer is TRUE, the following safety features are in effect:

- Only safe Xtras may be used.
- The safePlayer property cannot be reset.
- Pasting content from the Clipboard by using the pasteClipBoardInto command generates a warning dialog box that allows the user to cancel the operation.
- Handling Macintosh resource files by using the obsolete openResFile or closeResFile command is disabled.
- Saving a movie or cast by using Lingo is disabled.
- Printing by using the printFrom command is disabled.
- Opening an application by using the open command is disabled.
- The ability to stop an application or the user's computer by using the restart or shutDown command is disabled.
- Sending strings to Windows Media Control Interface (MCI) by using mci is disabled.
- Opening a file that is outside the DSWMedia folder is disabled.
- Discovering a local file name is disabled.
- Using getNetText() or postNetText(), or otherwise accessing a URL that does not have the same domain as the movie, generates a security dialog box.

sampleCount

Syntax

sound(*channelNum*).sampleCount

the sampleCount of sound(*channelNum*)

Description

Read-only property; the number of sound samples in the currently playing sound in sound channel *channelNum*. This is the total number of samples, and depends on the sampleRate and duration of the sound. It does not depend on the channelCount of the sound.

A 1-second, 44.1 KHz sound contains 44,100 samples.

Example

This statement displays the name and sampleCount of the cast member currently playing in sound channel 1 in the Message window.

```
put "Sound cast member" && sound(1).member.name && "contains" && \
sound(1).sampleCount && "samples."
```

See also

channelCount, sampleRate, sampleSize

sampleRate

Syntax

`member(whichCastMember).sampleRate`

the sampleRate of member *whichCastMember*

`sound(channelNum).sampleRate`

Description

Cast member property; returns, in samples per second, the sample rate of the sound cast member or in the case of SWA sound, the original file that has been Shockwave Audio-encoded. This property is available only after the SWA sound begins playing or after the file has been preloaded using the `preLoadBuffer` command. When a sound channel is given, the result is the sample rate of the currently playing sound cast member in the given sound channel.

This property can be tested but not set. Typical values are 8000, 11025, 16000, 22050, and 44100.

When multiple sounds are queued in a sound channel, Director plays them all with the `channelCount`, `sampleRate`, and `sampleSize` of the first sound queued, resampling the rest for smooth playback. Director resets these properties only after the channel's sound queue is exhausted or a `stop()` command is issued. The next sound to be queued or played then determines the new settings.

Example

This statement assigns the original sample rate of the file used in SWA streaming cast member Paul Robeson to the field cast member Sound Quality:

```
member("Sound Quality").text = string(member("Paul Robeson").sampleRate)"
```

Example

This statement displays the sample rate of the sound playing in sound channel 1 in the Message window:

```
put sound(1).sampleRate
```

sampleSize

Syntax

`member(whichCastMember).sampleSize`

the sampleSize of member *whichCastMember*

`sound(channelNum).sampleSize`

Description

Cast member property; determines the sample size of the specified cast member. The result is usually a size of 8 or 16 bits. If a sound channel is given, the value is for the sound member currently playing in the given sound channel.

This property can be tested but not set.

Example	This statement checks the sample size of the sound cast member Voice Over and assigns the value to the variable soundSize: soundSize = member("Voice Over").sampleSize
Example	This statement displays the sample size of the sound playing in sound channel 1 in the Message window: put sound(1).sampleSize

save castLib

Syntax	castLib(<i>whichCast</i>).save() castLib(<i>whichCast</i>).save(<i>pathName & newFileName</i>) save castLib <i>whichCast</i> {, <i>pathName&newFileName</i> }
Description	Command; saves changes to the cast in the cast's original file or, if the optional parameter <i>pathName</i> : <i>newFileName</i> is included, in a new file. If no file name is given, the original cast must be linked. Further operations or references to the cast use the saved cast member. This command does not work with compressed files. The save CastLib command doesn't support URLs as file references.
Example	This statement causes Director to save the revised version of the Buttons cast in the new file UpdatedButtons in the same folder: castLib("Buttons").save(the moviePath & "UpdatedButtons.cst")
See also	@ (<i>pathname</i>)

on savedLocal

Syntax	on savedLocal <i>statement(s)</i> end
Description	System message and event handler; This property is provided to allow for enhancements in future versions of Shockwave.
See also	allowSaveLocal

saveMovie

Syntax saveMovie {*pathName*&*fileName*}

Description	Command; saves the current movie. Including the optional parameter saves the movie to the file specified by <i>pathName</i> : <i>fileName</i> . This command does not work with compressed files. The specified filename must include the .dir file extension. The saveMovie command doesn't support URLs as file references.
Example	This statement saves the current movie in the Update file: <code>saveMovie the moviePath & "Update.dir"</code>
See also	<code>@ (pathname)</code> , <code>setPref</code>

scale

Syntax member(*whichCastMember*).scale
the scale of member *whichCastMember*
sprite(*whichSprite*).scale
the scale of sprite *whichSprite*

Description	Cast member property and sprite property; controls the scaling of a QuickTime, vector shape, or Flash movie sprite. For QuickTime, this property does not scale the sprite's bounding rectangle or the sprite's controller. Instead, it scales the image around the image's center point within the bounding rectangle. The scaling is specified as a Director list containing two percentages stored as float-point values:
	<code>[xPercent, yPercent]</code>
	The <i>xPercent</i> parameter specifies the amount of horizontal scaling; the <i>yPercent</i> parameter specifies vertical scaling.
	When the sprite's crop property is set to TRUE, the scale property can be used to simulate zooming within the sprite's bounding rectangle. When the sprite's crop property is set to FALSE, the scale property is ignored.
	This property can be tested and set. The default value is [1.0000,1.0000].
	For Flash movie or vector shape cast members, the scale is a floating-point value. The movie is scaled from its origin point, as specified by its originMode property.
	Note: This property must be set to the default value if the scaleMode property is set to #autoSize; otherwise the sprite does not display correctly.

Example This sprite script keeps the playback head looping in the current frame while the QuickTime sprite in channel 5 is scaled down in 5% increments. When the sprite is no longer visible (because its horizontal scale value is 0% or less), the playback head continues to the next frame.

```
on exitFrame me
    scaleFactor = sprite(spriteNum).scale[1]
    currentMemberNum = sprite(spriteNum).memberNum
    if member(currentMemberNum).crop = FALSE then
        member(currentMemberNum).crop = TRUE
    end if
    if scaleFactor > 0 then
        scaleFactor = scaleFactor - 5
        sprite(spriteNum).scale = [scaleFactor, scaleFactor]
        go the frame
    end if
end
```

Example This handler accepts a reference to a Flash movie sprite as a parameter, reduces the movie's scale to 0% (so it disappears), and then scales it up again in 5% increments until it is full size (100%) again.

```
on scaleMovie whichSprite
    sprite(whichSprite).scale = 0
    updatestage
repeat with i = 1 to 20
    sprite(whichSprite).scale = i * 5
    updatestage
end repeat
end
```

See also [scaleMode](#), [originMode](#)

scaleMode

Syntax

```
sprite(whichVectorOrFlashSprite).scaleMode  
the scaleMode of sprite whichVectorOrFlashSprite  
member(whichVectorOrFlashMember).scaleMode  
the scaleMode of member whichVectorOrFlashMember
```

Description

Cast member property and sprite property; controls the way a Flash movie or vector shape is scaled within a sprite's bounding rectangle. When you scale a Flash movie sprite by setting its scale and viewScale properties, the sprite itself is not scaled; only the view of the movie within the sprite is scaled. The scaleMode property can have these values:

- #showAll (default for Director movies prior to version 7)— Maintains the aspect ratio of the original Flash movie cast member. If necessary, fill in any gap in the horizontal or vertical dimension using the background color.
- #noBorder—Maintains the aspect ratio of the original Flash movie cast member. If necessary, crop the horizontal or vertical dimension.
- #exactFit—Does not maintain the aspect ratio of the original Flash movie cast member. Stretch the Flash movie to fit the exact dimensions of the sprite.
- #noScale—preserves the original size of the Flash media, regardless of how the sprite is sized on the Stage. If the sprite is made smaller than the original Flash movie, the movie displayed in the sprite is cropped to fit the bounds of the sprite.
- #autoSize (default)—This specifies that the sprite rectangle is automatically sized and positioned to account for rotation, skew, flipH, and flipV. This means that when a Flash sprite is rotated, it will not crop as in earlier versions of Director. The #autoSize setting only functions properly when scale, viewScale, originPoint, and viewPoint are at their default values.

This property can be tested and set.

Example

This sprite script checks the Stage color of the Director movie and, if the Stage color is indexed to position 0 in the current palette, the script sets the scaleMode property of a Flash movie sprite to #showAll. Otherwise, it sets the scaleMode property to #noBorder.

```
on beginsprite me  
    if the stagecolor = 0 then  
        sprite(me.spriteNum).scaleMode = #showAll  
    else  
        sprite(me.spriteNum).scaleMode = #noBorder  
    end if  
end
```

See also

scale

score

Syntax the score

Description Movie property; determines which Score is associated with the current movie. This property can be useful for storing the current contents of the Score before wiping out and generating a new one or for assigning the current Score contents to a film loop.

This property can be tested and set.

Example This statement assigns the film loop cast member Waterfall to the Score of the current movie:

```
the score = member("Waterfall").media
```

scoreColor

Syntax sprite(*whichSprite*).scoreColor

the scoreColor of sprite *whichSprite*

Description Sprite property; indicates the Score color assigned to the sprite specified by *whichSprite*. The possible values correspond to color chips 0 to 5 in the current palette.

This property can be tested and set. Setting this property is useful only during authoring and Score recording.

Example This statement displays in the Message window the value for the Score color assigned to sprite 7:

```
put sprite(7).scorecolor
```

scoreSelection

Syntax the scoreSelection

Description Movie property; determines which channels are selected in the Score window. The information is formatted as a linear list of linear lists. Each contiguous selection is in a list format consisting of the starting channel number, ending channel number, starting frame number, and ending frame number. Specify sprite channels by their channel numbers; use the following numbers to specify the other channels.

To specify:	Use:
Frame script channel	0
Sound channel 1	-1
Sound channel 2	-2
Transition channel	-3
Palette channel	-4
Tempo channel	-5

You can select discontinuous channels or frames.

This property can be tested and set.

Example This statement selects sprite channels 15 through 25 in frames 100 through 200:
set the scoreSelection = [[15, 25, 100, 200]]

Example This statement selects sprite channels 15 through 25 and 40 through 50 in frames 100 through 200:
set the scoreSelection = [[15, 25, 100, 200], [40, 50, 100, 200]]

Example This statement selects the frame script in frames 100 through 200:
set the scoreSelection = [[0, 0, 100, 200]]

script

Syntax the script of menuitem *whichItem* of menu *whichMenu*
childObject.script

the script of *childObject*

Description Menu item and child object property.

For menu items, determines which Lingo statement is executed when the specified menu item is selected. The *whichItem* expression can be either a menu item name or a menu item number; the *whichMenu* expression can be either a menu name or a menu number.

When a menu is installed, the script is set to the text following the “ \AA ” character in the menu definition.

This property can be tested and set.

Note: Menus are not available in Shockwave.

For child objects, the return value is the name of the child’s parent script. This property can be tested but not set.

To see an example of script used in a completed movie, see the Parent Scripts movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement makes goHandler the handler that is executed when the user chooses the Go command from the custom menu Control:

set the script of menuitem "Go" of menu "Control" to "goHandler"

Example This Lingo checks whether a child object is an instance of the parent script Warrior Ant:

```
if bugObject.script = script("Warrior Ant") then  
    bugObject.attack()  
end if
```

See also checkMark, installMenu, menu, handlers(), scriptText

scriptInstanceList

Syntax

sprite(*whichSprite*).scriptInstanceList

the scriptInstanceList of sprite *whichSprite*

Description

Sprite property; creates a list of script references attached to a sprite. This property is available only during run time. The list is empty when the movie is not running. Modifications to the list are not saved in the Score. This property is useful for the following tasks:

- Attaching a behavior to a sprite for use during run time
- Determining if behaviors are attached to a sprite; determining what the behaviors are
- Finding a behavior script reference to use with the sendSprite command

This property can be tested and set. (It can be set only if the sprite already exists and has at least one instance of a behavior already attached to it.)

Example

This handler displays the list of script references attached to a sprite:

```
on showScriptRefs spriteNum  
    put sprite(spriteNum).scriptInstanceList  
end
```

Example

These statements attach the script Big Noise to sprite 5:

```
x = script("Big Noise").new()  
sprite(5).scriptInstanceList.add(x)
```

See also

scriptNum, sendSprite

scriptList

Syntax

sprite(*whichSprite*).scriptList

the scriptList of sprite *whichSprite*

Description

Sprite property; returns the list of behaviors attached to the given sprite and their properties. This property may only be set by using setScriptList(). It may not be set during a score recording session.

Example

This statement displays the list of scripts attached to sprite 1 in the Message window:

```
put sprite(1).scriptList  
-- [[(member 2 of castLib 1), "[#myRotateAngle: 10.0000, #myClockwise: 1,  
#myInitialAngle: 0.0000]"], [(member 3 of castLib 1), "[#myAnglePerFrame: 10.0000,  
#myTurnFrames: 10, #myHShiftPerFrame: 10, #myShiftFrames: 10, #myTotalFrames: 60,  
#mySurfaceHeight: 0]"]]
```

See also

setScriptList(), value()

scriptNum

Syntax `sprite(whichSprite).scriptNum`

scriptNum of sprite *whichSprite*

Description Sprite property; indicates the number of the script attached to the sprite specified by *whichSprite*. If the sprite has multiple scripts attached, scriptNum sprite property returns the number of the first script. (To see a complete list of the scripts attached to a sprite, see the behaviors listed for that sprite in the Behavior Inspector.)

This property can be tested and set during Score recording.

Example This statement displays the number of the script attached to sprite 4:

```
put sprite(4).scriptNum
```

See also [scriptInstanceList](#)

scriptsEnabled

Syntax `member(whichCastMember).scriptsEnabled`

the scriptsEnabled of member *whichCastMember*

Description Director movie cast member property; determines whether scripts in a linked movie are enabled (TRUE or 1) or disabled (FALSE or 0).

This property is available for linked Director movie cast members only.

This property can be tested and set.

Example This statement turns off scripts in the linked movie Jazz Chronicle:

```
member("Jazz Chronicle").scriptsEnabled = FALSE
```

scriptText

Syntax

`member(whichCastMember).scriptText`

the scriptText of member *whichCastMember*

Description

Cast member property; indicates the content of the script, if any, assigned to the cast member specified by *whichCastMember*.

The text of a script is removed when a movie is converted to a projector, protected, or compressed for Shockwave. Such movies then lose their values for the scriptText cast member property. Therefore, the movie's scriptText cast member property values can't be retrieved when the movie is played back by a projector. However, Director can set new values for scriptText cast member property inside the projector. These newly assigned scripts are automatically compiled so that they execute quickly.

This property can be tested and set.

Example

This statement makes the contents of field cast member 20 the script of cast member 30:

```
member(30).scriptText = member(20).text
```

scriptType

Syntax

`member whichScript.scriptType`

the scriptType of member *whichScript*

Description

Cast member property; indicates the specified script's type. Possible values are #movie, #score, and #parent.

Example

This statement makes the script member Main Script a movie script:

```
member("Main Script").scriptType = #movie
```

scrollByLine

Syntax

member(*whichCastMember*). scrollByLine(*amount*)

scrollByLine member *whichCastMember*, *amount*

Description

Command; scrolls the specified field or text cast member up or down by the number of lines specified in *amount*. (Lines are defined as lines separated by carriage returns, not lines caused by wrapping.)

- When *amount* is positive, the field scrolls down.
- When *amount* is negative, the field scrolls up.

Example

This statement scrolls the field cast member Today's News down five lines:

```
member("Today's News").scrollbyline(5)
```

Example

This statement scrolls the field cast member Today's News up five lines:

```
scrollByLine member "Today's News", -1
```

scrollByPage

Syntax

member(*whichCastMember*). scrollByPage(*amount*)

scrollByPage member *whichCastMember*, *amount*

Description

Command; scrolls the specified field or text cast member up or down by the number of pages specified in *amount*. A page is equal to the number of lines of text visible on the screen.

- When *amount* is positive, the field scrolls down.
- When *amount* is negative, the field scrolls up.

The Director player for Java doesn't support the scrollByPage member property. Use the scrollTop member property to write Lingo that scrolls text.

Example

This statement scrolls the field cast member Today's News down one page:

```
member("Today's News").scrollbypage(1)
```

Example

This statement scrolls the field cast member Today's News up one page:

```
member("Today's News").scrollbypage(-1)
```

See also

scrollTop

scrollTop

Syntax

member(*whichCastMember*).scrollTop

the scrollTop of member *whichCastMember*

Description

Cast member property; determines the distance, in pixels, from the top of a field cast member to the top of the field that is currently visible in the scrolling box. By changing the value for scrollTop member property while the movie plays, you can change the section of the field that appears in the scrolling field.

This is a way to make custom scrolling behaviors for text and field members.

For example, the following Lingo moves the field cast member Credits up or down within a field's box, depending on the value in the variable sliderVal:

```
global sliderVal  
  
on prepareFrame  
    newVal = sliderVal * 100  
    member("Credits").scrollTop = newVal  
end
```

The global variable sliderVal could measure how far the user drags a slider. The statement set newVal = sliderVal * 100 multiplies sliderVal to give a value that is greater than the distance the user drags the slider. If sliderVal is positive, the text moves up; if sliderVal is negative, the text moves down.

Example

This repeat loop makes the field Credits scroll by continuously increasing the value of scrollTop:

```
on wa  
    member("Credits").scrollTop = 1  
    repeat with count = 1 to 150  
        member("Credits").scrollTop = member("Credits").scrollTop + 1  
        updateStage  
    end repeat  
end
```

searchCurrentFolder

Syntax the searchCurrentFolder

Description System property; determines whether Director searches the current folder when searching file names. This property is TRUE by default.

- When the searchCurrentFolder property is TRUE (1), Director searches the current folder when resolving file names.
- When the searchCurrentFolder property is FALSE (0), Director does not search the current folder when resolving file names.

This property can be tested and set.

Example This statement displays the status of the searchCurrentFolder property in the Message window:

```
put the searchCurrentFolder
```

The result is 1, which is the numeric equivalent of TRUE.

Example This statement sets the searchCurrentFolder property to TRUE, which causes Director to search the current folder when resolving file names:

```
the searchCurrentFolder = TRUE
```

See also searchPaths

searchPath

This is obsolete. Use searchPaths.

searchPaths

Syntax the searchPaths

Description System property; a list of paths that Director searches when trying to find linked media such as digital video, GIFs, bitmaps, or sound files. Each item in the list is a fully qualified pathname as it appears on the current platform at run time.

The value of searchPaths is a linear list that you can manipulate the same as any other list by using commands such as add, addAt, append, deleteAt, and setAt.

URLs should not be used as file references in the search paths.

Adding a large number of paths to searchPaths slows searching. Try to minimize the number of paths in the list.

This property can be tested and set, and is an empty list by default.

Note: This property will function on all subsequent movies after being set. Because the current movie's assets have already been loaded, changing the setting will not affect any of these assets.

Example This statement displays the paths that Director searches when resolving file names:

```
put the searchPaths
```

Example This statement assigns two folders to searchPaths in Windows. This version includes optional trailing backslashes.

```
set the searchPaths = ["c:\director\projects\", "d:\cdrom\sources\"]
```

This statement is the same, except that trailing backslashes have been omitted:

```
set the searchPaths = ["c:\director\projects", "d:\cdrom\sources"]
```

Example This statement assigns two folders to searchPaths on a Macintosh. This version includes optional trailing colons.

```
set the searchPaths = ["hard drive:director:projects:", "cdrom:sources:]
```

This statement is the same, except that trailing colons have been omitted:

```
set the searchPaths = ["hard drive:director:projects", "cdrom:sources"]
```

Example These statements cause Director to search in a folder named Sounds, which is in the same folder as the current Director movie:

```
set soundPaths = the moviePath & "Sounds"
```

```
add the searchPaths, soundPath
```

See also searchCurrentFolder, @ (pathname)

seconds

Syntax *dateObject.seconds*

- Description** Property; gives the seconds passed since midnight of the given date object. Only the `systemDate`, `creationDate`, and `modifiedDate` have a default seconds value. You must specify a seconds value for date objects that you create.
- This property can be used with the `creationDate` and the `modifiedDate` for source control purposes.

Example These statements display the seconds since midnight on the authoring computer:

```
mydate = the systemdate  
put mydate.seconds  
-- 1233
```

selectedText

Syntax *member(whichTextMember).selectedText*

- Description** Text cast member property; returns the currently selected chunk of text as a single object reference. This allows access to font characteristics as well as to the string information of the actual characters.

Example This handler displays the currently selected text being placed in a local variable object. Then that object is used to reference various characteristics of the text, which are detailed in the Message window.

```
property spriteNum  
  
on mouseUp me  
    mySelectionObject = sprite(spriteNum).member.selectedText  
    put mySelectionObject.text  
    put mySelectionObject.font  
    put mySelectionObject.fontSize  
    put mySelectionObject.fontStyle  
end
```

selection() (function)

Syntax	the selection
Description	Function; returns a string containing the highlighted portion of the current editable field. This function is useful for testing what a user has selected in a field. The selection function only indicates which string of characters is selected; you cannot use selection to select a string of characters.
Example	This statement checks whether any characters are selected and, if none are, displays the alert "Please select a word.": if the selection = EMPTY then alert "Please select a word."
See also	selStart, selEnd

selection (cast property)

Syntax	castLib (<i>whichCast</i>).selection
	the selection of castLib <i>whichCast</i>
	set the selection of castLib whichCast =[[startMember1 , endMember1] , \[[startMember2 , endMember2] , [startMember3 , endMember3]...]]
	castLib(<i>whichCast</i>). selection =[[startMember1 , endMember1] , \[[startMember2 , endMember2] , [startMember3 , endMember3]...]]
Description	Cast property; determines which cast members are selected in the specified Cast window. The range appears as a list of the starting and ending cast member numbers for the selected range. You can include more than one selection by specifying additional ranges of cast members. (To specify more than one selection, Control-drag (Windows) or Command-drag (Macintosh). This property can be tested and set.
Example	This statement selects cast members 1 through 10 in castLib 1: castLib(1).selection = [[1, 10]] This statement selects cast members 1 through 10, and 30 through 40, in castLib 1: castLib(1).selection = [[1, 10], [30, 40]]

selection (text cast member property)

Syntax	member(<i>whichTextMember</i>).selection
Description	Text cast member property; returns a list of the first and last character selected in the text cast member. This property can be tested and set.

Example	The following statement sets the selection displayed by the sprite of text member myAnswer so that characters 6 through 10 are highlighted:
	<code>member("myAnswer").selection = [6, 10]</code>
See also	color() , selStart , selEnd

selEnd

Syntax	<code>the selEnd</code>
Description	Global property; specifies the last character of a selection. It is used with selStart to identify a selection in the current editable field, counting from the beginning character.
	This property can be tested and set. The default value is 0.
Example	These statements select “cde” from the field “abcdefg”:
	<code>the selStart = 3 the selEnd = 5</code>
Example	This statement calls the handler noSelection when selEnd is the same as selStart :
	<code>if the selEnd = the selStart then noSelection</code>
Example	This statement makes a selection 20 characters long:
	<code>the selEnd = the selStart + 20</code>
See also	editable , hilite (command), selection() (function), selStart , text

selStart

Syntax	<code>the selStart</code>
Description	Cast member property; specifies the starting character of a selection. It is used with selEnd to identify a selection in the current editable field, counting from the beginning character.
	This property can be tested and set. The default value is 0.
Example	These statements select “cde” from the field “abcdefg”:
	<code>the selStart = 3 the selEnd = 5</code>
Example	This statement calls the handler noSelection when selEnd is the same as selStart :
	<code>if the selEnd = the selStart then noSelection</code>
Example	This statement makes a selection 20 characters long:
	<code>the selEnd = the selStart + 20</code>
See also	selection() (function), selEnd , text

sendAllSprites

Syntax

```
sendAllSprites (#customEvent, args)
```

Description

Command; sends a designated message to all sprites, not just the sprite that was involved in the event. As with any other message, the message is sent to every script attached to the sprite, unless the stopEvent command is used.

For best results, send the message only to those sprites that will properly handle the message through the sendSprite command. No error will occur if the message is sent to all the sprites, but performance may decrease. There may also be problems if different sprites have the same handler in a behavior, so avoid conflicts by using unique names for messages that will be broadcast.

After the message has been passed to all behaviors, the event follows the regular message hierarchy: cast member scripts, frame script, and then movie script.

When you use the sendAllSprites command, be sure to do the following:

- Replace *customEvent* with the message.
- Replace *args* with any arguments to be sent with the message.

If no sprite has an attached behavior containing the given handler, sendAllSprites returns FALSE.

Example

This handler sends the custom message allSpritesShouldBumpCounter and the argument 2 to all sprites when the user clicks the mouse:

```
on mouseDown me
    sendAllSprites (#allSpritesShouldBumpCounter, 2)
end
```

See also

[sendSprite](#)

sendSprite

Syntax

```
sendSprite (whichSprite, #customMessage, args)
```

Description

Command; sends a message to all scripts attached to a specified sprite.

Messages sent using sendSprite are sent to each of the scripts attached to the sprite. The messages then follow the regular message hierarchy: cast member script, frame script, and movie script.

If the given sprite does not have an attached behavior containing the given handler, sendSprite returns FALSE.

Example

This handler sends the custom message bumpCounter and the argument 2 to sprite 1 when the user clicks:

```
on mouseDown me
    sendSprite (1, #bumpCounter, 2)
end
```

See also

[sendAllSprites](#)

serialNumber

Syntax the serialNumber

Description Movie property; a string containing the serial number entered when Director was installed.

This property is available in the authoring environment only. It could be used in a movie in a window (MIAW) tool that is personalized to show the user's information.

Example This handler would be located in a movie script of a MIAW. It places the user's name and the serial number into a display field when the window is opened:

```
on prepareMovie
    displayString = the userName
    put RETURN&the organizationName after displayString
    put RETURN&the serialNumber after displayString
    member("User Info").text = displayString
end
```

See also organizationName, userName, window

set...to, set...=

Syntax set the *lingoProperty* to *expression*

the *lingoProperty* = *expression*

set *variable* to *expression*

variable = *expression*

Description Command; evaluates an expression and puts the result in the property specified by *lingoProperty* or the variable specified by *variable*.

Example This statement sets the name of member 3 to Sunset:

```
set member(3).name = "Sunset"
```

Example This statement sets the soundEnabled property to the opposite of its current state. When soundEnabled is TRUE (the sound is on), this statement turns it off. When soundEnabled is FALSE (the sound is off), this statement turns it on.

```
set the soundEnabled = not (the soundEnabled)
```

Example This statement sets the variable vowels to the string "aeiou":

```
set vowels = "aeiou"
```

See also property

setAlpha()

Syntax

imageObject.setAlpha(alphaLevel)
imageObject.setAlpha(alphaImageObject)

Description

Function; sets the alpha channel of an image object to a flat alphaLevel or to an existing *alphaImageObject*. The *alphaLevel* must be a number from 0–255. Lower values cause the image to appear more transparent. Higher values cause the image to appear more opaque. The value 255 has the same effect as a value of zero. In order for the *alphaLevel* to have effect, the *useAlpha()* of the image object must be set to TRUE.

The image object must be 32-bit. If you specify an alpha image object, it must be 8-bit. Both images must have the same dimensions. If these conditions are not met, *setAlpha()* has no effect and returns FALSE. The function returns TRUE when it is successful.

Example

This Lingo statement makes the image of the bitmap cast member Foreground opaque and disables the alpha channel altogether. This is a good method for removing the alpha layer from an image:

```
member("Foreground").image.setAlpha(255)  
member("Foreground").image.useAlpha = FALSE
```

This Lingo gets the alpha layer from the cast member Sunrise and places it into the alpha layer of the cast member Sunset:

```
tempAlpha = member("Sunrise").image.extractAlpha()  
member("Sunset").image.setAlpha(tempAlpha)
```

See also

useAlpha(), *extractAlpha()*

setaProp

Syntax

setaProp list, listProperty, newValue
setaProp (childObject, listProperty, newValue)
list.listProperty = newValue
list[listProperty] = newValue
childObject.listProperty = newValue

Description

Command; replaces the value assigned to *listProperty* with the value specified by *newValue*. The *setaProp* command works with property lists and child objects. Using *setaProp* with a linear list produces a script error.

- For property lists, *setaProp* replaces a property in the list specified by *list*. When the property isn't already in the list, Lingo adds the new property and value.
- For child objects, *setaProp* replaces a property of the child object. When the property isn't already in the object, Lingo adds the new property and value.
- The *setaProp* command can also set ancestor properties.

Example These statements create a property list and then adds the item #c:10 to the list:

```
newList = [#a:1, #b:5]
put newList
-- [#a:1, #b:5]
setaProp newList, #c, 10
put newList
```

Example Using the dot operator, you can alter the property value of a property already in a list without using setaProp:

```
newList = [#a:1, #b:5]
put newList
-- [#a:1, #b:5]
newList.b = 99
put newList
-- [#a:1, #b:99]
```

Note: To use the dot operator to manipulate a property, the property must already exist in the list, child object, or behavior.

See also ancestor, property, . (dot operator)

setAt

Syntax setAt *list*, *orderNumber*, *value*

list[*orderNumber*] = *value*

Description Command; replaces the item specified by *orderNumber* with the value specified by *value* in the list specified by *list*. When *orderNumber* is greater than the number of items in a property list, the setAt command returns a script error. When *orderNumber* is greater than the number of items in a linear list, Director expands the list's blank entries to provide the number of places specified by *orderNumber*.

Example This handler assigns a name to the list [12, 34, 6, 7, 45], replaces the fourth item in the list with the value 10, and then displays the result in the Message window:

```
on enterFrame
    set vNumbers = [12, 34, 6, 7, 45]
    setAt vnumbers, 4, 10
    put vNumbers
end enterFrame
```

When the handler runs, the Message window displays the following:

[12, 34, 6, 10, 45]

You can perform this same operation may be done using bracket access to the list in the following manner:

```
on enterFrame
    set vNumbers = [12, 34, 6, 7, 45]
    vnumbers[4] = 10
    put vNumbers
end enterFrame
```

When the handler runs, the Message window displays the following:

```
[12, 34, 6, 10, 45]
```

See also [\[\] \(bracket access\)](#)

setFlashProperty()

Syntax `sprite(spriteNum).setFlashProperty("targetName", #property, newValue)`

Description Function; allows Lingo to call the Flash action script function `setProperty()` on the given Flash sprite. Use the `setFlashProperty()` function to set the properties of movie clips or levels within a Flash movie. This is similar to setting sprite properties within Director.

The `targetName` is the name of the movie clip or level whose property you want to set within the given Flash sprite.

The `#property` is the name of the property to set. You can set the following movie clip properties: `#posX`, `#posY`, `#scaleX`, `#scaleY`, `#visible`, `#rotate`, `#alpha`, and `#name`.

To set a global property of the Flash sprite, pass an empty string as the `targetName`. You can set the global Flash properties: `#focusRect` and `#spriteSoundBufferTime`.

See the Flash documentation for descriptions of these properties.

Example This statement sets the value of the `#rotate` property of the movie clip Star in the Flash member in sprite 3 to 180.

```
sprite(3).setFlashProperty("Star", #rotate, 180)
getFlashProperty()
```

setPixel()

Syntax

```
imageObject.setPixel(x, y, colorObject)  
imageObject.setPixel(point(x, y), colorObject)  
imageObject.setPixel(x, y, integerValue)  
imageObject.setPixel(point(x, y), integerValue)
```

Description

Function; sets the color value of the pixel at the specified point in the given image object, to either *colorObject* or to a raw integer with *integerValue*. If you are setting many pixels to the color of another pixel with *getPixel()*, it is faster to set them as raw integers.

For best performance with color objects, with 8-bit or lower images use an indexed color object, and with 16-bit or higher images, use an RGB color object.

The *SetPixel()* function returns FALSE if the specified pixel falls outside the specified image.

To see an example of *setPixel()* used in a completed movie, see the Imaging movie in the LearningLingo Examples folder inside the Director application folder.

Example

This Lingo statement draws a horizontal black line 50 pixels from left to right in cast member 5.

```
repeat with x = 1 to 50  
    member(5).image.setPixel(x, 0, rgb(0, 0, 0))  
end repeat
```

See also

getPixel(), *draw()*, *fill()*, *color()*

setPlaylist()

Syntax

```
sound(channelNum).setPlaylist([#member: member(whichmember), {#startTime:  
    milliseconds, #endTime: milliseconds, #loopCount: numberofLoops, #loopStartTime:  
    milliseconds, #loopEndTime: milliseconds, #preloadTime: milliseconds}]. . . )  
  
setPlaylist(sound(channelNum), [#member: member(whichmember), {#startTime:  
    milliseconds, #endTime: milliseconds, #loopCount: numberofLoops, #loopStartTime:  
    milliseconds, #loopEndTime: milliseconds, #preloadTime: milliseconds}]. . . )
```

Description

Function; sets or resets the playlist of the given sound channel. This command is useful for queueing several sounds at once.

You can specify these parameters for each sound to be queued:

Property	Description
#member	The sound cast member to queue. This property must be provided; all others are optional.
#startTime	The time within the sound at which playback begins, in milliseconds. The default is the beginning of the sound. See startTime.
#endTime	The time within the sound at which playback ends, in milliseconds. The default is the end of the sound. See endTime.
#loopCount	The number of times to play a loop defined with #loopStartTime and #loopEndTime. The default is 1. See loopCount.
#loopStartTime	The time within the sound to begin a loop , in milliseconds. See loopStartTime.
#loopEndTime	The time within the sound to end a loop, in milliseconds. See loopEndTime.
#preloadTime	The amount of the sound to buffer before playback, in milliseconds. See preloadTime.

To see an example of setPlaylist() used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This handler queues and plays the cast member introMusic, starting at its 3-second point, with a loop repeated 5 times from the 8-second point to the 8.9-second point, and stopping at the 10-second point.

```
on playMusic
    sound(2).queue([#member: member("introMusic"), #startTime: 3000,
        #endTime: 10000, #loopCount: 5, #loopStartTime: 8000, #loopEndTime: 8900])
    sound(2).play()
end
```

See also endTime, getPlaylist(), startTime, loopCount, loopEndTime, loopStartTime, member (sound property), play() (sound), preLoadTime, queue()

setPref

Syntax

setPref *prefName*, *prefValue*

Description

Command; writes the string specified by *prefValue* in the file specified by *prefName* on the computer's local disk. The file is a standard text file.

The *prefName* argument must be a valid file name. To make sure the file name is valid on all platforms, use no more than eight alphanumeric characters for the file name.

After the setPref command runs, if the movie is playing in a browser, a folder named Prefs is created in the Plug-In Support folder. The setPref command can write only to that folder.

If the movie is playing in a projector or Director, a folder is created in the same folder as the application. The folder receives the name *Prefs*.

Do not use this command to write to read-only media. Depending on the platform and version of the operating system, you may encounter errors or other problems.

This command does not perform any sophisticated manipulation of the string data or its formatting. You must perform any formatting or other manipulation in conjunction with getPref(); you can manipulate the data in memory and write it over the old file using setPref.

In a browser, data written by setPref is not private; any Shockwave movie can read this information and upload it to a server. Do not store confidential information using setPref.

To see an example of setPref used in a completed movie, see the Read and Write Text movie in the Learning\Lingo Examples folder inside the Director application folder.

Example

This handler saves the contents of the field cast member Text Entry in a file named DayWare settings:

```
on mouseUp me
    setPref "CurPrefs", member("Text Entry").text
end
```

See also

getPref()

setProp

Syntax

`setProp list, property, newValue`

`list.listProperty = newValue`

`list[listProperty] = newValue`

Description

Command; replaces the value assigned to *property* with the value specified by *newValue* in the list specified by *list*. If the list doesn't contain the specified property, setProp returns a script error.

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

This command is similar to the setaProp command, except that setProp returns an error when the property is not already in the list.

Example

This statement changes the value assigned to the age property of property list x to 11:

`setProp x, #age, 11`

Example

Using the dot operator, you can alter the property value of a property already in a list, exactly as above:

`x.age = 11`

See also

[setaProp](#)

setScriptList()

Syntax

`sprite(whichSprite).setScriptList(scriptList)`

Description

This command sets the scriptList of the given sprite. The scriptList indicates which scripts are attached to the sprite and what the settings of each script property are. By setting this list, you can change which behaviors are attached to a sprite or change the behavior properties.

The list takes the form:

`[[(whichBehaviorMember), " [#property1: value, #property2: value, . . .] ",
[(whichBehaviorMember), " [#property1: value, #property2: value, . . .] "]]`

This command cannot be used during a score recording session. Use setScriptList() for sprites added during score recording after the score recording session has ended.

See also

[scriptList](#), [value\(\)](#), [string\(\)](#)

setTrackEnabled

Syntax

```
sprite(whichSprite).setTrackEnabled(whichTrack, trueOrFalse )  
setTrackEnabled(sprite whichSprite, whichTrack, trueOrFalse)
```

Description

Command; determines whether the specified track in the digital video is enabled to play.

- When setTrackEnabled is TRUE, the specified track is enabled and playing.
- When setTrackEnabled is FALSE, the specified track is disabled and muted. For video tracks, this means they will no longer be updated on the screen.

To test whether a track is already enabled, test the trackEnabled sprite property.

Example

This statement enables track 3 of the digital video assigned to sprite channel 8:

```
sprite(8).setTrackEnabled(3, TRUE)
```

See also

trackEnabled

setVariable()

Syntax

```
setVariable(sprite flashSpriteNum, "variableName", newValue)
```

Description

Function; sets the value of the given variable in the given Flash sprite. Flash variables were introduced in Flash version 4.

Example

This statement sets the value of the variable currentURL in the Flash cast member in sprite 3. The new value of currentURL will be “<http://www.macromedia.com/software/flash/>”.

```
setVariable(sprite 3, "currentURL", "http://www.macromedia.com/software/flash/")
```

See also

hitTest(), getVariable()

shapeType

Syntax

```
member(whichCastMember).shapeType
```

the shapeType of member *whichCastMember*

Description

Shape cast member property; indicates the specified shape's type. Possible types are #rect, #roundRect, #oval, and #line. You can use this property to specify a shape cast member's type after creating the shape cast member using Lingo.

Example

These statements create a new shape cast member numbered 100 and then define it as an oval:

```
new(#shape, member 100)  
member(100).shapeType = #oval
```

shiftDown

Syntax the shiftDown

Description System property; indicates whether the user is pressing the Shift key (TRUE) or not (FALSE).

In the Director player for Java, this function returns TRUE only if the Shift key and another key are pressed simultaneously. If the Shift key is pressed by itself, shiftDown returns FALSE.

The Director player for Java supports key combinations with the Shift key. However, the browser receives the keys before the movie and thus responds to and intercepts any key combinations that are also browser keyboard shortcuts.

This property must be tested in conjunction with another key.

Example This statement checks whether the Shift key is being pressed and calls the handler doCapitalA if it is:

```
if (the shiftDown) then doCapitalA (the key)
```

See also commandDown, controlDown, key(), optionDown

short

See date() (system clock), time()

showGlobals

Syntax showGlobals

Description Command; displays all global variables in the Message window. This command is useful for debugging scripts.

Example This statement displays all global variables in the Message window:

```
showGlobals
```

See also clearGlobals, showLocals, global, globals

showLocals

Syntax showLocals

Description Command; displays all global variables in the Message window. This command is useful only within handlers or parent scripts that contain local variables to display. All variables used in the Message window are automatically global.

Local variables in a handler are no longer available after the handler executes.
Inserting the statement

showLocals

in a handler displays all the local variables in that handler in the Message window.

This command is useful for debugging scripts.

See also clearGlobals, showGlobals, global

showProps()

Syntax member(*whichFlashOrVectorCastMember*). showProps
member(*whichFlashOrVectorCastMember*).showProps()
sprite(*whichFlashOrVectorSprite*).showProps()
sound(*channelNum*).showProps()

Description Command; displays a list of the current property settings of a Flash movie, Vector member, or currently playing sound in the Message window. This command is useful for authoring only; it does not work in projectors or in Shockwave movies.

Example This handler accepts the name of a cast as a parameter, searches that cast for Flash movie cast members, and displays the cast member name, number, and properties in the Message window:

```
on ShowCastProperties whichCast
    repeat with i = 1 to the number of members of castLib whichCast
        castType = member(i, whichCast).type
        if (castType = #flash) OR (castType = #vectorShape) then
            put castType&&"cast member" && i & ":" && member(i, whichCast).name
            put RETURN
            member(i ,whichCast).showProps()
        end if
    end repeat
end
```

See also queue(), setPlayList()

showResFile

Description This Lingo is obsolete.

showXlib

Syntax

```
showXlib {Xlibfilename}
```

Description

Command; shows all Xtras in *Xlibfilename* (which must be open), or all open Xlibraries if no file is specified. Xlibrary files are resource files that contain DLLs (Windows) or Xtra resources (Macintosh). Because the types of Xlibrary files in Windows and on the Macintosh differ, the list of files that the showXlib command generates is different on different platforms.

The showXlib command doesn't support URLs as file references.

You can use interface() to display online documentation for an Xtra.

Note: This command is not supported in Shockwave.

Example

This statement displays the Xtras in the VideoDisc Xlibrary:

```
showXlib "VideoDisc Xlibrary"
```

See also

closeXlib, interface(), openXlib

shutDown

Syntax

```
shutDown
```

Description

Command; closes all open applications and turns off the computer.

Example

This statement checks whether the user has pressed Control+S (Windows) or Command+S (Macintosh) and, if so, shuts down the computer:

```
if the key = "s" and the commandDown then  
    shutDown  
end if
```

See also

quit, restart

sin()

Syntax

```
sin(angle)
```

Description

Math function; calculates the sine of the specified angle. The angle must be expressed in radians as a floating-point number.

Example

The following statement calculates the sine of pi/2:

```
put sin (PI/2.0)  
-- 1
```

See also

PI

size

Syntax member(*whichCastMember*).size
the size of member *whichCastMember*

Description Cast member property; returns the size in memory, in bytes, of a specific cast member number or name. Divide bytes by 1024 to convert to kilobytes.

Example The following line outputs the size of the cast member Shrine in a field named How Big:

```
member("How Big").text = string(member("shrine").size)
```

skew

Syntax sprite(*whichSpriteNumber*).skew

Description Sprite property; returns, as a float value in hundredths of a degree, the angle to which the vertical edges of the sprite are tilted (skewed) from the vertical. Negative values indicate a skew to the left; positive values indicate a skew to the right. Values greater than 90° flip an image vertically.

The Score can retain information for skewing an image from +21,474,836.47° to -21,474,836.48°, allowing 59,652 full rotations in either direction.

When the skew limit is reached (slightly past the 59,652th rotation), the skew resets to +116.47° or -116.48°—not 0.00°. This is because +21,474,836.47° is equal to +116.47°, and -21,474,836.48° is equal to -116.48° (or +243.12°). To avoid this reset condition, constrain angles to ±360° in either direction when using Lingo to perform continuous skewing.

Example The following behavior causes a sprite to skew continuously by 2° every time the playback head advances, while limiting the angle to 360°:

```
property spriteNum  
  
on prepareFrame me  
    sprite(spriteNum).skew = integer(sprite(spriteNum).skew + 2) mod 360  
end
```

See also flipH, flipV, rotation

sort

Syntax	list.sort() sort <i>list</i>
Description	Command; puts list items into alphanumeric order. <ul style="list-style-type: none">• When the list is a linear list, the list is sorted by values.• When the list is a property list, the list is sorted alphabetically by properties. After a list is sorted, it maintains its sort order even when you add new variables using the add command.
Example	This statement puts the list <i>Values</i> , which consists of [#a: 1, #d: 2, #c: 3], into alphanumeric order. The result appears below the statement. <pre>put values -- [#a: 1, #d: 2, #c: 3] values.sort() put values --[#a: 1, #c: 3, #d: 2]</pre>

sound

Syntax	member(<i>whichCastMember</i>).sound the sound of member <i>whichCastMember</i>
Description	Cast member property; controls whether a movie, digital video, or Flash movie's sound is enabled (TRUE, default) or disabled (FALSE). In flash members, the new setting takes effect after the currently playing sound ends. This property can be tested and set. To see an example of sound used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.
Example	This handler accepts a member reference and toggles the member's sound property on or off: <pre>on ToggleSound whichMember member(whichMember).sound = not member(whichMember).sound end</pre>

soundBusy()

Syntax

`soundBusy(whichChannel)`

Description

Function; determines whether a sound is playing (TRUE) or not playing (FALSE) in the sound channel specified by *whichChannel*.

Make sure that the playback head has moved before using `soundBusy()` to check the sound channel. If this function continues to return FALSE after a sound should be playing, add the `updateStage` command to start playing the sound before the playback head moves again.

This function works for those sound channels occupied by actual audio cast members. QuickTime, Flash, and Shockwave audio handle sound differently, and this function will not work with those media types.

Example

This statement checks whether a sound is playing in sound channel 1 and loops in the frame if it is. This allows the sound to finish before the playback head goes to another frame.

```
if soundBusy(1) then go to the frame
```

See also

`sound playFile`, `sound stop`

soundChannel

Syntax

`member(whichCastMember).soundChannel`

the `soundChannel` of member *whichCastMember*

Description

Shockwave Audio (SWA) cast member property; specifies the sound channel in which the SWA sound plays.

If no channel number or channel 0 is specified, the SWA streaming cast member assigns the sound to the highest numbered sound channel that is unused.

Shockwave Audio streaming sounds can appear as sprites in sprite channels, but they play sound in a sound channel. Refer to SWA sound sprites by their sprite channel number, not their sound channel number.

This property can be tested and set.

Example

This statement tells the SWA streaming cast member Frank Zappa to play in sound channel 3:

```
member("Frank Zappa").soundChannel = 3
```

sound close

This is obsolete. Use `puppetSound` instead.

soundDevice

Syntax	the soundDevice
Description	System property; allows the sound mixing device to be set while the movie plays. The possible settings are the devices listed in soundDeviceList.
	Several sound devices can be referenced. The various sound devices for Windows have different advantages.
	<ul style="list-style-type: none">• MacroMix (Windows)—The lowest common denominator for Windows playback. This device functions on any Windows computer, but its latency is not as good as that of other devices.• QT3Mix (Windows)—Mixes sound with QuickTime audio and possibly with other applications if they use DirectSound. This device requires that QuickTime be installed and has better latency than MacroMix.• MacSoundManager (Macintosh)—The only sound device available on the Macintosh.
Example	This statement sets the sound device to the MacroMix for a Windows computer. If the newly assigned device fails, the soundDevice property is not changed. <pre>set the soundDevice = "MacroMix"</pre>
See also	soundDeviceList

soundDeviceList

Syntax	the soundDeviceList
Description	System property; creates a linear list of sound devices available on the current computer. For the Macintosh, this property lists one device, MacSoundManager. This property can be tested but not set.
Example	This statement displays a typical sound device list on a Windows computer: <pre>Put the soundDeviceList --["QT3Mix", "MacroMix", "DirectSound"]</pre>
See also	soundDevice

soundEnabled

Syntax	the soundEnabled
Description	System property; determines whether the sound is on (TRUE, default) or off (FALSE). When you set this property to FALSE, the sound is turned off, but the volume setting is not changed.
	This property can be tested and set.
Example	This statement sets soundEnabled to the opposite of its current setting; it turns the sound on if it is off and turns it off if it is on: <code>the soundEnabled = not(the soundEnabled)</code>
See also	soundLevel, volume (sound channel), volume (sprite property)

sound fadeIn

Syntax	<code>sound(whichChannel).fadeIn()</code> <code>sound fadeIn whichChannel</code> <code>sound(whichChannel).fadeIn(ticks)</code> <code>sound fadeIn whichChannel, ticks</code>
Description	Command; fades in a sound in the specified sound channel over a period of frames or ticks. <ul style="list-style-type: none">• When <i>ticks</i> is specified, the fade in occurs evenly over that period of time.• When <i>ticks</i> is not specified, the default number of ticks is calculated as $15 * (60 / (\text{tempo setting}))$ based on the tempo setting for the first frame of the fade in. The fade in continues at a predetermined rate until the number of ticks has elapsed, or until the sound in the specified channel changes.
Example	This statement fades in the sound in channel 1 over 5 seconds: <code>sound(1).fadeIn(5 * 60)</code>
See also	sound fadeOut, fadeTo()

sound fadeOut

Syntax

```
sound(whichChannel).fadeOut()  
sound fadeOut whichChannel  
sound(whichChannel).fadeOut(ticks)  
sound fadeOut whichChannel, ticks
```

Description

Command; fades out a sound in the specified sound channel over a period of frames or ticks.

- When *ticks* is specified, the fade out occurs evenly over that period of time.
- When *ticks* is not specified, the default number of ticks is calculated as $15 * (60 / (\text{tempo setting}))$ based on the tempo setting for the first frame of the fade out.

The fade out continues at a predetermined rate until the number of ticks has elapsed, or until the sound in the specified channel changes.

If the sound is stopped before it reaches the minimum volume, it remains at the level it was stopped at, causing subsequent playback to be at this volume. Be sure to allow the sound to finish fading completely.

Note: You may want to use the volume sound property to create a custom sound fade to allow more control over the actual volume of the channel.

Example

This statement fades out the sound in channel 1 over 5 seconds:

```
sound(1).fadeOut(5 * 60)
```

See also

puppetSound, sound fadeln, volume (sprite property), fadeTo()

soundKeepDevice

Syntax

the soundKeepDevice

Description

System property; for Windows only, prevents the sound driver from unloading and reloading each time a sound needs to play. The default value is TRUE.

You may need to set this property to FALSE before playing a sound to ensure that the sound device is unloaded and made available to other applications or processes on the computer after the sound has finished.

Setting this property to FALSE may adversely affect performance if sound playback is used frequently throughout the Director application.

This property can be tested and set.

Example

This statement sets the soundKeepDevice property to FALSE:

```
set the soundKeepDevice = FALSE
```

soundLevel

Syntax the soundLevel

Description System property; sets the volume level of the sound played through the computer's speaker. Possible values range from 0 (no sound) to 7 (the maximum, default).

In Windows, the system sound setting combines with the volume control of the external speakers. Thus, the actual volume that results from setting the sound level can vary. Avoid setting the soundLevel property unless you are sure that the result is acceptable to the user. It is better to set the individual volumes of the channels and sprites with volume of sound, volume of member, and volume of sprite.

These values correspond to the settings in the Macintosh Sound control panel. Using this property, Lingo can change the sound volume directly or perform some other action when the sound is at a specified level.

This property can be tested and set.

To see an example of soundLevel used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement sets the variable oldSound equal to the current sound level:

```
oldSound = the soundLevel
```

Example This statement sets the sound level to 5:

```
the soundLevel = 5
```

See also soundEnabled, volume (sound channel)

the soundMixMedia

Syntax the soundMixMedia

Description This global property enables Flash cast members to mix their sound with sounds in the score sound channels when it is set to TRUE. When it is set to FALSE, these sounds will not be mixed and must be played at separate times. This property defaults to TRUE for movies made with Director 7 and later and false for earlier ones.

This property is valid only on Windows. When the soundMixMedia is TRUE, Director takes over the mixing and playback of sounds from Flash cast members. It is possible that slight differences may occur in the way Flash sounds play back. To hear the Flash sounds exactly they would be rendered in Flash, set this property to FALSE.

sound playFile

Syntax

sound playFile *whichChannel*, *whichFile*

Description

Command; plays the AIFF, SWA, AU, or WAV sound located at *whichFile* in the sound channel specified by *whichChannel*. For the sound to be played properly, the correct MIX Xtra must be available to the movie, usually in the Xtras folder of the application.

When the sound file is in a different folder than the movie, *whichFile* must specify the full path to the file.

To play sounds obtained from a URL, it's usually a good idea to use downloadNetThing or preloadNetThing() to download the file to a local disk first. This approach can minimize problems that may occur while the file is downloading.

The sound playFile command streams files from disk rather than playing them from RAM. As a result, using the sound playFile command when playing digital video or when loading cast members into memory can cause conflicts when the computer tries to read the disk in two places at once.

Example

This statement plays the file named Thunder in channel 1:

```
sound playFile 1, "Thunder.wav"
```

Example

This statement plays the file named Thunder in channel 3:

```
sound playFile 3, the moviePath & "Thunder.wav"
```

See also

sound stop

sound stop

Syntax

sound(*whichChannel*).stop()

sound stop *whichChannel*

Description

Command; stops the sound playing in the specified channel.

The sound stop command was used in earlier versions of Director. For best results, use the puppetSound command instead.

Example

These statements stop any sound playing in sound channel 1:

```
sound(1).stop()
```

Example

This statement checks whether a sound is playing in sound channel 1 and, if it is, stops the sound:

```
if soundBusy(1) then sound(1).stop()
```

See also

puppetSound, soundBusy()

sourceRect

Syntax	<code>window <i>whichWindow</i>.sourceRect</code> the sourceRect of window <i>whichWindow</i>
Description	Window property; specifies the original Stage coordinates of the movie playing in the window specified by <i>whichWindow</i> . This property is useful for returning a window to its original size and position after it has been dragged or its rectangle has been set.
Example	This statement displays the original coordinates of the Stage named Control Panel in the Message window: <code>put window("Control Panel").sourceRect</code>
See also	<code>drawRect</code> , <code>rect()</code>

SPACE

Syntax	SPACE
Description	Constant; read-only, value that represents the space character.
Example	This statement displays “Age Of Aquarius” in the Message window: <code>put "Age"&SPACE&"Of"&SPACE&"Aquarius"</code>

sprite

Syntax	<code>sprite(<i>whichSprite</i>).property</code> the property of sprite <i>whichSprite</i>
Description	Keyword; tells Lingo that the value specified by <i>whichSprite</i> is a sprite channel number. It is used with every sprite property. A sprite is an occurrence of a cast member in a sprite channel of the Score. This term has special meaning in the Director player for Java. Don't use the term <code>sprite</code> in Java code that you embed within a Lingo script.
Example	This statement sets the variable named horizontal to the locH of sprite 1: <code>horizontal = sprite(1).loc</code>
Example	This statement displays the current member in sprite channel 100 in the Message window: <code>put sprite (100).member</code>
See also	<code>puppetSprite</code> , <code>spriteNum</code>

sprite...intersects

Syntax

```
sprite(sprite1).intersects(sprite2)  
sprite sprite1 intersects sprite2
```

Description

Keyword; operator that compares the position of two sprites to determine whether the quad of *sprite1* touches (TRUE) or does not touch (FALSE) the quad of *sprite2*.

If both sprites have matte ink, their actual outlines, not the quads, are used. A sprite's outline is defined by the nonwhite pixels that make up its border.

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

Note: The dot operator is required whenever *sprite1* is not a simple expression—that is, one that contains a math operation.

Example

This statement checks whether two sprites intersect and, if they do, changes the contents of the field cast member Notice to "You placed it correctly."

```
if sprite i intersects j then put "You placed it correctly." into member "Notice"
```

See also

sprite...within, quad

sprite...within

Syntax

```
sprite(sprite1).within(sprite2)  
sprite sprite1 within sprite2
```

Description

Keyword; operator that compares the position of two sprites and determines whether the quad of *sprite1* is entirely inside the quad of *sprite2* (TRUE) or not (FALSE).

If both sprites have matte ink, their actual outlines, not the quads, are used. A sprite's outline is defined by the nonwhite pixels that make up its border.

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

Note: The dot operator is required whenever *sprite1* is not a simple expression—that is, one that contains a math operation.

Example

This statement checks whether two sprites intersect and calls the handler `dolinside` if they do:

```
if sprite(3).within(2) then dolinside
```

See also

sprite...intersects, quad

spriteNum

Syntax

```
spriteNum  
the spriteNum of me
```

Description

Sprite property; determines the channel number the behavior's sprite is in and makes it available to any behaviors. Simply declare the property at the top of the behavior, along with any other properties the behavior may use.

If you use an on new handler to create an instance of the behavior, the script's on new handler must explicitly set the spriteNum property to the sprite's number. This provides a way to identify the sprite the script is attached to. The sprite's number must be passed to the on new handler as an argument when the on new handler is called.

Example

In this handler, the spriteNum property is automatically set for script instances that are created by the system:

```
property spriteNum  
  
on mouseDown me  
    sprite(spriteNum).member = member("DownPict")  
end
```

Example

This handler uses the automatic value inserted into the spriteNum property to assign the sprite reference to a new property variable pMySpriteRef, as a convenience:

```
property spriteNum, pMySpriteRef  
  
on beginSprite me  
    pMySpriteRef = sprite(me.spriteNum)  
end
```

This approach allows the use of the reference pMySpriteRef later in the script, with the handler using the syntax

```
currMember = pMySpriteRef.member  
instead of the following syntax which is somewhat longer:  
currMember = sprite(spriteNum).member
```

This alternative approach is merely for convenience, and provides no different functionality.

See also

on beginSprite, on endSprite, currentSpriteNum

sqrt()

Syntax	<code>sqrt(number)</code>
	the sqrt of <i>number</i>
Description	Math function; returns the square root of the number specified by <i>number</i> , which is either a floating-point number or an integer rounded to the nearest integer. The value must be a decimal number greater than 0. Negative values return 0.
Example	This statement displays the square root of 3.0 in the Message window: <code>put sqrt(3.0)</code> -- 1.7321
Example	This statement displays the square root of 3 in the Message window: <code>put sqrt(3)</code> -- 2
See also	floatPrecision

stage

Syntax	the stage
Description	System property; refers to the main movie. This property is useful when using the tell command to send a message to the main movie from a child movie.
Example	This statement causes the main Stage movie to go to the marker named Menu. This statement can be used in a movie in a window (MIAW): <code>tell the Stage to go to "Menu"</code>
Example	This statement displays the current setting for the Stage: <code>put the stage.rect</code> --rect (0, 0, 640, 480)

stageBottom

Syntax	the stageBottom
Description	Function; along with stageLeft, stageRight, and stageTop, indicates where the Stage is positioned on the desktop. It returns the bottom vertical coordinate of the Stage relative to the upper left corner of the main screen. The height of the Stage in pixels is determined by the stageBottom - the stageTop. When the movie plays back as an applet, the stageBottom property is the height of the applet in pixels.

This function can be tested but not set.

Example These statements position sprite 3 a distance of 50 pixels from the bottom edge of the Stage:

See also stageHeight = the stageBottom - the stageTop

```
sprite(3).locV = stageHeight - 50
```

Sprite coordinates are expressed relative to the upper left corner of the Stage. See *Using Director* for more information.

See also stageLeft, stageRight, stageTop, locH, locV

stageColor

Syntax the stageColor

Description System property; determines the color of the movie background for index color only.

Use bgColor for more accurate, reliable, and flexible stage color specification with RGB values.

The value of the stageColor property ranges from 0 to 255 for 8-bit index color, and from 0 to 15 for 4-bit color. You can click a color in the color palette to see that color's index number in the lower left corner of the window. Setting the stageColor property in a Lingo script is equivalent to choosing the Stage color from the pop-up palette in the panel window.

Note: For compatibility when playing back as a Java applet, use the bgColor property to define the color as an RGB value.

Example This statement sets the variable oldColor to the index number of the current Stage color:

Related Lingo oldColor = the stageColor

This statement sets the Stage color to the color assigned to chip 249 on the current palette:

Related Functions the stageColor = 249

See also bgColor, foreColor, color()

stageLeft

Syntax the stageLeft

Description Function; along with stageRight, stageTop, and stageBottom, indicates where the Stage is positioned on the desktop. It returns the left horizontal coordinate of the Stage relative to the upper left corner of the main screen. When the Stage is flush with the left side of the main screen, this coordinate is 0.

When the movie plays back as an applet, the stageLeft property is 0, which is the location of the left side of the applet.

This property can be tested but not set.

Sprite coordinates are expressed relative to the upper left corner of the Stage.

Example This statement checks whether the left edge of the Stage is beyond the left edge of the screen and calls the handler leftMonitorProcedure if it is:

```
if the stageLeft < 0 then leftMonitorProcedure
```

See also stageBottom, stageRight, stageTop, locH, locV

stageRight

Syntax the stageRight

Description Function; along with stageLeft, stageTop, and stageBottom, indicates where the Stage is positioned on the desktop. It returns the right horizontal coordinate of the Stage relative to the upper left corner of the main screen's desktop. The width of the Stage in pixels is determined by the stageRight - the stageLeft.

When the movie plays back as an applet, the stageRight property is the width of the applet in pixels.

This function can be tested but not set.

Sprite coordinates are expressed relative to the upper left corner of the Stage.

Example These two statements position sprite 3 a distance of 50 pixels from the right edge of the Stage:

```
stageWidth = the stageRight - the stageLeft  
sprite(3).locH = stageWidth - 50
```

See also stageBottom, stageLeft, stageTop, locH, locV

stageToFlash()

Syntax

```
sprite(whichFlashSprite).stageToFlash(pointOnDirectorStage)
```

```
stageToFlash (sprite whichFlashSprite, pointOnDirectorStage)
```

Description

Function; returns the coordinate in a Flash movie sprite that corresponds to a specified coordinate on the Director Stage. The function both accepts the Director Stage coordinate and returns the Flash movie coordinate as Director point values: for example, point (300,300).

Flash movie coordinates are measured in Flash movie pixels, which are determined by the original size of the movie when it was created in Flash. Point (0,0) of a Flash movie is always at its upper left corner. (The cast member's `originPoint` property is not used to calculate movie coordinates; it is used only for rotation and scaling.)

The `stageToFlash()` function and the corresponding `flashToStage()` function are helpful for determining which Flash movie coordinate is directly over a Director Stage coordinate. For both Flash and Director, point (0,0) is the upper left corner of the Flash Stage or Director Stage. These coordinates may not match on the Director Stage if a Flash sprite is stretched, scaled, or rotated.

Example

This handler checks to see if the mouse (whose location is tracked in Director Stage coordinates) is over a specific coordinate (130,10) in a Flash movie sprite in channel 5. If the mouse is over that Flash movie coordinate, the script stops the Flash movie.

```
on checkFlashRollover
    if sprite(5).stageToFlash(point(the mouseH,the mouseV)) = point(130,10) then
        sprite(5).stop()
    end if
end
```

See also

[flashToStage\(\)](#)

stageTop

Syntax the stageTop

Description Function; along with stageBottom, stageLeft, and stageRight, indicates where the Stage is positioned on the desktop. It returns the top vertical coordinate of the Stage relative to the upper left corner of the main screen's desktop. If the Stage is in the upper left corner of the main screen, this coordinate is 0.

When the movie plays back as an applet, the stageTop property is always 0, which is the location of the left side of the applet.

This function can be tested but not set.

Sprite coordinates are expressed relative to the upper left corner of the Stage.

Example This statement checks whether the top of the Stage is beyond the top of the screen and calls the handler upperMonitorProcedure if it is:

```
if the stageTop < 0 then upperMonitorProcedure
```

See also stageBottom, stageLeft, stageRight, locH, locV

startFrame

Syntax sprite(*whichSprite*).startFrame

Description Function; returns the frame number of the starting frame of the sprite span.

This function is useful in determining the span in the Score that a particular sprite covers. It is available only in a frame that contains the sprite, and cannot be applied to sprites in different frames of the movie, nor is it possible to set this value.

Example This statement displays the starting frame of the sprite in channel 5 in the Message window:

```
put sprite(5).startFrame
```

See also endFrame()

on startMovie

Syntax on startMovie

```
    statement(s)  
end
```

Description System message and event handler; contains statements that run just before the playback head enters the first frame of the movie. The startMovie event occurs after the prepareFrame event and before the enterFrame event.

An on startMovie handler is a good place to put Lingo that initializes sprites in the first frame of the movie.

Example This handler makes sprites invisible when the movie starts:

```
on startMovie  
    repeat with counter = 10 to 50  
        sprite(counter).visible = 0  
    end repeat  
end startMovie
```

See also on prepareMovie

starts

Syntax *string1* starts *string2*

Description Operator; compares to determine whether *string1* starts with *string2* (TRUE or 1) or not (FALSE or 0).

The string comparison is not sensitive to case or diacritical marks; *a* and *À* are considered to be the same.

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

Example This statement reports in the Message window whether the word *Macromedia* starts with the string “Macro”:

```
put "Macromedia" starts "Macro"
```

The result is 1, which is the numerical equivalent of TRUE.

See also contains

startTime

Syntax	<code>sprite(whichSprite).startTime</code> the startTime of sprite <i>whichSprite</i> <code>sound(channelNum).startTime</code>
Description	Sprite and sound property; for digital video sprites, determines when the specified digital video sprite begins. The value of startTime is measured in ticks. For digital video sprites, this property can be tested and set. Set the startTime before the digital video member begins playback. For sound channels, this property indicates the start time of the currently playing or paused sound as set when the sound was queued. It cannot be set after the sound has been queued. If no value was supplied when the sound was queued, this property returns zero.
Example	This statement starts the digital video sprite in channel 5 at 100 ticks into the digital video: <code>sprite(5).startTime = 100</code>
See also	<code>queue()</code> , <code>setPlayList()</code> , <code>play()</code>

startTimer

Syntax	<code>startTimer</code>
Description	Command; sets the timer property to 0 and resets all the accumulating timers for the <code>lastClick()</code> , <code>lastEvent()</code> , <code>lastKey</code> , and <code>lastRoll</code> functions to 0. If multiple timers are required, you must create and manage your own. These can be properties in a behavior, a global list, or even a parent script. Typically, you use the ticks property to track time in this manner.
Example	This handler sets the timer to 0 when a key is pressed: <code>on keyDown startTimer end</code>
See also	<code>lastClick()</code> , <code>lastEvent()</code> , <code>lastKey</code> , <code>lastRoll</code> ; <code>timeoutLength</code> , <code>timeoutMouse</code> , <code>timeoutPlay</code> , <code>timeoutScript</code> , <code>timer</code>

state

Syntax

member(*whichCastMember*).state

state of member *whichCastMember*

Description

Cast member property; for Shockwave Audio (SWA) streaming cast members and Flash movie cast members, determines the current state of the streaming file. The properties *streamName*, *URL*, and *preLoadTime* can be changed only when the SWA sound is stopped.

The following properties for the SWA file return meaningful information only after the file begins streaming: *cuePointNames*, *cuePointTimes*, *currentTime*, *duration*, *percentPlayed*, *percentStreamed*, *bitRate*, *sampleRate*, and *numChannels*.

For SWA streaming cast members, the following values are possible:

- 0—Cast streaming has stopped.
- 1—The cast member is reloading.
- 2—Preloading ended successfully.
- 3—The cast member is playing.
- 4—The cast member is paused.
- 5—The cast member has finished streaming.
- 9—An error occurred.
- 10—There is insufficient CPU space.

For Flash movie cast members, this property returns a valid value only when the Director movie is running. The following values are possible:

- 0—The cast member is not in memory.
- 1—The header is currently loading.
- 2—The header has finished loading.
- 3—The cast member's media is currently loading.
- 4—The cast member's media has finished loading.
- -1—An error occurred.

This property can be tested but not set.

Example

This statement issues an alert if an error is detected for the SWA streaming cast member:

```
on mouseDown
    if member("Ella Fitzgerald").state = 9 then
        alert "Sorry, can't find an audio file to stream."
    end if
end
```

Example This frame script checks to see if a Flash movie cast member named Intro Movie has finished streaming into memory. If it hasn't, the script reports in the Message window the current state of the cast member and keeps the playback head looping in the current frame until the movie finishes loading into memory.

```
on exitFrame
    if member("Intro Movie").percentStreamed < 100 then
        put "Current download state:" && member("Intro Movie").state
        go the frame
    end if
end
```

See also [clearError](#), [getError\(\)](#)

static

Syntax `sprite(whichFlashSprite).static`

the static of sprite *whichFlashSprite*

`member(whichFlashMember).static`

the static of member *whichFlashMember*

Description Cast member property and sprite property; controls playback performance of a Flash movie sprite depending on whether the movie contains animation. If the movie contains animation (FALSE, default), the property redraws the sprite for each frame; if the movie doesn't contain animation (TRUE), the property redraws the sprite only when it moves or changes size.

This property can be tested and set.

Note: Set the static property to TRUE only when the Flash movie sprite does not intersect other moving Director sprites. If the Flash movie intersects moving Director sprites, it may not redraw correctly.

Example This sprite script displays in the Message window the channel number of a Flash movie sprite and indicates whether the Flash movie contains animation:

```
property spriteNum

on beginSprite me
    if sprite(spriteNum).static then
        animationType = "does not have animation."
    else
        animationType = "has animation."
    end if
    put "The Flash movie in channel" && spriteNum && animationType
end
```

staticQuality

Syntax staticQuality of sprite *whichQTVRSprite*

Description QuickTime VR sprite property; specifies the codec quality used when the panorama image is static. Possible values are #minQuality, #maxQuality, and #normalQuality.

This property can be tested and set.

status

Syntax *soundObject*.status
the status of *soundObject*

Description Read-only property indicates the status of sound channel *channelNum*. Possible values include:

Status	Name	Meaning
0	Idle	No sounds are queued or playing.
1	Loading	A queued sound is being preloaded but is not yet playing.
2	Queued	The sound channel has finished preloading a queued sound but is not yet playing the sound.
3	Playing	A sound is playing.
4	Paused	A sound is paused.

Example This statement displays the current status of sound channel 2 in the Message window:

```
put sound(2).status
```

See also [isBusy\(\)](#), [pause\(\)](#) (sound playback), [play\(\)](#) (sound), [stop\(\)](#) (sound)

on stepFrame

Syntax

```
on stepFrame  
    statement(s)  
end
```

Description System message and event handler; works in script instances in actorList because these are the only objects that receive on stepFrame messages. This event handler is executed when the playback head enters a frame or the Stage is updated.

An on stepFrame handler is a useful location for Lingo that you want to run frequently for a specific set of objects. Assign the objects to actorList when you want Lingo in the on stepFrame handler to run; remove the objects from actorList to prevent Lingo from running. While the objects are in actorList, the objects' on stepFrame handlers run each time the playback head enters a frame or the updateStage command is issued.

The stepFrame message is sent before the prepareFrame message.

Assign objects to actorList so they respond to stepFrame messages. Objects must have an on stepFrame handler to use this built-in functionality with actorList.

The go, play, and updateStage commands are disabled in an on stepFrame handler.

Example If the child object is assigned to actorList, the on stepFrame handler in this parent script updates the position of the sprite that is stored in the mySprite property each time the playback head enters a frame:

```
property mySprite  
  
on new me, theSprite  
    mySprite = theSprite  
    return me  
end  
  
on stepFrame me  
    sprite(mySprite).loc = point(random(640),random(480))  
end
```

stillDown

Syntax

the stillDown

Description System property; indicates whether the user is pressing the mouse button (TRUE) or not (FALSE).

This function is useful within a mouseDown script to trigger certain events only after the mouseUp function.

Lingo cannot test stillDown when it is used inside a repeat loop. Use the mouseDown function inside repeat loops instead.

Example	This statement checks whether the mouse button is being pressed and calls the handler dragProcedure if it is: if the stillDown then dragProcedure
See also	the mouseDown (system property)

stop (Flash)

Syntax	sprite(<i>whichFlashSprite</i>).stop() stop sprite <i>whichFlashSprite</i>
Description	Flash command; stops a Flash movie sprite that is playing in the current frame.
Example	This frame script stops the Flash movie sprites playing in channels 5 through 10: on enterFrame repeat with i = 5 to 10 sprite(i).stop() end repeat end

See also hold

stop() (sound)

Syntax	sound(<i>channel/Num</i>).stop() stop(sound(<i>channel/Num</i>))
Description	Command; stops the currently playing sound in sound channel <i>channelNum</i> . Issuing a play() command begins playing the first sound of those that remain in the queue of the given sound channel. To see an example of stop() (sound) used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.
Example	This statement stops playback of the sound cast member currently playing in sound channel 1. sound(1).stop()
See also	getPlaylist(), pause() (sound playback), play() (sound), playNext(), rewind()

stopEvent

Syntax

stopEvent

Description

Command; prevents Lingo from passing an event message to subsequent locations in the message hierarchy. Equivalent to the dontPassEvent command used in earlier versions of Director, this command also applies to sprite scripts.

Use the stopEvent command to stop the message in a primary event handler or a sprite script, thus making the message unavailable for subsequent sprite scripts.

By default, messages are available first to a primary event handler (if one exists) and then to any scripts attached to a sprite involved in the event. If more than one script is attached to the sprite, the message is available to each of the sprite's scripts. If no sprite script responds to the message, the message passes to a cast member script, frame script, and movie script, in that order.

The stopEvent command applies only to the current event being handled. It does not affect future events. The stopEvent command applies only within primary event handlers, handlers that primary event handlers call, or multiple sprite scripts. It has no effect elsewhere.

Example

This statement shows the mouseUp event being stopped in a behavior if the global variable grandTotal is equal to 500:

```
global grandTotal  
  
on mouseUp me  
    if grandTotal = 500 then  
        stopEvent  
    end if  
end
```

Neither subsequent scripts nor other behaviors on the sprite receive the event if it is stopped in this manner.

See also

pass

stop member

Syntax

member (*whichCastMember*).stop()

stop member (*whichCastMember*)

Description

Command; stops the playback of a Shockwave Audio (SWA) streaming cast member. When the cast member is stopped, the state member property equals 0.

For you to change properties such as streamName, preLoadTime, and URL, the SWA streaming cast member must be stopped.

Example

This statement stops the SWA cast member Big Band from playing:

```
member("Big Band").stop()
```

See also

play member, pause member

on stopMovie

Syntax on stopMovie

```
    statement(s)  
end
```

Description System message and event handler; contains statements that run when the movie stops playing.

An on stopMovie handler is a good place to put Lingo that performs cleanup tasks—such as closing resource files, clearing global variables, erasing fields, and disposing of objects—when the movie is finished.

An on stopMovie handler in a MIAW is called only when the movie plays through to the end or branches to another movie. It isn't called when the window is closed or when the window is deleted by the forget window command.

Example This handler clears global variables and closes two resource files when the movie stops:

```
global gCurrentScore  
on stopMovie  
    set gCurrentScore = 0  
    closeResFile "Special Fonts"  
    closeResFile "Special Cursors"  
end
```

See also on prepareMovie

stopTime

Syntax sprite(*whichSprite*).stopTime

the stopTime of sprite *whichSprite*

Description Sprite property; determines when the specified digital video sprite stops. The value of stopTime is measured in ticks.

This property can be tested and set.

Example This statement stops the digital video sprite in channel 5 at 100 ticks into the digital video:

```
sprite(5).stopTime = 100
```

stream

Syntax

```
member(whichFlashSprite).stream(numberOfBytes )  
stream(member whichFlashSprite, numberOfBytes)
```

Description

Command; manually streams a portion of a specified Flash movie cast member into memory. You can optionally specify the number of bytes to stream as an integer value. If you omit the *numberOfBytes* parameter, Director tries to stream the number of bytes set by the cast member's *bufferSize* property.

The *stream* command returns the number of bytes actually streamed. Depending on a variety of conditions (such as network speed or the availability of the requested data), the number of bytes actually streamed may be less than the number of bytes requested.

You can always use the *stream* command for a cast member regardless of the cast member's *streamMode* property.

Example

This frame script checks to see if a linked Flash movie cast member has streamed into memory by checking its *percentStreamed* property. If the cast member is not completely loaded into memory, the script tries to stream 32,000 bytes of the movie into memory.

The script also saves the actual number of bytes streamed in a variable called *bytesReceived*. If the number of bytes actually streamed does not match the number of bytes requested, the script updates a text cast member to report the number of bytes actually received. The script keeps the playback head looping in the current frame until the cast member has finished loading into memory.

```
on exitFrame  
    if member(10).percentStreamed < 100 then  
        bytesReceived = member(10).stream(32000)  
        if bytesReceived < 32000 then  
            member("Message Line").text = "Received only" && bytesReceived \  
                && " of 32,000 bytes requested."  
            updateStage  
        else  
            member("Message Line").text = "Received all 32,000 bytes."  
        end if  
        go the frame  
    end if  
end
```

streaming

Syntax	<code>member(<i>whichMember</i>).streaming</code> the streaming of member <i>whichMember</i>
Description	QuickTime cast member property. When TRUE, allows QuickTime playing over the Internet to begin playing immediately while the member downloads to the user's computer. When FALSE, the member must download completely before playback will begin. Defaults to TRUE for Director movies made with Director 7.02 and later. Defaults to FALSE for movies made with earlier versions of Director. If the QuickTime member contains a text track with cue points, the text track must be set to preload in order for Director to make use of the cue points. You set the text track to preload using a video editor.
Example	This statement sets the streaming of member SunriseVideo to FALSE, causing it to download completely before playing back in Shockwave or ShockMachine: <code>member("SunriseVideo").streaming = 0</code>

streamMode

Syntax	<code>member(<i>whichFlashMember</i>).streamMode</code> the streamMode of member <i>whichFlashMember</i>
Description	Flash cast member property; controls the way a linked Flash movie cast member is streamed into memory, as follows: <ul style="list-style-type: none">• #frame (default)—Streams part of the cast member each time the Director frame advances while the sprite is on the Stage.• #idle—Streams part of the cast member each time an idle event is generated or at least once per Director frame while the sprite is on the Stage.• #manual—Streams part of the cast member into memory only when the stream command is issued for that cast member.
	This property can be tested and set.
Example	This startMovie script searches the internal cast for Flash movie cast members and sets their streamMode properties to #manual: <code>on startMovie repeat with i = 1 to the number of members of castLib 1 if member(i, 1).type = #flash then member(i, 1).streamMode = #manual end if end repeat end</code>

streamName

Syntax	<code>member(<i>whichCastMember</i>).streamName</code> the streamName of member <i>whichCastMember</i>
Description	Shockwave Audio (SWA) cast member property; specifies a URL or file name for a streaming cast member. This property functions the same as the URL member property.
	This property can be tested and set.

Example

This statement links the file BigBand.swa to an SWA streaming cast member. The linked file is on the disk MyDisk in the folder named Sounds.

```
member("SWAstream").streamName = "MyDisk/sounds/BigBand.swa"
```

streamSize

Syntax	<code>member(<i>whichFlashMember</i>).streamSize</code> the streamSize of member <i>whichFlashMember</i>
Description	Cast member property; reports an integer value indicating the total number of bytes in the stream for the specified cast member. The streamSize property returns a value only when the Director movie is playing.
	This property can be tested but not set.

Example

This frame script checks to see if a Flash movie cast member named Intro Movie has finished streaming into memory. If it hasn't, the script updates a field cast member to indicate the number of bytes that have been streamed (using the bytesStreamed member property) and the total number of bytes for the cast member (using the streamSize member property). The script keeps the playback head looping in the current frame until the movie finishes loading into memory.

```
on exitFrame
    if member("Intro Movie").percentStreamed < 100 then
        member("Message Line").text = string(member("Intro Movie").bytesStreamed) \
            && "of" && string(member("Intro Movie").streamSize) \
            &&"bytes have downloaded so far."
        go to the frame
    end if
end
```

on streamStatus

Syntax

```
on streamStatus URL, state, bytesSoFar, bytesTotal, error  
    statement(s)  
end
```

Description System message and event handler; called periodically to determine how much of an object has been downloaded from the Internet. The handler is called only if tellStreamStatus (TRUE) has been called, and the handler has been added to a movie script.

The on streamStatus event handler has the following parameters:

<i>URL</i>	Displays the Internet address of the data being retrieved.
<i>state</i>	Displays the state of the stream being downloaded. Possible values are Connecting, Started, InProgress, Complete, and Error.
<i>bytesSoFar</i>	Displays the number of bytes retrieved from the network so far.
<i>bytesTotal</i>	Displays the total number of bytes in the stream, if known. The value may be 0 if the HTTP server does not include the content length in the MIME header.
<i>error</i>	Displays an empty string ("") if the download has not finished; OK (OK) if the download completed successfully; displays an error code if the download was unsuccessful.

These parameters are automatically filled in by Director with information regarding the progress of the download. The handler is called by Director automatically, and there is no way to control when the next call will be. If information regarding a particular operation is needed, call getStreamStatus().

You can initiate network streams using Lingo commands, by linking media from a URL, or by using an external cast member from a URL. A streamStatus handler will be called with information about all network streams.

Place the streamStatus handler in a movie script.

Example This handler determines the state of a streamed object and displays the URL of the object:

```
on streamStatus URL, state, bytesSoFar, bytesTotal  
    if state = "Complete" then  
        put URL && "download finished"  
    end if  
end streamStatus
```

See also [getStreamStatus\(\)](#), [tellStreamStatus\(\)](#)

string()

Syntax string(*expression*)

Description Function; converts an integer, floating-point number, object reference, list, symbol, or other nonstring expression to a string.

Example This statement adds 2.0 + 2.5 and inserts the results in the field cast member Total:

```
member("total").text = string(2.0 + 2.5)
```

Example This statement converts the symbol #red to a string and inserts it in the field cast member Color:

```
member("Color").text = string(#red)
```

See also value(), stringP(), float(), integer(), symbol()

stringP()

Syntax stringP(*expression*)

Description Function; determines whether an expression is a string (TRUE) or not (FALSE).

The *P* in stringP stands for *predicate*.

Example This statement checks whether 3 is a string:

```
put stringP("3")
```

The result is 1, which is the numeric equivalent of TRUE.

Example This statement checks whether the floating-point number 3.0 is a string:

```
put stringP(3.0)
```

Because 3.0 is a floating-point number and not a string, the result is 0, which is the numeric equivalent of FALSE.

See also floatP(), ilk(), integerP(), objectP(), symbolP()

strokeColor

Syntax member(*whichCastMember*).strokeColor

Description Vector shape cast member property; indicates the color in RGB of the shape's framing stroke.

To see an example of strokeColor used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.

Example	This sets the strokeColor of cast member "line" to red.
	member("line").strokeColor = rgb(255, 0, 0)
See also	color(), fillColor, endColor, backgroundColor

strokeWidth

Syntax	member(<i>whichCastMember</i>).strokeWidth
Description	Vector shape cast member property; indicates the width, in pixels, of the shape's framing stroke. The value is a floating point number between 0 and 100, and can be tested and set. To see an example of strokeWidth used in a completed movie, see the Vector Shapes movie in the Learning\Lingo Examples folder inside the Director application folder.
Example	This sets the strokeWidth of cast member "line" to 10 pixels.
	member(1).strokeWidth = 10

substituteFont

Syntax	TextMemberRef.substituteFont(<i>originalFont</i> , <i>newFont</i>) substituteFont(<i>textMemberRef</i> , <i>originalFont</i> , <i>newFont</i>)
Description	Text cast member command; replaces all instances of <i>originalFont</i> with <i>newFont</i> in <i>textMemberRef</i> .
Example	This script checks to see if the font Bonneville is available in a text cast member, and replaces it with Arial if it is not: property spriteNum on beginSprite me currMember = sprite(spriteNum).member if currMember.missingFonts contains "Bonneville" then currMember.substituteFont("Bonneville", "Arial") end if end
See also	missingFonts

swing()

Syntax

WhichQTVRSprite.swing(*pan*, *tilt*, *fieldOfView*, *speedToSwing*)

swing (whichQTVRSprite, *pan*, *tilt*, *fieldOfView*, *speedToSwing*)

Description

QuickTime VR sprite function; swings a QuickTime 3 sprite containing a VR Pano around to the new view settings. The swing is a smooth “camera dolly” effect.

- *whichQTVRSprite*—Sprite number of the sprite with the QuickTime VR member.
- *pan*—New pan position, in degrees.
- *tilt*—New tilt, in degrees.
- *fieldOfView*—New field of view, in degrees.
- *speedToSwing*—Rate at which the swing should take place; specify an integer from 1 to 10 (slow to fast).

The function returns immediately, but the sprite continues to change view until it reaches the final view. The duration required to change to the final settings varies depending on machine type, size of the sprite rectangle, color depth of the screen, and other typical performance constraints.

To check if the swing has finished, check if the *pan* property of the sprite has arrived at the final value.

Example

This very gradually adjusts the view of QTVR sprite 1 to a pan position of 300 degrees, a tilt of -15 degrees, and a field of view of 40 degrees.

```
sprite(1).swing(300, -15, 40, 1)
```

See also

[pan \(QTVR property\)](#)

switchColorDepth

Syntax

the switchColorDepth

Description

System property; determines whether Director switches the monitor that the Stage occupies to the color depth of the movie being loaded (TRUE) or leaves the color depth of the monitor unchanged when a movie is loaded (FALSE). False is the default value.

When *switchColorDepth* is TRUE, nothing happens until a new movie is loaded.

Setting the monitor's color depth to that of the movie is good practice.

- When the monitor's color depth is set below that of the movie, resetting it to the color depth of the movie (assuming that the monitor can provide that color depth) helps maintain the movie's original appearance.
- When the monitor's color depth is higher than that of the movie, reducing the monitor's color depth plays the movie using the minimum amount of memory, loads cast members more efficiently, and causes animation to occur more quickly.

This property can be tested and set. The default value comes from the Reset Monitor to Movie's Color Depth option in the General Preferences dialog box.

Example This statement sets the variable named switcher to the current setting of switchColorDepth:

```
switcher = the switchColorDepth
```

Example This statement checks whether the current color depth is 8-bit and turns the switchColorDepth property on if it is:

```
if the colorDepth = 8 then the switchColorDepth = TRUE
```

See also colorDepth

symbol()

Syntax
stringValue.symbol
symbol(stringValue)

Description Function; takes a string, *stringValue*, and returns a symbol.

Example This statement displays the symbol #hello:

```
put ("hello").symbol  
-- #hello
```

Example This statement displays the symbol #goodbye:

```
x = "goodbye"  
put x.symbol  
-- #goodbye
```

See also value(), string()

symbolP()

Syntax	Expression.symbolP symbolP(expression)
Description	Function; determines whether the expression specified by <i>expression</i> is a symbol (TRUE) or not (FALSE). The <i>P</i> in symbolP stands for <i>predicate</i> .
Example	This statement checks whether the variable myVariable is a symbol: put myVariable.symbolP
See also	ilk()

systemDate

Syntax	the systemDate
Description	System property; returns the current date in a standard date format and can be used in conjunction with other date operations for international and cross-platform date manipulation. Math operations on the date are performed in days. This property can be tested but not set.
Example	This script displays the current date being retrieved and then determines the date 90 days from the current date: on ShowEndOfTrialPeriodDate set today = the systemDate set trialEndDate = today + 90 put "The trial period for this software is over on"&&trialEndDate end
See also	date() (system clock)

TAB

Syntax	TAB
Description	Constant; represents the Tab key.
Example	This statement checks whether the character typed is the tab character and calls the handler doNextField if it is: if the key = TAB then doNextField

Example These statements move the playback head forward or backward, depending on whether the user presses Tab or Shift-Tab:

```
if the key = TAB then
    if the shiftDown then
        go the frame -1
    else
        go the frame +1
    end if
end if
```

See also BACKSPACE, EMPTY, RETURN (constant)

tabCount

Syntax *chunkExpression.tabCount*

Description Text cast member property; indicates how many unique tab stops are in the specified chunk expression of the text cast member.

The value is an integer equal to or greater than 0, and may be tested but not set.

tabs

Syntax *member(whichTextMember).tabs*

Description Text cast member property; this property contains a list of all the tab stops set in the text cast member.

Each element of the list contains information regarding that tab for the text cast member. The possible properties in the list are as follows:

#type	Can be #left, #center, #right, or #decimal.
#position	Integer value indicating the position of the tab in points.

You can get and set this property. When tabs is set, the type property is optional. If type is not specified, the tab type defaults to #left.

Example This statement retrieves and displays in the Message window all the tabs for the text cast member Intro credits:

```
put member("Intro credits").tabs
-- [[#type: #left, #position: 36], [#type: #Decimal, #position: 141], \
[#type: #right, #position: 216]]
```

tan()

Syntax	<code>tan(<i>angle</i>)</code>
Description	Math function; yields the tangent of the specified angle expressed in radians as a floating-point number.
Example	The following function yields the tangent of pi/4: <code>tan (PI/4.0) = 1</code> The π symbol cannot be used in a Lingo expression.
See also	<code>PI</code>

target

Syntax	<code><i>timeoutObject</i>.target</code>
Description	Timeout object property; indicates the child object that the given <i>timeoutObject</i> will send its timeout events to. Timeout objects whose target property is VOID will send their events to a handler in a movie script. This property is useful for debugging behaviors and parent scripts that use timeout objects.
Example	This statement displays the name of the child object that will receive timeout events from the timeout object timerOne in the Message window: <code>put timeout("timerOne").target</code>
See also	<code>name</code> (timeout property), <code>timeout()</code> , <code>timeoutHandler</code> , <code>timeoutList</code>

tell

Syntax	<code>tell <i>whichWindow</i> to <i>statement(s)</i></code> <code>tell <i>whichWindow</i> <i>statement(s)</i> end</code>
Description	Command; communicates statements to the window specified by <i>whichWindow</i> . The tell command is useful for allowing movies to interact. It can be used within a main movie to send a message to a movie playing in a window, or to send a message from a movie playing in a window to the main movie. For example, the tell command can let a button in a control panel call a handler in a movie playing in a window. The movie playing in a window may react to the first movie handler by executing the handler. The movie playing in the window may interact with the main movie by sending a value back to the movie.

When you use the tell command to send a message to a movie playing in a window, identify the window object by using the full pathname or its number in windowList. If you use windowList, use the expression getAt(the windowList, *windowNum*), where *windowNum* is a variable that contains the number of the window's position in the list. Because the opening and closing of windows may change the order of windowList, it is a good idea to store the full pathname as a global variable when referencing windows with getAt in windowList.

Example A multiple-line tell command resembles a handler and requires an end tell statement:

```
global childMovie
```

```
tell window childMovie
    go to frame "Intro"
    the stageColor = 100
    sprite(4).member = member "Diana Ross"
    updateStage
end tell
```

Example When a message calls a handler, a value returned from the handler can be found in the global result property after the called handler is done. These statements send the childMovie window the message calcBalance and then return the result:

```
global childMovie
```

```
tell window childMovie to calcBalance
-- a handler name
myBalance = result()
-- return value from calcBalance handler
```

Example When you use the tell command to send a message from a movie playing in a window to the main movie, use the stage system property as the object name:

```
tell the stage to go frame "Main Menu"
```

When you use the tell command to call a handler in another movie, make sure that you do not have a handler by the same name in the same script in the local movie. If you do, the local script will be called. This restriction applies only to handlers in the same script in which you are using the tell command.

Example This statement causes the Control Panel window to instruct the Simulation movie to branch to another frame:

```
tell window "Simulation" to go frame "Save"
```

tellStreamStatus()

Syntax tellStreamStatus(*onOrOffBoolean*)

Description Function; turns the stream status handler on (TRUE) or off (FALSE).

The form tellStreamStatus() determines the status of the handler.

When the streamStatusHandler is TRUE, Internet streaming activity causes periodic calls to the movie script, triggering streamStatusHandler. The handler is executed, with Director automatically filling in the parameters with information regarding the progress of the downloads.

Example This on prepareMovie handler turns the on streamStatus handler on when the movie starts:

```
on prepareMovie  
    tellStreamStatus(TRUE)  
end
```

Example This statement determines the status of the stream status handler:

```
on mouseDown  
    put tellStreamStatus()  
end
```

See also on streamStatus

text

Syntax member(*whichCastMember*).text

the text of member *whichCastMember*

Description Text cast member property; determines the character string in the field cast member specified by *whichCastMember*.

The text cast member property is useful for displaying messages and recording what the user types.

This property can be tested and set.

When you use Lingo to change the entire text of a cast member you remove any special formatting you have applied to individual words or lines. Altering the text cast member property reapplies global formatting. To change particular portions of the text, refer to lines, words, or items in the text.

When the movie plays back as an applet, this property's value is "" (an empty string) for a field cast member whose text has not yet streamed in.

To see an example of text used in a completed movie, see the Forms and Post, and Text movies in the Learning\Lingo Examples folder inside the Director application folder.

Example	<p>This statement places the phrase “Thank you.” in the empty cast member Response:</p> <pre>if member("Response").text = EMPTY then member("Response").text = "Thank You."</pre>
	<p>This statement sets the content of cast member Notice to “You have made the right decision!”</p> <pre>member("Notice").text = "You have made the right decision!"</pre>
See also	selEnd, selStart

the

Syntax	the <i>property</i>
Description	<p>Keyword; must precede many functions and all Lingo properties written in verbose syntax. This keyword also distinguishes the property or function from a variable or object name.</p> <p>Earlier versions of Director required you to use the the keyword to express cast member and sprite properties. This syntax is still supported as alternate form.</p> <p>Properties are globally available to handlers even if you don't declare them globally. Like global variables, Lingo system properties are available between different movies in the same presentation. Sprite properties change when a new movie is loaded.</p>

thumbnail

Syntax	<pre>member(<i>whichMember</i>).thumbnail</pre> <p>the thumbnail of member <i>whichMember</i></p>
Description	<p>Cast member property; contains the image used to preview a cast member in the Cast window. This image can be customized for any cast member.</p> <p>This property can be tested and set only during authoring.</p>
Example	<p>This statement shows how to use a placeholder cast member to display another thumbnail on the Stage. The placeholder cast member is placed on the Stage, then the picture of that member is set to the thumbnail of member 10. This makes it possible to show a reduced image without having to scale or otherwise manipulate a graphic:</p> <pre>member("Placeholder").picture = member(10).thumbnail</pre>
See also	picture (cast member property)

ticks

Syntax	the ticks
Description	System property; returns the current time in ticks (1 tick = 1/60 of a second). Counting ticks begins from the time the computer is started.
Example	This statement converts ticks to seconds and minutes by dividing the number of ticks by 60 twice and then sets the variable minutesOn to the result:
	<pre>currentSeconds = the ticks/60 currentMinutes = currentSeconds/60</pre>

See also time(), timer, milliseconds

tilt

Syntax	tilt of sprite (<i>whichQTVRSprite</i>)
Description	QuickTime VR sprite property; the current tilt, in degrees, of the QuickTime VR movie. This property can be tested and set.

time()

Syntax	the time the short time the long time the abbreviated time the abbrev time the abbr time
Description	Function; returns the current time in the system clock as a string in one of three formats: short, long, or abbreviated. If you don't specify a format, the default format is short. The abbreviated format can also be referred to as abbrev and abbr. In the United States, the short and abbreviated formats are the same.
Example	The <code>se</code> statements display the time in different formats in the Message window. Possible results appear below each statement.

```
put the short time  
--"1:30 PM"  
put the long time  
--"1:30:24 PM"  
put the abbreviated time  
--"1:30 PM"
```

The three time formats vary, depending on the individual computer's time format. The preceding examples are for the United States.

See also [date\(\)](#) (system clock)

time (timeout object property)

Syntax `timeoutObject.time`

Description Timeout object property; the system time, in milliseconds, when the next timeout event will be sent by the given `timeoutObject`.

Note that this is not the time until the next event, but the absolute time of the next timeout event.

Example This handler determines the time remaining until the next timeout event will be sent by the timeout object Update by calculating the difference between its time property and the current value of the milliseconds and displaying the result in the field Time Until:

```
on prepareFrame  
    msBeforeUpdate = timeout("Update").time - the milliseconds  
    secondsBeforeUpdate = msBeforeUpdate / 1000  
    minutesBeforeUpdate = secondsBeforeUpdate / 60  
    member("Time Until").text = string(minutesBeforeUpdate) && "minutes before next \\\n        timeout"  
end
```

See also [milliseconds](#), [period](#), [persistent](#), [target](#), [timeout\(\)](#), [timeoutHandler](#)

timeout()

Syntax `timeout("timeoutName")`

Description Function; returns the timeout object named `timeoutName`. Use `timeout("name").new` to add a new timeout object to the `timeoutList`. See `new()`.

Example This handler deletes the timeout object named Random Lightning:

```
on exitFrame  
    timeout("Random Lightning").forget()  
end
```

See also [forget\(\)](#), [new\(\)](#), [timeoutHandler](#), [timeoutList](#)

on timeOut

Syntax on timeOut

```
    statement(s)  
end
```

Description System message and event handler; contains statements that run when the keyboard or mouse is not used for the time period specified in timeoutLength. Always place an on timeOut handler in a movie script.

To have a timeout produce the same response throughout a movie, use the timeoutScript to centrally control timeout behavior.

Example This handler plays the movie Attract Loop after users do nothing for the time set in the timeoutLength property. It can be used to respond when users leave the computer.

```
on timeOut  
    play movie "Attract Loop"  
end timeOut
```

See also timeoutScript, timeoutLength

timeoutHandler

Syntax *timeoutObject*.timeoutHandler

Description System property; represents the name of the handler that will receive timeout messages from the given *timeoutObject*. Its value is a symbol, such as #timeExpiredHandler. The timeoutHandler is always a handler within the timeout object's target object, or in a movie script if the timeout object has no target specified.

This property can be tested and set.

Example This statement displays the timeoutHandler of the timeout object Quiz Timer in the Message window:

```
put timeout("Quiz Timer").timeoutHandler
```

See also target, timeout(), timeoutList

timeoutKeyDown

Syntax the timeoutKeyDown

Description System property; determines whether keyDown events set the timeoutLapsed property to 0 (TRUE, default) or not (FALSE). This property is useful for restarting the countdown for a timeout each time a key is pressed.

This property can be tested and set.

Example This statement sets the variable timing to the value of the timeoutKeyDown property:

```
timing = timeoutKeyDown
```

This statement turns off the timeoutKeyDown property:

```
timeoutKeyDown = FALSE
```

See also keyDownScript

timeoutLapsed

Syntax the timeoutLapsed

Description System property; indicates how many of ticks have elapsed since the last timeout. A timeout event occurs when the timeoutLapsed property reaches the time specified by the timeoutLength property.

The timeoutLapsed property can be tested and set.

Example This statement sets the Countdown member field to the value of the timeoutLapsed property. Dividing timeoutLapsed by 60 converts the value to seconds.

```
member("Countdown").text = string(the timeoutLapsed / 60)
```

timeoutLength

Syntax the timeoutLength

Description System property; determines how many ticks must elapse before a timeout event occurs. A timeout occurs when the timeoutLapsed property reaches the time specified by timeoutLength.

This property can be tested and set. The default value is 10,800 ticks or 3 minutes.

Example This statement sets timeoutLength to 10 seconds:

```
set the timeoutLength to 10 * 60
```

or

```
the timeoutLength = 10 * 60
```

timeoutList

Syntax	the timeoutList
Description	System property; a linear list containing all currently active timeout objects. Use the forget() function to delete a timeout object.
	Timeout objects are added to the timeoutList with the new() function.
Example	This statement deletes the third timeout object from the timeout list: <code>the timeoutList[3].forget()</code>
See also	forget(), new(), timeout(), target, timeoutHandler

timeoutMouse

Syntax	the timeoutMouse
Description	System property; determines whether mouseDown events reset the timeoutLapsed property to 0 (TRUE, default) or not (FALSE). This property can be tested and set.
Example	This statement records the current setting of timeoutMouse by setting the variable named timing to the timeoutMouse: <code>timing = the timeoutMouse</code>
Example	This statement sets the timeoutMouse property to FALSE. The result is that the timeoutLapsed property keeps its current value when the mouse button is pressed. <code>the timeoutMouse = FALSE</code>
See also	mouseDownScript, mouseUpScript

timeoutPlay

Syntax	the timeoutPlay
Description	System property; determines whether the timeoutLapsed property will be set to TRUE when the movie is paused with the pause command. When TRUE, timeouts will occur when the movie is paused. When FALSE, timeouts will not occur when the movie is paused. The default value is FALSE. This property can be tested and set.
Example	This statement sets timeoutPlay to TRUE, which tells Lingo to reset the timeoutLapsed property to 0 after a movie is played: <code>set the timeoutPlay to TRUE</code> <code>or</code> <code>the timeoutPlay = TRUE</code>

timeoutScript

Syntax the timeoutScript

Description System property; determines the Lingo that Director executes as a primary event handler when a timeout occurs. The Lingo is written as a string, surrounded by quotation marks. The default value is EMPTY.

To define a primary event handler for timeouts, set timeoutScript to a string of the appropriate Lingo: either a simple statement or a calling statement for a handler. When the assigned event script is no longer appropriate, turn it off with the statement set the timeoutScript to EMPTY.

This property can be tested and set.

Example This statement sets timeoutScript to a calling script for the handler timeoutProcedure:
the timeoutScript = "timeoutProcedure"

See also on timeOut

timer

Syntax the timer

Description System property; a free running timer that counts time in ticks (1 tick = 1/60 second). It has nothing to do with the timeoutScript property. It is used only for convenience in timing certain events. The startTimer command sets timer to 0.

The timer property is useful for determining the amount of time passed since the startTimer command was issued. For example, you can use timer to synchronize pictures with a soundtrack by inserting a delay that makes the movie wait until a certain amount of time has elapsed.

Example This behavior for a frame script creates a 2-second delay:
on beginSprite
 startTimer
end
on exitFrame
 if (the timer < 60 * 2) then go the frame
end

Example This statement sets the variable startTicks to the current timer value:
startTicks = the timer

See also lastClick(), lastEvent(), lastKey, lastRoll; startTimer

timeScale

Syntax member(*whichCastMember*).timeScale
the timeScale of member *whichCastMember*

Description Cast member property; returns the time unit per second on which the digital video's frames are based. For example, a time unit in a QuickTime digital video is 1/600 of a second.
This property can be tested but not set.

See also digitalVideoTimeScale

title

Syntax window (*whichWindow*.title)
the title of window *whichWindow*

Description Window property; assigns a title to the window specified by *whichWindow*.
This property can be tested and set for windows other than the Stage.

Example This statement makes Action View the title of window X:
`window("X").title = "Action View"`

titleVisible

Syntax window (*whichWindow*.titleVisible)
the titleVisible of window *whichWindow*

Description Window property; specifies whether the window specified by *whichWindow* displays the window title in the window's title bar.
This property can be tested and set for windows other than the Stage.

Example This statement displays the title of the Control Panel window by setting the window's titleVisible property to TRUE:
`window("Control Panel").titleVisible = TRUE`

to

The word **to** occurs in a number of Lingo constructs.

See also char...of, item...of, line...of, word...of, repeat with, set...to, set...=

top

Syntax `sprite(whichSprite).top`

the top of sprite *whichSprite*

Description Sprite property; returns the top vertical coordinate of the bounding rectangle of the sprite specified by *whichSprite* as the number of pixels from the upper left corner of the Stage.

When the movie plays back as an applet, this property's value is relative to the upper left corner of the applet.

This property can be tested and set.

Example This statement checks whether the top of sprite 3 is above the top of the Stage and calls the handler offTopEdge if it is:

```
if sprite(3).top < 0 then offTopEdge
```

See also bottom, height, locH, left, locV, right, width

topSpacing

Syntax `chunkExpression.topSpacing`

Description Text cast member property; allows you to specify additional spacing applied to the top of each paragraph in the *chunkExpression* portion of the text cast member.

The value itself is an integer, with less than 0 indicating less spacing between paragraphs and greater than 0 indicating more spacing between paragraphs.

The default value is 0, which results in default spacing between paragraphs.

Example This statement sets the topSpacing of the second paragraph in text cast member "myText" to 20:

```
member(1).paragraph[2].topSpacing = 20
```

See also bottomSpacing

trace

Syntax	the trace
Description	Movie property; specifies whether the movie's trace function is on (TRUE) or off (FALSE). When the trace function is on, the Message window displays each line of Lingo that is being executed. This property can be tested and set.
Example	This statement turns the trace property on: the trace = TRUE
See also	the traceLogFile

traceLoad

Syntax	the traceLoad
Description	Movie property; specifies the amount of information that is displayed about cast members as they load: <ul style="list-style-type: none">• 0—Displays no information.• 1—Displays cast members' names.• 2—Displays cast members' names, the number of the current frame, the movie name, and the file seek offset (the relative amount the drive had to move to load the media). The default value for the traceLoad property is 0. This property can be tested and set.
Example	This statement causes the movie to display the names of cast members as they are loaded: the traceLoad = 1

traceLogFile

Syntax	the traceLogFile
Description	System property; specifies the name of the file in which the Message window display is written. You can close the file by setting the traceLogFile property to EMPTY (""). Any output that would appear in the Message window is written into this file. You can use this property for debugging when running a movie in a projector and when authoring.
Example	This statement instructs Lingo to write the contents of the Message window in the file "Messages.txt" in the same folder as the current movie: the traceLogFile = the moviePath & "Messages.txt"
Example	This statement closes the file that the Message window display is being written to: the traceLogFile = ""

trackCount (cast member property)

Syntax	member(<i>whichCastMember</i>).trackCount() trackCount(member <i>whichCastMember</i>)
Description	Digital video cast member property; returns the number of tracks in the specified digital video cast member. This property can be tested but not set.
Example	This statement determines the number of tracks in the digital video cast member Jazz Chronicle and displays the result in the Message window: put member("Jazz Chronicle").trackCount()

trackCount (sprite property)

Syntax	sprite(<i>whichDigitalVideoSprite</i>).trackCount() trackCount(sprite <i>whichSprite</i>)
Description	Digital video sprite property; returns the number of tracks in the specified digital video sprite. This property can be tested but not set.
Example	This statement determines the number of tracks in the digital video sprite assigned to channel 10 and displays the result in the Message window: put sprite(10).trackCount()

trackEnabled

Syntax

```
sprite(whichDigitalVideoSprite).trackEnabled(whichTrack)  
trackEnabled(sprite whichSprite, whichTrack)
```

Description

Digital video sprite property; indicates the status of the specified track of a digital video. This property is TRUE if the track is enabled and playing. This property is FALSE if the track is disabled and no longer playing or is not updating.

This property cannot be set. Use the setTrackEnabled property instead.

Example

This statement checks whether track 2 of digital video sprite 1 is enabled:

```
put sprite(1).trackEnabled(2)
```

See also

setTrackEnabled

trackNextKeyTime

Syntax

```
sprite(whichDigitalVideoSprite).trackNextKeyTime(whichTrack)  
trackNextKeyTime(sprite whichSprite, whichTrack)
```

Description

Digital video sprite property; indicates the time of the keyframe that follows the current time in the specified digital video track.

This property can be tested but not set.

Example

This statement determines the time of the keyframe that follows the current time in track 5 of the digital video assigned to sprite channel 15 and displays the result in the Message window:

```
put sprite(15).trackNextKeyTime(5)
```

trackNextSampleTime

Syntax

```
sprite(whichDigitalVideoSprite).trackNextSampleTime(whichTrack)  
trackNextSampleTime(sprite whichSprite, whichTrack)
```

Description

Digital video sprite property; indicates the time of the next sample that follows the digital video's current time. This property is useful for locating text tracks in a digital video.

This property can be tested but not set.

Example

This statement determines the time of the next sample that follows the current time in track 5 of the digital video assigned to sprite 15:

```
put sprite(15).trackNextSampleTime(5)
```

trackPreviousKeyTime

Syntax

```
sprite(whichDigitalVideoSprite).trackPreviousKeyTime(whichTrack)  
trackPreviousKeyTime(sprite whichSprite, whichTrack)
```

Description

Digital video sprite property; reports the time of the keyframe that precedes the current time.

This property can be tested but not set.

Example

This statement determines the time of the keyframe in track 5 that precedes the current time in the digital video sprite assigned to channel 15 and displays the result in the Message window:

```
put sprite(2).trackPreviousKeyTime(1)
```

trackPreviousSampleTime

Syntax

```
sprite(whichDigitalVideoSprite).trackPreviousSampleTime(whichTrack)  
trackPreviousSampleTime(sprite whichSprite, whichTrack)
```

Description

Digital video sprite property; indicates the time of the sample preceding the digital video's current time. This property is useful for locating text tracks in a digital video.

This property can be tested but not set.

Example

This statement determines the time of the sample in track 5 that precedes the current time in the digital video sprite assigned to channel 15 and displays the result in the Message window:

```
put sprite(15).trackPreviousSampleTime(5)
```

trackStartTime (cast member property)

Syntax

```
member(whichDigitalVideoCastmember).trackStartTime(whichTrack)  
trackStartTime(member whichCastMember, whichTrack)
```

Description

Digital video cast member property; returns the start time of the specified track of the specified digital video cast member.

This property can be tested but not set.

Example

This statement determines the start time of track 5 in the digital video cast member Jazz Chronicle and displays the result in the Message window:

```
put member("Jazz Chronicle").trackStartTime(5)
```

trackStartTime (sprite property)

Syntax `sprite(whichDigitalVideoSprite).trackStartTime(whichTrack)`

`trackStartTime(sprite whichSprite, whichTrack)`

Description Digital video sprite property; sets the starting time of a digital video movie in the specified sprite channel. The value of trackStartTime is measured in ticks.

This property can be tested but not set.

Example In the Message window, this statement reports when track 5 in sprite channel 10 starts playing. The starting time is 120 ticks (2 seconds) into the track.

```
put sprite(10).trackStartTime(5)  
-- 120
```

See also duration, movieRate, movieTime

trackStopTime (cast member property)

Syntax `member(whichDigitalVideoCastmember).trackStopTime(whichTrack)`

`trackStopTime(member whichCastMember, whichTrack)`

Description Digital video cast member property; returns the stop time of the specified track of the specified digital video cast member. It can be tested but not set.

Example This statement determines the stop time of track 5 in the digital video cast member Jazz Chronicle and displays the result in the Message window:

```
put member("Jazz Chronicle").trackStopTime(5)
```

trackStopTime (sprite property)

Syntax `sprite(whichDigitalVideoSprite).trackStopTime(whichTrack)`

`trackStopTime(sprite, whichSprite, whichTrack)`

Description Digital video sprite property; returns the stop time of the specified track of the specified digital video sprite.

When a digital video movie is played, trackStopTime is when playback halts or loops if the loop property is turned on.

This property can be tested but not set.

Example This statement determines the stop time of track 5 in the digital video assigned to sprite 6 and displays the result in the Message window:

```
put sprite(6).trackStopTime(5)
```

See also movieRate, movieTime, trackStartTime (cast member property)

trackText

Syntax `sprite(whichDigitalVideoSprite).trackText(whichTrack)`

`trackText(sprite whichSprite, whichTrack)`

Description Digital video sprite property; provides the text that is in the specified track of the digital video at the current time. The result is a string value, which can be up to 32K characters long. This property applies to text tracks only.

This property can be tested but not set.

Example This statement assigns the text in track 5 of the digital video assigned at the current time to sprite 20 to the field cast member Archives:

```
member("Archives").text = string(sprite(20).trackText(5))
```

trackType (cast member property)

Syntax `member(whichDigitalVideoCastmember).trackType(whichTrack)`

`trackType(member whichCastMember, whichTrack)`

Description Digital video cast member property; indicates which type of media is in the specified track of the specified cast member. Possible values are #video, #sound, #text, and #music.

This property can be tested but not set.

Example The following handler checks whether track 5 of the digital video cast member Today's News is a text track and then runs the handler textFormat if it is:

```
on checkForText
    if member("Today's News").trackType(5) = #text then textFormat
end
```

trackType (sprite property)

Syntax `sprite(whichDigitalVideoSprite).trackType(whichTrack)`

`trackType(sprite whichSprite, whichTrack)`

Description Digital video sprite property; returns the type of media in the specified track of the specified sprite. Possible values are #video, #sound, #text, and #music.

This property can be tested but not set.

Example The following handler checks whether track 5 of the digital video sprite assigned to channel 10 is a text track and runs the handler textFormat if it is:

```
on checkForText
    if sprite(10).trackType(5) = #text then textFormat
end
```

trails

Syntax	<code>sprite(<i>whichSprite</i>).trails</code>
	the trails of sprite <i>whichSprite</i>
Description	Sprite property; for the sprite specified by <i>whichSprite</i> , turns the trails ink effect on (1 or TRUE) or off (0 or FALSE). For the value set by Lingo to last beyond the current sprite, the sprite must be a puppet.
	To erase trails, animate another sprite across these pixels or use a transition.
Example	This statement turns on trails for sprite 7: <code>sprite(7).trails = 1</code>
See also	<code>directToStage</code>

transitionType

Syntax	<code>member(<i>whichCastMember</i>).transitionType</code>
	the transitionType of member <i>whichCastMember</i>
Description	Transition cast member property; determines a transition's type, which is specified as a number. The possible values are the same as the codes assigned to transitions for the <code>puppetTransition</code> command.
Example	This statement sets the type of transition cast member 3 to 51, which is a pixel dissolve cast member: <code>member(3).transitionType = 51</code>

translation

Syntax

member(*whichQuickTimeMember*).translation
the translation of member *whichQuickTimeMember*
sprite(*whichQuickTimeSprite*).translation
the translation of sprite *whichQuickTimeSprite*

Description

QuickTime cast member and sprite property; controls the offset of a QuickTime sprite's image within the sprite's bounding box.

This offset is expressed in relation to the sprite's default location as set by its center property. When center is set to TRUE, the sprite is offset relative to the center of the bounding rectangle; when center is set to FALSE, the sprite is offset relative to the upper left corner of the bounding rectangle.

The offset, specified in pixels as positive or negative integers, is set as a Director list: [xTrans, yTrans]. The *xTrans* parameter specifies the horizontal offset from the sprite's default location; the *yTrans* parameter specifies the vertical offset. The default setting is [0,0].

When the sprite's crop property is set to TRUE, the translation property can be used to mask portions of the QuickTime movie by moving them outside the bounding rectangle. When the crop property is set to FALSE, the translation property is ignored, and the sprite is always positioned at the upper left corner of the sprite's rectangle.

This property can be tested and set.

Example

This frame script assumes that the center property of the cast member of a 320-pixel-wide QuickTime sprite in channel 5 is set to FALSE, and its crop property is set to TRUE . It keeps the playback head in the current frame until the movie's horizontal translation point has moved to the right edge of the sprite, in 10-pixel increments. This has a wipe right effect, moving the sprite out of view to the right. When the sprite is out of view, the playback head continues to the next frame.

```
on exitFrame
    horizontalPosition = sprite(5).translation[1]
    if horizontalPosition < 320 then
        sprite(5).translation = sprite(5).translation + [10, 0]
        go the frame
    end if
end
```

triggerCallback

Syntax

```
sprite(whichQTVRSprite).triggerCallback  
triggerCallback of sprite whichQTVRSprite
```

Description

QuickTime VR sprite property; contains the name of the handler that runs when the user clicks a hotspot in a QuickTime VR movie. The handler is sent two arguments: the me parameter and the ID of the hotspot that the user clicked.

The value that the handler returns determines how the movie processes the hotspot. If the handler returns #continue, the QuickTime VR sprite continues to process the hotspot normally. If the handler returns #cancel, the default behavior for the hotspot is canceled.

Set this property to 0 to clear the callback.

The QuickTime VR sprite receives the message first.

To avoid a decrease in performance, set the triggerCallback property only when necessary.

This property can be tested and set.

Example

This statement sets the callback handler for a QuickTime VR sprite to the handler named MyHotSpotCallback when the playback head first enters the sprite span. Every time that hotspot is triggered, the MyHotSpotCallback handler is executed. When the playback head leaves the sprite span, the callback is canceled.

```
property pMySpriteNum, spriteNum  
  
on beginSprite me  
    pMySpriteNum = me.spriteNum  
    sprite(pMySpriteNum).triggerCallback = #MyHotSpotCallback  
end  
  
on MyHotSpotCallback me, hotSpotID  
    put "Hotspot" && hotSpotID && "was just triggered"  
end  
  
on endSprite me  
    sprite(pMySpriteNum).triggerCallback = 0  
end
```

trimWhiteSpace

Syntax

```
member(whichMember).trimWhiteSpace
```

Description

Cast member property. Determines whether the white pixels around the edge of a bitmap cast member are removed or left in place. This property is set when the member is imported. It can be changed in Lingo or in the Bitmap tab of the Property Inspector.

trimWhitespace()

Syntax *imageObject.trimWhitespace()*

Description Function; removes any white pixels that lie outside the minimum rectangle and returns the result in a new image object.

Example This statement trims the white space from member Flower and returns the new, trimmed image object in the variable trimmedImage.

```
trimmedImage = member("flower").image.trimWhitespace()
```

See also [crop\(\)](#) (member command)

TRUE

Syntax TRUE

Description Constant; represents the value of a logically true expression, such as $2 < 3$. It has a traditional numerical value of 1, but any nonzero integer evaluates to TRUE in a comparison.

Example This statement turns on the soundEnabled property by setting it to TRUE:

```
the soundEnabled = TRUE
```

See also FALSE, if

tweened

Syntax *sprite(whichSprite).tweened*

the tweened of sprite *whichSprite*

Description Sprite property; determines whether only the first frame in a new sprite is created as a keyframe (TRUE), or whether all frames in the new sprite are created as keyframes (FALSE).

This property does not affect playback and is useful only during Score recording.

This property can be tested and set.

Example When this statement is issued, newly created sprites in channel 25 have a keyframe only in the first frame of the sprite span:

```
sprite(25).tweened = 1
```

type (cast member property)

Syntax

```
member(whichCastMember).type  
the type of member whichCastMember  
member( whichCastMember, which castLib). type  
member whichCastMember of castLib whichCast. type  
the type of member whichCastMember of castLib whichCast
```

Description Cast member property; indicates the specified cast member's type. This property replaces the castType property used in previous versions of Director.

The type member property can be one of the following values:

#animgif	#ole
#bitmap	#palette
#button	#picture
#cursor	#QuickTimeMedia
#digitalVideo	#script
#empty	#shape
#field	#sound
#filmLoop	#swa
#flash	#text (#richText is now obsolete)
#font	#transition
#movie	#vectorShape

This list includes those types of cast members that are available in Director and the Xtras that come with it. You can also define custom cast member types for custom cast members.

When a movie plays back as an applet, the type member property is valid only for cast member types that the player supports.

For movies created in Director 5 and 6, the type member property returns #field for field cast members and #richText for text cast members. However, field cast members originally created in Director 4 return #text for the member type, providing backward compatibility for movies that were created in Director 4.

This property can be tested but not set.

Example The following handler checks whether the cast member Today's News is a field cast member and displays an alert if it is not:

```
on checkFormat
    if member("Today's News").type <> #field then alert \
    "Sorry, this cast member must be a field."
end
```

type (sprite property)

Syntax `sprite(whichSprite).type`
the type of sprite *whichSprite*

Description Sprite property; clears sprite channels during Score recording by setting the type `sprite` property value for that channel to 0.

Note: Switch the member of a sprite only to another member of the same type to avoid changing the sprite's properties when the member type is switched.

This property can be tested and set.

Example This statement clears sprite channel 1 when issued during a Score recording session:

```
sprite(1).type = 0
```

union()

Syntax `rect(1).union(rect(2))`
`union (rect1, rect2)`

Description Function; returns the smallest rectangle that encloses the two rectangles `rect1` and `rect2`.

Example This statement returns the rectangle that encloses the specified rectangles:

```
put union (rect (0, 0, 10, 10), rect (15, 15, 20, 20))
-- rect (0, 0, 20, 20)
```

or

```
put rect(0, 0, 10, 10).union(rect(15, 15, 20, 20))
--rect (0, 0, 20, 20)
```

See also `map()`, `rect()`

unLoad

Syntax	unLoad unLoad <i>theFrameNum</i> unLoad <i>fromFrameNum</i> , <i>toFrameNum</i>
Description	Command; forces Director to clear the cast members used in a specified frame from memory. Director automatically unloads the least recently used cast members to accommodate preLoad commands or normal cast loading. <ul style="list-style-type: none">• When used without an argument, the unLoad command clears from memory the cast members in all the frames of a movie.• When used with one argument, <i>theFrameNum</i>, the unLoad command clears from memory the cast members in that frame.• When used with two arguments, <i>fromFrameNum</i> and <i>toFrameNum</i>, the unLoad command unloads all cast members in the range specified. You can specify a range of frames by frame numbers or frame labels.
Example	This statement clears the cast members used in frame 10 from memory: unLoad 10
Example	This statement clears the cast members used from the frame labeled first to the frame labeled last: unLoad "first", "last"
See also	preLoad (command), preLoadMember, unLoadMember, purgePriority

unLoadMember

Syntax	unLoadMember member(<i>whichCastMember</i>). unLoad() unLoadMember member <i>whichCastMember</i> member(<i>whichCastMember</i> , <i>whichCastLib</i>). unLoad() unLoadMember member <i>whichCastMember</i> of <i>castLib</i> <i>whichCast</i> member(<i>firstCastmember</i>). unLoad(<i>lastCastMember</i>) unLoadMember member <i>firstCastMember</i> , <i>lastCastMember</i>
---------------	--

Description	Command; forces Director to clear the specified cast members from memory. Director automatically unloads the least recently used cast members to accommodate preLoad commands or normal cast loading.
	<ul style="list-style-type: none"> • When used without an argument, unLoadMember clears from memory the cast members in all the frames of a movie. • When used with the arguments <i>whichCastMember</i> and <i>whichCast</i>, the unLoadMember command clears from memory the cast member name or number that you specify. • When used with the arguments <i>firstCastMember</i> and <i>lastCastMember</i>, the unLoadMember command unloads all cast members in the range specified.
	When used in a new movie with no loaded cast members, this command returns an error.
	Cast members that you have modified during authoring or by setting picture, pasteClipBoardInto, and so on, cannot be unloaded.
Example	<p>This statement clears from memory the cast member Screen1:</p> <pre>unLoadMember member "Screen1"</pre> <p>or</p> <pre>member("Screen1").unload()</pre>
Example	<p>This statement clears from memory all cast members from cast member 1 to cast member Big Movie:</p> <pre>unLoadMember 1, member "Big Movie"</pre> <p>or</p> <pre>member(1).unload("Big Movie")</pre>
See also	<p>preLoad (command), preLoadMember, purgePriority</p>

unloadMovie

Syntax	<code>unloadMovie <i>whichMovie</i></code>
Description	Command; removes the specified preloaded movie from memory. This command is useful in forcing movies to unload when memory is low. You can use a URL as the file reference. If the movie isn't already in RAM, the result is -1.
Example	<p>This statement checks whether the largest contiguous block of free memory is less than 100K and unloads the movie Parsifal if it is:</p> <pre>if (the freeBlock < (100 * 1024)) then unLoadMovie "Parsifal"</pre>
Example	<p>This statement unloads the movie at http://www.cbDemille.com/SunsetBlvd.dir:</p> <pre>unLoadMovie "http://www.cbDemille.com/SunsetBlvd.dir"</pre>

updateFrame

Syntax

`updateFrame`

Description Command; during Score generation only, enters the changes to the current frame that have been made during Score recording and moves to the next frame. Any objects that were already in the frame when the update session started remain in the frame. You must issue an `updateFrame` command for each frame that you are updating.

Example When used in the following handler, the `updateFrame` command enters the changes that have been made to the current frame and moves to the next frame each time Lingo reaches the end of the repeat loop. The number of frames is determined by the argument `numberOfFrames`.

```
on animBall numberOfFrames
beginRecording
    horizontal = 0
    vertical = 300
    repeat with i = 1 to numberOfFrames
        go to frame i
        sprite(20).memberNum = member("Ball").number
        sprite(20).locH = horizontal
        sprite(20).locV = vertical
        sprite(20).type = 1
        sprite(20).foreColor = 255
        horizontal = horizontal + 3
        vertical = vertical + 2
        updateFrame
    end repeat
endRecording
end
```

See also `beginRecording`, `endRecording`, `scriptNum`, `tweened`

updateLock

Syntax

`the updateLock`

Description Movie property; determines whether the Stage is updated during Score recording (FALSE) or not (TRUE).

You can keep the Stage display constant during a Score recording session by setting `updateLock` to TRUE before Lingo updates the Score. If `updateLock` is FALSE, the Stage updates to show a new frame each time the frame is entered.

You can also use `updateLock` to prevent unintentional Score updating when leaving a frame, such as when you temporarily leave a frame to examine properties in another frame.

Although this property can be used to mask changes to a frame during run time, be aware that changes to field cast members appear immediately when the content is modified, unlike changes to location or members with other sprites, which are not updated until this property is turned off.

updateMovieEnabled

Syntax the updateMovieEnabled

Description System property; specifies whether changes made to the current movie are automatically saved (TRUE) or not saved (FALSE, default) when the movie branches to another movie.

This property can be tested and set.

Example This statement instructs Director to save changes to the current movie whenever the movie branches to another movie:

```
the updateMovieEnabled = TRUE
```

updateStage

Syntax updateStage

Description Command; redraws the Stage immediately instead of only between frames.

The updateStage command redraws sprites, performs transitions, plays sounds, sends a prepareFrame message (affecting movie and behavior scripts), and sends a stepFrame message (which affects actorList).

Example This handler changes the sprite's horizontal and vertical locations and redraws the Stage so that the sprite appears in the new location without having to wait for the playback head to move:

```
on moveRight whichSprite, howFar
    sprite(whichSprite).locH = sprite(whichSprite).locH + howFar
    updateStage
end moveRight
```

URL

Syntax

member(*whichCastMember*).URL
the URL of member *whichCastMember*

Description

Cast member property; specifies the URL for Shockwave Audio (SWA) and Flash movie cast members.

For Flash movie members, this property is synonymous with the pathName member property.

The URL property can be tested and set. For SWA members, it can be set only when the SWA streaming cast member is stopped.

Example

This statement makes a file on an Internet server the URL for SWA cast member Benny Goodman:

```
on mouseDown
    member("Benny Goodman").URL = "http://audio.macromedia.com/samples/classic.swa"
end
```

URLEncode

Syntax

URLEncode(*propList_or_string* {, *serverOSString*} {, *characterSet*})

Description

Function; returns the URL-encoded string for its first argument. Allows CGI parameters to be used in other commands. The same translation is done as for postNetText and getNetText() when they are given a property list.

Use the optional parameter *serverOSString* to encode any return characters in *propList_or_string*. The value defaults to "Unix" but may be set to "Win" or "Mac" and translates any carriage returns in the *propList_or_string* argument into those used on the server. For most applications, this setting is unnecessary because line breaks are usually not used in form responses.

The optional parameter *characterSet* applies only if the user is running on a Shift-JIS (Japanese) system. Its possible settings are "JIS", "EUC", "ASCII", and "AUTO".

Retrieved data is converted from Shift-JIS to the named character set. Returned data is handled exactly as by getNetText() (converted from the named character set to Shift-JIS). If you use "AUTO", the posted data from the local character set is not translated; the results sent back by the server are translated as they are for getNetText(). "ASCII" is the default if *characterSet* is omitted. "ASCII" provides no translation for posting or results.

Example

In the following example, URLEncode supplies the URL-encoded string to a CGI query at the specified location.

```
URL = "http://aserver/cgi-bin/echoquery.cgi"
gotonetpage URL & "?" & URLEncode( [#name: "Ken", #hobby: "What?"] )
```

See also

getNetText(), postNetText

useAlpha

Syntax
`member(whichCastMember).useAlpha`
`imageObject.useAlpha`

Description Bitmap cast member and image object property; for 32-bit cast members and image objects with alpha channel information, determines whether Director uses the alpha information when drawing the image onto the Stage (TRUE), or whether Director ignores the alpha information when drawing to the Stage (FALSE).

Example This toggles the alpha channel of cast member "foreground" on and off.
`member("foreground").useAlpha=not member("foreground").useAlpha`

useFastQuads

Syntax
`the useFastQuads`

Description Global property; when set to TRUE, Director uses a faster, less precise method for calculating quad operations. Fast quads calculations are good for simple rotation and skew sprite effects. Director's slower, default quad calculation method provides more visually pleasing results when using quads for distortion and other arbitrary effects. Defaults to FALSE.

Simple sprite rotation and skew operations always use the fast quad calculation method, regardless of this setting. Setting the useFastQuads to TRUE will not result in an increase in the speed of these simple operations.

Example This statement tells Director to use its faster quad calculation code for all quad operations in the movie:

`the useFastQuads = TRUE`

See also [quad](#)

useHypertextStyles

Syntax

```
member(whichTextMember).useHypertextStyles
```

Description

Text cast member property; controls the display of hypertext links in the text cast member.

When useHypertextStyles is TRUE, all links are automatically colored blue with underlines, and the cursor changes to a pointing finger when it is over a link.

Setting this property to FALSE turns off the automatic formatting and cursor change.

Example

This behavior toggles the formatting of hypertext on and off in text cast member "myText":

```
on mouseUp
    member("myText").usehypertextStyles = not member("myText").usehypertextStyles
end
```

userName

Syntax

```
the userName
```

Description

Movie property; a string containing the user name entered when Director was installed.

This property is available in the authoring environment only. It could be used in a movie in a window (MIAW) tool that is personalized to show the user's information.

Example

This handler places the user's name and serial number in a display field when the window is opened. (A movie script in the MIAW is a good location for this handler.)

```
on prepareMovie
    displayString = the userName
    put RETURN&the organizationName after displayString
    put RETURN&the serialNumber after displayString
    member("User Info").text = displayString
end
```

See also

organizationName, serialNumber, window

value()

Syntax

value(stringExpression)

Description

Function; returns the value of a string. The string can be any expression that Lingo can understand. When value() is called, Lingo parses through the *stringExpression* provided and returns its logical value.

Any Lingo expression that can be put in the Message window or set as the value of a variable can also be used with value().

These two Lingo statements are equivalent:

```
put sprite(2).member.duration * 5  
put value("sprite(2).member.duration * 5")
```

These two Lingo statements are also equivalent:

```
x = (the mouseH - 10) / (the mouseV + 10)  
x = value("(the mouseH - 10) / (the mouseV + 10)")
```

Expressions that Lingo cannot parse will produce unexpected results, but will not produce Lingo errors. The result is the value of the initial portion of the expression up to the first syntax error found in the string.

The value() function can be useful for parsing expressions input into text fields by end-users, string expressions passed to Lingo by Xtras, or any other expression you need to convert from a string to a Lingo value.

Keep in mind that there may be some situations where using value() with user-input can be dangerous, such as when the user enters the name of a custom handler into the field. This will cause the handler to be executed when it is passed to value().

Do not confuse the actions of the value function with the integer() and float() functions.

Example

This statement displays the numerical value of the string "the sqrt of" && "2.0":

```
put value("the sqrt of" && "2.0")
```

The result is 1.4142.

Example

This statement displays the numerical value of the string "penny":

```
put value("penny")
```

The resulting display in the Message window is VOID, because the word *penny* has no numerical value.

Example

You can convert a string that is formatted as a list into a true list by using this syntax:

```
myString = "[" & QUOTE & "cat" & QUOTE & ", " & QUOTE & "dog" & QUOTE & "]"  
myList = value(myString)  
put myList  
-- ["cat", "dog"]
```

This allows a list to be placed in a field or text cast member and then extracted and easily reformatted as a list.

Example This statement parses the string "3 5" and returns the value of the portion of the string that Lingo understands:

```
put value("3 5")
-- 3
```

See also [string\(\)](#), [integer\(\)](#), [float\(\)](#)

version

Syntax `version`

Description Keyword; system variable that contains the version string for Director. The same string appears in the Macintosh Finder's Info window.

Example This statement displays the version of Director in the Message window:

```
put version
```

vertex

Syntax `member(whichVectorShapeMember).vertex[whichVertexPosition]`

Description Chunk expression; enables direct access to parts of a vertex list of a vector shape cast member.

Use this chunk expression to avoid parsing different chunks of the vertex list. It's possible to both test and set values of the vertex list using this type of chunk expression.

Example The following code shows how to determine the number of vertex points in a member:

```
put member("Archie").vertex.count
-- 2
```

Example To obtain the second vertex for the member, you can use code like this:

```
put member("Archie").vertex[2]
-- point(66.0000, -5.0000)
```

Example You can also set the value in a control handle:

```
member("Archie").vertex[2].handle1 = point(-63.0000, -16.0000)
```

See also [vertexList](#)

vertexList

Syntax

`member(whichVectorShapeMember).vertexList`

Description

Cast member property; returns a linear list containing property lists, one for each vertex of a vector shape. The property list contains the location of the vertex and the control handle. There are no control handles if the location is (0,0).

Each vertex can have two control handles that determine the curve between this vertex and the adjacent vertexes. In `vertexList`, the coordinates of the control handles for a vertex are kept relative to that vertex, rather than absolute in the coordinate system of the shape. If the first control handle of a vertex is located 10 pixels to the left of that vertex, its location is stored as (-10, 0). Thus, when the location of a vertex is changed with Lingo, the control handles move with the vertex and do not need to be updated (unless the user specifically wants to change the location or size of the handle).

When modifying this property, be aware that you must reset the list contents after changing any of the values. This is because when you set a variable to the value of the property, you are placing a copy of the list, not the list itself, in the variable. To effect a change, use code like this:

```
- Get the current property contents  
currVertList = member(1).vertexList  
-- Add 25 pixels to the horizontal and vertical positions of the first vertex in the list  
currVertList[1] .vertex = currVertList[1] .vertex + point(25, 25)  
-- Reset the actual property to the newly computed position  
member(1).vertexList = currVertList
```

Example

This statement displays the `vertexList` value for an arched line with two vertexes:

```
put member("Archie").vertexList  
-- [[#vertex: point(-66.0000, 37.0000), #handle1: point(-70.0000, -36.0000), \  
#handle2: point(-62.0000, 110.0000)], [#vertex: point(66.0000, -5.0000), \  
#handle1: point(121.0000, 56.0000), #handle2: point(11.0000, -66.0000)]]
```

See also

`addVertex`, `count()`, `deleteVertex()`, `moveVertex()`, `moveVertexHandle()`, `originMode`, `vertex`

video

Syntax	member(<i>whichCastMember</i>).video the video of member <i>whichCastMember</i>
Description	Digital video cast member property; determines whether the graphic image of the specified digital video cast member plays (TRUE or 1) or not (FALSE or 0). Only the visual element of the digital video cast member is affected. For example, when video is set to FALSE, the digital video's soundtrack, if present, continues to play.
Example	This statement turns off the video associated with the cast member Interview: member("Interview").video = FALSE
See also	setTrackEnabled, trackEnabled

videoForWindowsPresent

Syntax	the videoForWindowsPresent
Description	System property; indicates whether Video for Windows (AVI) is present on the computer. This property can be tested but not set.
Example	This statement checks whether Video for Windows is missing and branches the playback head to the Alternate Scene marker if it isn't: if the videoForWindowsPresent= FALSE then go to "Alternate Scene"
See also	quickTimeVersion()

viewH

Syntax	sprite(<i>whichVectorOrFlashSprite</i>).viewH the viewH of sprite <i>whichVectorOrFlashSprite</i> member(<i>whichVectorOrFlashMember</i>).viewH the viewH of member <i>whichVectorOrFlashMember</i>
Description	Cast member and sprite property; controls the horizontal coordinate of a Flash movie and vector shape's view point, specified in pixel units. The values can be floating-point numbers. The default value is 0. A Flash movie's view point is set relative to its origin point.

Setting a positive value for viewH shifts the movie to the left inside the sprite; setting a negative value shifts the movie to the right. Therefore, changing the viewH property can have the effect of cropping the movie or even of removing the movie from view entirely.

This property can be tested and set.

Note: This property must be set to the default value if the scaleMode property is set to #autoSize, or the sprite will not display correctly.

Example This handler accepts a sprite reference as a parameter and moves the view of a Flash movie sprite from left to right within the sprite's bounding rectangle:

```
on panRight whichSprite
    repeat with i = 120 down to -120
        sprite(whichSprite).viewH = i
        updateStage
    end repeat
end
```

See also [scaleMode](#), [viewV](#), [viewPoint](#), [viewScale](#)

viewPoint

Syntax

```
sprite(whichVectorOrFlashSprite).viewPoint  
the viewPoint of sprite whichVectorOrFlashSprite  
member(whichVectorOrFlashMember).viewPoint  
the viewPoint of member whichVectorOrFlashMember
```

Description

Cast member property and sprite property; controls the point within a Flash movie or vector shape that is displayed at the center of the sprite's bounding rectangle in pixel units. The values are integers.

Changing the view point of a cast member changes only the view of a movie in the sprite's bounding rectangle, not the location of the sprite on the Stage. The view point is the coordinate within a cast member that is displayed at the center of the sprite's bounding rectangle and is always expressed relative to the movie's origin (as set by the *originPoint*, *originH*, and *originV* properties). For example, if you set a Flash movie's view point at point (100,100), the center of the sprite is the point within the Flash movie that is 100 Flash movie pixel units to the right and 100 Flash movie pixel units down from the origin point, regardless of where you move the origin point.

The *viewPoint* property is specified as a Director point value: for example, point (100,200). Setting a Flash movie's view point with the *viewPoint* property is the same as setting the *viewH* and *viewV* properties separately. For example, setting the *viewPoint* property to point (50,75) is the same as setting the *viewH* property to 50 and the *viewV* property to 75.

Director point values specified for the *viewPoint* property are restricted to integers, whereas *viewH* and *viewV* can be specified with floating-point numbers. When you test the *viewPoint* property, the point values are truncated to integers. As a general guideline, use the *viewH* and *viewV* properties for precision; use the *originPoint* property for speed and convenience.

This property can be tested and set. The default value is point (0,0).

Note: This property must be set to the default value if the *scaleMode* property is set to *#autoSize*, or the sprite will not display correctly.

Example

This handler makes a specified Flash movie sprite move down and to the right in increments of five Flash movie pixel units:

```
on panAcross whichSprite  
repeat with i = 1 to 10  
    sprite(whichSprite).viewPoint = sprite(whichSprite).viewPoint + point(i * -5, i * -5)  
    updateStage  
end repeat  
end
```

See also

scaleMode, *viewV*, *viewH*, *viewScale*

viewScale

Syntax

```
sprite(whichVectorOrFlashSprite).viewScale  
the viewScale of sprite whichVectorOrFlashSprite  
member(whichVectorOrFlashMember).viewScale  
the viewScale of member whichVectorOrFlashMember
```

Description

Cast member property and sprite property; sets the overall amount to scale the view of a Flash movie or vector shape sprite within the sprite's bounding rectangle. You specify the amount as a percentage using a floating-point number. The default value is 100.

The sprite rectangle itself is not scaled; only the view of the cast member within the rectangle is scaled. Setting the `viewScale` property of a sprite is like choosing a lens for a camera. As the `viewScale` value decreases, the apparent size of the movie within the sprite increases, and vice versa. For example, setting `viewScale` to 200% means the view inside the sprite will show twice the area it once did, and the cast member inside the sprite will appear at half its original size.

One significant difference between the `viewScale` and `scale` properties is that `viewScale` always scales from the center of the sprite's bounding rectangle, whereas `scale` scales from a point determined by the Flash movie's `originMode` property.

This property can be tested and set.

Note: This property must be set to the default value if the `scaleMode` property is set to `#autoSize`, or the sprite will not display correctly.

Example

This sprite script sets up a Flash movie sprite and doubles its view scale:

```
on beginSprite me  
    sprite(spriteNum of me).viewScale = 200  
end
```

See also

`scaleMode`, `viewV`, `viewPoint`, `viewH`

viewV

Syntax

```
sprite(whichVectorOrFlashSprite).viewV  
the viewV of sprite whichVectorOrFlashSprite  
member(whichVectorOrFlashMember).viewV  
the viewV of member whichVectorOrFlashMember
```

Description

Cast member and sprite property; controls the vertical coordinate of a Flash movie and vector shape's view point, specified in pixel units. The values can be floating-point numbers. The default value is 0.

A Flash movie's view point is set relative to its origin point.

Setting a positive value for viewV shifts the movie up inside the sprite; setting a negative value shifts the movie down. Therefore, changing the viewV property can have the effect of cropping the movie or even of removing the movie from view entirely.

This property can be tested and set.

Note: This property must be set to the default value if the scaleMode property is set to #autoSize, or the sprite will not display correctly.

Example

This handler accepts a sprite reference as a parameter and moves the view of a Flash movie sprite from the top to the bottom within the sprite's bounding: rectangle.

```
on panDown whichSprite  
    repeat with i = 120 down to -120  
        sprite(whichSprite).viewV = i  
        updateStage  
    end repeat  
end
```

See also

scaleMode, viewV, viewPoint, viewH

visible (sprite property)

Syntax `sprite(whichSprite).visible`

the visible of sprite *whichSprite*

Description Sprite property; determines whether the sprite specified by *whichSprite* is visible (TRUE) or not (FALSE). This property affects all sprites in the channel, regardless of their position in the Score.

Note: Setting the visible property of a sprite channel to FALSE makes the sprite invisible and prevents only the mouse-related events from being sent to that channel. The beginSprite, endSprite, prepareFrame, enterFrame, and exitFrame events continue to be sent regardless of the sprite's visibility setting. Clicking the Mute button on that channel in the Score, however, will set the visible property to FALSE and prevent all events from being sent to that channel. Muting disables a channel, while setting a sprite's visible property to FALSE merely affects a graphic property.

This property can be tested and set. If set to FALSE, this property will not automatically reset to TRUE when the sprite ends. You must set the visible property of the sprite to TRUE in order to see any other members using that channel.

Example This statement makes sprite 8 visible:

```
sprite(8).visible = TRUE
```

visible (window property)

Syntax `window whichWindow.visible`

the visible of window *whichWindow*

Description Window property; determines whether the window specified by *whichWindow* is visible (TRUE) or not (FALSE).

This property can be tested and set.

Example This statement makes the Control Panel window visible:

```
window("Control Panel").visible = TRUE
```

VOID

Syntax `VOID`

Description Constant; indicates the value VOID.

Example This statement checks whether the value in the variable currentVariable is VOID:

```
if currentVariable = VOID then  
    put "This variable has no value"  
end if
```

See also `voidP()`

voidP()

Syntax voidP(*variableName*)

Description Function; determines whether the variable specified by *variableName* has any value. If the variable has no value or is VOID, this function returns TRUE. If the variable has a value other than VOID, this function returns FALSE.

Example This statement checks whether the variable *answer* has an initial value:

```
put voidP(answer)
```

See also ilk(), VOID

volume (cast member property)

Syntax member(*whichCastMember*).volume

the volume of member *whichCastMember*

Description Shockwave Audio (SWA) cast member property; determines the volume of the specified SWA streaming cast member. Values range from 0 to 255.

This property can be tested and set.

Example This statement sets the volume of an SWA streaming cast member to half the possible volume:

```
member("SWAfile").volume = 128
```

volume (sound channel)

Syntax sound(*whichChannel*).volume

the volume of sound *channel/Num*

Description System property; determines the volume of the sound channel specified by *channel/Num*. Sound channels are numbered 1, 2, 3, and so on. Channels 1 and 2 are the channels that appear in the Score.

The value of the volume sound property ranges from 0 (mute) to 255 (maximum volume). A value of 255 indicates the full volume set for the machine, as controlled by the soundLevel property, and lower values are scaled to that total volume. This property allows several channels to have independent settings within the available range.

The lower the value of the volume sound property, the more static or noise you're likely to hear. Using soundLevel may produce less noise, although this property offers less control.

This property does not support dot syntax. Use the syntax exactly as shown here.

To see an example of volume (sound channel) used in a completed movie, see the Sound Control movie in the Learning\Lingo Examples folder inside the Director application folder.

Example This statement sets the volume of sound channel 2 to 130, which is a medium sound level setting:

```
sound(2).volume = 130
```

See also fadeIn(), fadeOut(), soundEnabled, soundLevel, fadeTo(), pan()

volume (sprite property)

Syntax `sprite(whichSprite).volume`

the volume of sprite *whichSprite*

Description Sprite property; controls the volume of a digital video movie cast member specified by name or number. The values range from 0 to 256. Values of 0 or less mute the sound. Values exceeding 256 are loud and introduce considerable distortion.

Example This statement sets the volume of the QuickTime movie playing in sprite channel 7 to 256, which is the maximum sound volume:

```
sprite(7).volume = 256
```

See also soundLevel

warpMode

Syntax `sprite(whichQTVRSprite).warpMode`

warpMode of sprite *whichQTVRSprite*

Description QuickTime VR sprite property; specifies the type of warping performed on a panorama.

Possible values are #full, #partial, and #none.

This property can be tested and set. When tested, if the values for the static and motion modes differ, the property's value is the value set for the current mode. When set, the property determines the warping for both the static and motion modes.

Example This sets the warpMode of sprite 1 to #full:

```
sprite(1).warpMode = #full
```

width

Syntax

`member(whichCastMember).width`
the width of member *whichCastMember*
`imageObject.width`
`sprite(whichSprite).width`
the width of sprite *whichSprite*

Description Cast member, image object and sprite property; for vector shape, Flash, animated GIF, bitmap, and shape cast members, determines the width, in pixels, of the cast member specified by *whichCastMember*, *imageObject* or *whichSprite*. Field and button cast members are not affected.

For cast members and image objects, this property can be tested; for sprites, this property can be both tested and set.

Setting this property on bitmap sprites automatically sets the sprite's stretch of sprite property to TRUE.

Example This statement assigns the width of member 50 to the variable height:
`height = member(50).width`

Example This statement sets the width of sprite 10 to 26 pixels:
`sprite(10).width = 26`

Example This statement assigns the width of sprite number *i* + 1 to the variable howWide:
`howWide = sprite(i + 1).width`

See also `height`

window

Syntax

`window whichWindow`

Description Keyword; refers to the movie window—a window that contains a Director movie—specified by *whichWindow*.

Windows that play movies are useful for creating floating palettes, separate control panels, and windows of different shapes. Using windows that play movies, you can have several movies open at once and allow them to interact.

Example This statement opens a window named Navigation:
`open window "Navigation"`

Example This statement moves the Navigation window to the front:

```
moveToFront window "Navigation"
```

or

```
window("Navigation").moveToFront()
```

See also close window, moveToBack, moveToFront, open window

windowList

Syntax the windowList

Description System property; displays a list of references to all known movie windows.

Example This statement displays a list of all known movie windows in the Message window:

```
put the windowList
```

Example This statement clears windowList:

```
the windowList = [ ]
```

See also windowPresent()

windowPresent()

Syntax windowPresent(*windowName*)

Description Function; indicates whether the object specified by *windowName* is running as a movie in a window (TRUE) or not (FALSE). If a window had been opened, windowPresent remains TRUE for the window until the window has been removed from the windowList property.

The *windowName* argument must be the window's name as it appears in the windowList property.

Example This statement tests whether the object myWindow is a movie in a window (MIAW) and then displays the result in the Message window:

```
put windowPresent(myWindow)
```

See also the windowList

windowType

Syntax

window *whichWindow*.windowType

the windowType of window *whichWindow*

Description

Window property; controls the display style of the window specified by *whichWindow*, as follows:

- 0—Movable, sizable window without zoom box
- 1—Alert box or modal dialog box
- 2—Plain box, no title bar
- 3—Plain box with shadow, no title bar
- 4—Movable window without size box or zoom box
- 5—Movable modal dialog box
- 8—Standard document window
- 12—Zoomable, nonresizable window
- 16—Rounded-corner window
- 49—Floating palette, during authoring (in Macintosh projectors, the value 49 specifies a stationary window)

You can set this property before opening the window. Numbers 6, 7, 9, 10, 11, 13, 14, 15, and 17 through 48 have no effect when used as the windowType value.

You can change the windowType setting after a window has been opened, but a delay may occur while the window is redrawn.

If no windowType setting is specified, a value of 0 is used.

In Microsoft Windows, these numbers create windows with the same functionality just described, but with a Windows appearance. Other values for windowType are possible, but use them with caution, because some modal windows can be exited only when you restart the computer.

Example

This statement sets the value of the display style of the window named Control Panel to 8:

```
window("Control Panel").windowType = 8
```

word...of

Syntax	<code>member(whichCastMember).word[whichWord]</code> <code>textMemberExpression.word[whichWord]</code> <code>chunkExpression.word[whichWord]</code> <code>word whichWord of fieldOrStringVariable</code> <code>fieldOrStringVariable. word[whichWord]</code> <code>textMemberExpression.word[firstWord..lastWord]</code> <code>member(whichCastMember).word[firstWord..lastWord]</code> <code>word firstWord to lastWord of chunkExpression</code> <code>chunkExpression.word[whichWord..lastWord]</code>
Description	Chunk expression; specifies a word or a range of words in a chunk expression. A word chunk is any sequence of characters delimited by spaces. (Any nonvisible character, such as a tab or carriage return, is considered a space.) The expressions <code>whichWord</code> , <code>firstWord</code> , and <code>lastWord</code> must evaluate to integers that specify a word in the chunk. Chunk expressions refer to any character, word, item, or line in any source of characters. Sources of characters include field and text cast members and variables that hold strings. To see an example of <code>word...of</code> used in a completed movie, see the Text movie in the Learning\Lingo Examples folder inside the Director application folder.
Example	These statements set the variable named <code>animalList</code> to the string “fox dog cat” and then insert the word <code>elk</code> before the second word of the list: <code>animalList = "fox dog cat"</code> <code>put "elk" before animalList.word[2]</code> The result is the string “fox elk dog cat”.
Example	This statement tells Director to display the fifth word of the same string in the Message window: <code>put "fox elk dog cat".word[5]</code> Because there is no fifth word in this string, the Message window displays two quotation marks (""), which indicate an empty string.
See also	<code>char...of</code> , <code>line...of</code> , <code>item...of</code> , <code>count()</code> , <code>number (words)</code>

wordWrap

Syntax	<code>member(<i>whichCastMember</i>).wordWrap</code>
	the wordWrap of member <i>whichCastMember</i>
Description	Cast member property; determines whether line wrapping is allowed (TRUE) or not (FALSE).
Example	This statement turns line wrapping off for the field cast member <i>Rokujo</i> : <code>member("Rokujo").wordWrap = FALSE</code>

xtra

Syntax	<code>xtra <i>whichXtra</i></code>
Description	Function; returns an instance of the scripting Xtra specified by <i>whichXtra</i> . To see an example of xtra used in a completed movie, see the Read and Write Text movie in the Learning\Lingo Examples folder inside the Director application folder.
Example	This statement uses the new() function to create a new instance of the Multiuser Xtra and assigns it to the variable <i>tool</i> : <code>tool = new(xtra "Multiuser")</code>
See also	new()

xtraList

Syntax	<code>the xtraList</code>
Description	System property; displays a linear property list of all available Xtras and their file versions. This property is useful when the functionality of a movie depends on a certain version of an Xtra. There are two possible properties that can appear in XtraList:
<hr/>	

#name	Specifies the file name of the Xtra on the current platform. It is possible to have a list without a #name entry, such as when the Xtra exists only on one platform.
#version	Specifies the same version number that appears in the Properties dialog box (Windows) or Info window (Macintosh) when the file is selected on the desktop. An Xtra may not necessarily have a version number.

Example	This statement displays the xtraList in the Message window: <code>put the xtraList</code>
----------------	--

Example This handler checks the version of a given Xtra:

```
-- This handler checks all the available Xtras to return the version of the requested Xtra
-- The XtraFileName may be only a partial match if desired
-- Use the full file name for more exact usage
-- The string returned is either the version property for the Xtra or an empty string
-- It may be empty if either the version property doesn't exist or the Xtra is not found in the
available list
on GetXtraVersion XtraFileName
    -- Get the entire list of Xtras and their information
    listOfXtras = the xtraList
    -- Initialize the local variable to contain the version
    theVersion = ""
    -- Iterate through all the Xtras listed
    repeat with currentXtra in listOfXtras
        -- If the current Xtra's name contains the Xtra passed in, then check for the version
        if currentXtra.name contains XtraFileName then
            -- First determine if the version property exists for that Xtra
            versionFlag = getaProp(currentXtra, #version)
            -- If the version property is not VOID, then set the local variable to
            -- the string contained in that property value
            if not voidP(versionFlag) then
                theVersion = currentXtra.version
            end if
        end if
    end repeat
    -- Return the version information found or an empty string
    return theVersion
end
```

See also movieXtraList, getaProp

xtras

See number of xtras

zoomBox

Syntax

```
zoomBox startSprite, endSprite {,delayTicks}
```

Description

Command; creates a zooming effect, like the expanding windows in the Macintosh Finder. The zoom effect starts at the bounding rectangle of *startSprite* and finishes at the bounding rectangle of *endSprite*. The zoomBox command uses the following logic when executing:

- 1 Look for *endSprite* in the current frame; otherwise,
- 2 Look for *endSprite* in the next frame.

Note, however, that the zoomBox command does not work for *endSprite* if it is in the same channel as *startSprite*.

The *delayTicks* variable is the delay in ticks between each movement of the zoom rectangles. If *delayTicks* is not specified, the delay is 1.

Example

This statement creates a zoom effect between sprites 7 and 3:

```
zoomBox 7, 3
```

zoomWindow

Syntax

```
on zoomWindow  
    statement(s)  
end
```

Description

System message and event handler; contains statements that execute whenever a movie running as a movie in a window (MIAW) is resized. This happens when the user clicks the Minimize or Maximize button (Windows) or the Zoom button (Macintosh). The operating system determines the dimensions after resizing the window.

An on zoomWindow event handler is a good place to put Lingo that rearranges sprites when window dimensions change.

Example

This handler moves sprite 3 to the coordinates stored in the variable centerPlace when the window that the movie is playing in is resized:

```
on zoomWindow  
    centerPlace = point(10, 10)  
    sprite(3).loc = centerPlace  
end
```

See also

drawRect, sourceRect, on resizeWindow

INDEX

-- (comment delimiter) 42
- (minus sign) operator 41

Symbols

" (quotation mark) 409
" (string constant) 51
(symbol definition operator) 40
& (concatenation operator) 42
&& (concatenation operator) 43
() (parentheses operator) 44
* (multiplication operator) 45
+ (addition operator) 45
/ (division operator) 46
< (less than operator) 46
<= (less than or equal to operator) 47
<> (not equal operator) 47
= (equal sign operator) 47
> (greater than operator) 48
>= (greater than or equal to operator) 48
@ (pathname operator) 51
[] (list brackets) 49
¬ (continuation symbol) 51

A

abbr, abbrev, or abbreviated elements 53
abbr, abbrev, or abbreviated time function 504
abs (absolute value) function 53
absolute (abs) function 53
active sprites 108
activeCastLib system property 55
activeWindow system property 56
actorList property 56, 368
adding
 to linear lists 57, 58, 67
 to property lists 58
addition operator (+) 45
AIFF files 472
Alt key (Windows) 353
ampersand operators (& or &&) 42, 43
ancestor, sending messages to 91

and logical operator 65
applications
 shutDown command and 464
 starting 351
arctangent (atan) function 68
ASCII codes 102, 347
assigning
 cast members to sprites 292
 palettes to cast members 360, 361
asterisk operator (*) 45

B

background colors, setting with cast members 70
background events, processing 123
BACKSPACE character constant 72
beep elements
 alert command 59
 beep command 72
 beepOn property 72
Behavior Inspector, strings in description pane 204
behavior scripts, sprites and 442
behaviors, attaching 252
bit rates 77
bitmap, trimming whitespace 520
bitmap compression 243, 320
bitmapped cast members
 color depth of 144
 palettes associated with 361
 registration point of 416
 size of 465
bitmaps, picture of member property 371, 372
borders
 of field cast members 80
 of shape cast members 270
boxes for field cast members 284
brackets ([]) 49

branching
case keyword 93
end case keyword 157
if...then statements 237
otherwise keyword 359
repeat while keyword 418

browsers
clearing cache in 105
displaying strings in 332
location of 84
preloading files from internet 389
retrieving files and 211
sending strings to 167
setting cache size of 88

buttons
buttonStyle property 86
buttonType property 86
checkBoxAccess 103
hilite of member 226
radio button sprites 226

C

cache
clearing in browsers 105
refreshing web pages and 88
setting size in browsers 88

cancelling network operations 327

case keyword 93, 157, 359

cast members
assigned to sound channels 197
assigning to sprites 292
borders 270
canceling loading of 92
cast member numbers 180, 341
changing cast member names 324
changing cast members used for cursors 130
copying 122, 152, 288, 371
creating 333
deleting 162
determining if loaded in memory 273
displaying information about when loading 512
downloading from internet 195, 289
duration of 153
editing 154, 298
empty cast members 180
font used to display 188
height of 225

cast members (*continued*)
importing files into 243
in cast window 450
last cast member in casts 346
line spacing for 270
lines in 269
loading 181, 235, 236
media in cast member tracks 517
media of member function 288, 371
moving 316
names 324
palettes associated with 360, 361
pasting Clipboard contents into 364
pausing playback of 367
preloading 235, 385, 387
replacing content of 243
setting background colors with 70
size of 465
text boxes for 82
text of script assigned to 444
unloading from memory 402, 524
width of 542

cast window, cast members selected in 450

casts

activeCastLib system property 55
cast numbers 340
castLib keyword 94
castLibNum of member 95
file names of 173
last cast member in 346
names of 324
number of in movies 341
preload mode of 388
saving changes to 435
selection of castLib 450

CGI query 209

channel numbers, sprite behaviors and 475

channels

cast members assigned to sound channels 197
channelCount of member property 99
checking sound channels 467
closing sound channels 467
palette channels as puppets 397
selecting in Score window 440
sound channels as puppets 398
tempo channels as puppets 400

character constants
 BACKSPACE 72
 EMPTY 155
 ENTER 160
 QUOTE 409
 RETURN 423
 TAB 498

character strings
 ASCII codes 102, 347
 comparing 117
 converting expressions to 494
 counting characters in 346
 counting items in 342
 counting lines in 343
 counting words in chunk expressions 345
 deleting chunk expressions 140
 EMPTY character constant 155
 ending character in selections 451
 expressions as 494
 field keyword 172
 highlighting 226
 in field cast members 502
 integer function 248
 item...of keyword 254
 last function 264
 length function 267
 line...of keyword 268
 numerical values of 531
 offset function 349
 put...after command 403
 put...before command 404
 put...into command 404
 selecting 450
 selecting words in 545
 starting character in selections 451
 starts comparison operator 481

characters, locating in field cast members 275, 276

checkboxes
 checkbox sprites 226
 checkBoxAccess property 103
 checkBoxType property 103
 hilite of member property 226

child objects
 ancestor property 64
 creating 333
 me keyword 287
 sending messages to ancestor of 91
 tracking 56

chunk expressions
 char...of keyword 99
 counting characters in 346
 counting items in 342
 counting lines in 343
 counting words in 345
 deleting 140
 field keyword 172
 highlighting 226
 item...of keyword 254
 last function 264
 line...of keyword 268
 put...after command 403
 put...before command 404
 put...into command 404
 selecting words in 545
 word...of keyword 545

Clipboard
 copying cast members to 122
 pasting contents into cast members 364

closing
 sound channels 467
 windows 109, 191

color depth
 colorDepth property 113
 depth of member property 144
 of bitmapped cast members 144
 of monitors 113
 setting (Macintosh) 496

colors
 Score color assigned to sprites 439
 setting background colors with cast members 70
 setting foreground colors of field
 cast members 190
 setting Stage color 477

Command key (Macintosh) 114

comment delimiter (--) 42

comparing values 93

comparison operators
 contains 117
 equal sign operator (=) 47
 greater than operator (>) 48
 greater than or equal to operator (>=) 48
 less than operator (<) 46
 less than or equal to operator (<=) 47
 not equal operator (<>) 47
 sprite...within 474
 starts 481

computers
 platform system property 372
 restarting computers 421
 turning off 464

concatenation operators (& or &&) 42, 43

conditions
 comparing 93
 end case keyword 157
 if...then statements 237
 otherwise keyword 359
 repeat while keyword 418

constants
 BACKSPACE 72
 EMPTY 155
 ENTER 160
 FALSE 172
 QUOTE 409
 RETURN 423
 SPACE 473
 TAB 498
 TRUE 521
 VOID 539

continuation symbol () 51

Control key (Macintosh) 118

Control key (Windows) 114, 118

converting
 ASCII codes to characters 347
 characters to ASCII codes 102
 duration in time to frames 228
 expressions to floating-point numbers 185
 frames to duration in time 198
 time to ticks 504

coordinates
 of points 378
 of rectangles 414, 473
 of sprites 81, 115, 117, 266, 425, 511
 of windows 415

Copy command, enabling 155

copying
 cast members 122, 152, 288, 371
 frames 151
 lists 151

cos (cosine) function 122

counting
 characters in strings 267
 items in lists 123
 parameters sent to handler 363
 tracks on digital video sprites 513

CPU, processing background events 123

creating
 cast members 333
 child objects 333
 custom menus 294
 delimiters for items 255
 linear lists 272
 rectangles 413
 Xtras 333

cue points
 list of names 128
 list of times 128
 sprites and 253, 300

cursors
 changing cast members used for 130
 cursor command 130
 cursor of sprite property 132
 cursor resources 132
 identifying character under 301
 mouse cursor position 305, 313, 314
 mouseChar function 301
 mouseMember function 309
 rollOver tests 428

curve, adding 336

Cut command, enabling 155

D

debugging
 with put command 403
 with showGlobals command 462
 with showLocals command 462

deleting
 all items in list 141
 cast members 162
 chunk expressions 140
 contents of frame's sprite 106
 frames from movies 142
 items in lists 141, 143
 movies from memory 525
 values from lists 142
 windows 191

delimiters
 comment delimiter (--) 42
 defining for items 255

description pane, strings in 204

digital video
 deleting frames from movies 142
 duration of 153
 format of 146
 pausing movies 118
 playing in front of layers 146
 playing tracks on 461
 preloading movies 390

digital video cast member properties
 center of member 96
 controller of member 119
 digitalVideoType of member 146
 directToStage of member 146
 frameRate of member 194
 loop of member 278
 streaming of member 491
 timeScale of member property 510
 trackStartTime(member) 515
 trackStopTime(member) 516
 trackType(member) 517
 video of member 534, 539
 volume of sprite 541

digital video cast members
 counting tracks on 513
 media in tracks 517
 Paused At Start of member property 366
 preloading into memory 385
 start time of tracks 515

digital video cast members (*continued*)

 stop time of tracks 516
 time scale for 145
 turning play of on or off 534, 539

digital video sprite properties
 trackNextKeyTime 514
 trackNextSampleTime 514
 trackPreviousKeyTime 515
 trackPreviousSampleTime 515
 trackText 517
 trackType(sprite) 517

digital video sprites
 counting tracks on 513
 media in tracks 517
 playing tracks on 514
 text in tracks 517

dimensions of rectangles 245

Director, exiting 408, 464

disabling
 sound 469

 stream status reporting 502

distance, of lines 270

division operator (/) 46

division, remainders of 297

documents, opening applications with 351

double-byte characters 246

downloading cast members from internet 289

drawing 149

drop shadows of field cast members 82, 150

E

e logarithm functions 167, 277

editing

 cast members 154

 field sprites 155

EMBED tags, external parameters 169, 170

empty cast members 180

EMPTY character constant 155

enabling sound 469

enabling stream status reporting 502

end case keyword 157

end keyword 157

end repeat 158

ending Score recording session 158

ENTER character constant 160

Enter key 160

equal sign operator (=) 47

errors
 during network operations 328
 on alertHook event handler 159
 Shockwave Audio cast members and 204, 206
evaluating expressions 147, 403, 404, 453
event handlers
 ancestor property 64
 counting parameters sent to 363
 exit repeat keyword 167
 exiting 53, 165, 224
 marking end of 157
 on activateApplication 54
 on activateWindow 55
 on alertHook 159
 on closeWindow 159
 on cuePassed 159
 on deactivateApplication 137
 on deactivateWindow 159
 on endSprite 159
 on enterFrame 165, 368
 on exitFrame 165
 on idle 233
 on keyDown 507
 on keyUp 261
 on keyword 351
 on mouseDown 302, 508
 on mouseEnter 306
 on mouseLeave 306
 on mouseUp 314
 on mouseUpOutSide 314
 on mouseWithin 314
 on moveWindow 352
 on openWindow 352
 on prepareFrame 391
 on rightMouseUp 427
 on stopMovie 493
 on zoomWindow 351
result function 422
return keyword 423
tell command 500
timeOut 351
event messages
 custom 452
 passing 38, 363
 stopping 351

events
 identifying sprites in 129
 outside of windows 298
exiting
 Director 408, 464
 handlers 53, 165, 224
 projectors 166
exponents 384
expressions
 as integers 249
 as strings 494
 as symbols 497, 498
 converting to floating-point numbers 185
 converting to strings 494
 evaluating 147, 403, 404, 453
 FALSE expressions 172
 floating-point numbers in 186
 logical negation of 338

F

fading in sound 172
fading out sound 172
FALSE logical constant 172
field cast member properties
 alignment of member 61
 autoTab of member 70
 border of member 80
 boxDropShadow of member 82
 dropShadow of member 150
 editable of member 154
 font of member 188
 fontSize of member 189
 fontStyle of member 189
 lineCount of member 269
 lineHeight of member 270
 margin of member 284
 pageHeight of member 359
 selEnd 451
 selStart 451
 wordWrap of member property 546
field cast members
 drop shadows for 150
 field keyword 172
 font size of 189
 font style of 189
 height of lines in 269
 installing menus defined in 248

field cast members (*continued*)
lineHeight function 269
lineHeight of member property 270
locToCharPos function 275
locVToLinePos function 276
position of lines in 270
scrolling 445, 446
strings in 502
text boxes for 82
field sprites
editing 155
mouseChar function 301
file names
fileName of window property 175
finding 211, 447
of casts 173
of linked cast members 174
file size of movies 319
files
preloading from internet 148, 389
retrieving 211
retrieving text from over network 209
writing strings to 459
Finder, quitting from Director to (Macintosh) 408
finding
empty cast members 180
file names 211, 447
movies 211
Flash movies
mixing sounds 471
setting properties of 456
setting variables in 461
floating-point numbers 185, 186
folders
of current movie 321, 365
searchCurrentFolder function 447
fonts 188, 189
foreground colors, setting with cast members 190
forward slash (/) 46
frame properties
frameLabel 193
framePalette 194
frameRate of member 194
frameScript 197
frameSound1 197
frameSound2 197
frameTempo 199

frame properties (*continued*)
frameTransition 199
lastFrame 266
timeScale 510
trackNextKeyTime 514
trackPreviousKeyTime 515
frame scripts
cast member number of 197
rollOver tests 428
frames
clearFrame command 106
converting duration in time to 228
converting to duration in time 198
copying 151
deleting 142
deleting frame's sprite contents 106
displaying number of current frame 192
framesToHMS function 198
going to 219
HMSToFrames function 228
inserting 247
labels assigned to 193
listing frame labels 263
marker function 284
marker labels and 263
markers before and after 284
memory needed to display 409
number of palette in 194
on enterFrame event handler 165
on exitFrame event handler 165
on prepareFrame event handler 391
playing 373
printing 393
tempo settings 199
transitions between 199, 400
updating 526
free memory 199, 200, 293, 319, 409
front slash (/) 46
FTP proxy servers, setting values of 395

G

generating random numbers 411
global properties, cpuHogTicks 123
global variables 106, 218, 404, 462
greater than operator (>) 48
greater than or equal to operator (>=) 48
grouping operator () 44

H

handlers
 counting parameters sent to 363
 exit repeat keyword 167
 exiting 53, 165, 224
 invoking 89
 marking end of 157
 result function 422
 return keyword 423
 tell command 500
 timeOut 351

height
 height of member property 225
 lineHeight function 269
 of cast members 225
 of field cast members 359
 of lines in field cast members 269

hiding cast member controllers 119

highlighting text 226

HTTP headers, date last modified 330

HTTP proxy servers, setting values of 395

I

idle time, length of 234

if...then...else statements 237, 339

image object
 copying 152
 drawing in 149
 size of 414

importing 243

in keyword 244

inks
 ink of sprite property 245
 trails effect 518

inserting frames 247

installing menus defined in field cast members 248

integers
 integer function 248
 integerP function 249
 maxInteger property 286
 random 410

Internet
 downloading cast members from 195, 289
 playing Shockwave movies from 222
 preloading files from 148, 389

internet files, MIME types and 331

interpreters for Lingo 429
intersecting sprites 474
items, separating 255

K

key character constants
 BACKSPACE 72
 ENTER 160
 RETURN 423
 TAB 498

key frames
 sprites and 521
 trackNextKeyTime property 514
 trackPreviousKeyTime property 515

keyboard, on timeOut event handler and 351

keys
 Alt key (Windows) 353
 assigning scripts for 260
 assigning scripts to 262
 Backspace key (Windows) 72
 Command key (Macintosh) 114
 Control key (Macintosh) 118
 Control key (Windows) 114, 118
 Delete key (Macintosh) 72
 Enter key 160
 last key pressed 256, 258, 260
 lastEvent function 265
 lastKey function 266
 on keyDown event handler 507
 on keyUp event handler 261
 Option key (Macintosh) 353
 RETURN character constant 423
 Shift key 462
 Tab key 498

L

labels 193, 263
launching applications 351
less than operator (<) 46
less than or equal to operator (≤) 47
line spacing for cast members 270
line wrapping 546
linear lists
 adding to 57, 58
 appending to 67
 creating 272
 deleting values from 142
lines
 continuation symbol (\) 51
 distance of 270
 drawing in image object 149
 height of 269
 in cast members 269
Lingo errors 159
Lingo interpreters 429
Lingo Xtras 346
linked cast members 173
linked movies 175, 443
linked scripts 271
list brackets ([]) 49
list function 272
list of sprite references 442
list operators ([]) 49
listing frame labels 263
lists
 adding to linear lists 57, 58
 adding to property lists 58
 appending to 67
 copying 151
 count function 123
 counting items in 123
 creating linear lists 272
 cue point names 128
 cue point times 128
 deleting all items in 141
 deleting items or values from 141, 142, 143
 findPos command 180
 findPosNear command 181
 getProp command 201
 getAt command 202
 getLast command 208
 getOne command 211

lists (*continued*)
 getPos command 214
 getProp command 215
 getPropAt command 215
 identifying items in 201, 202, 208, 211, 214, 215
 ilk function 238
 list operators ([]) 49
 listP function 273
 maximum value in 286
 minimum value in 296
 of movie windows 543
 position of properties in property lists 180, 181
 replacing property values from 454, 460
 setProp command 454
 setAt command 455
 setProp command 460
 sorting 466
 types of 238, 273
 value of parameters in 362
 windowList property 543
literal quotation mark ("") 409
loading cast members
 cancelIdleLoad command 92
 determining if loaded 273
 displaying information when 512
 finishIdleLoad command 181
 idleLoadDone function 235
 idleLoadPeriod property 235
 idleLoadTag system property 236
 idleReadChunkSize property 236
 preloading cast members 385, 387
 purge priority of cast members 402
 unloading cast members 524
local variables 404, 463
location
 of cast members 100
 of mouse cursor 305, 313, 314
 of sprites 116, 274, 275, 474
 of Stage on desktop 476, 478, 480
location number of sprites 291
log files of Message window display 513
logarithm functions 167, 277
logical constants
 FALSE 172
 TRUE 521
logical expressions 65
logical negation of expressions 338

long time format 504
loops
 loop keyword 277
 next repeat keyword 337
 repeat with keyword 419
 repeat with...down to keyword 420
 repeat with...in list keyword 420

M

Macintosh computers
 beep sound and 72
 Lingo interpreters 429
 platform system property 372
 restarting 421
 turning off 464
marker labels, frames associated with 263
markers
 before and after frames 284
 getting list of 285
 go loop command 220
 going to next marker 221
 going to previous marker 221
 loop keyword 277
 next keyword 336
marking end of handlers 157
MCI (Media Control Interface) 287
Media Control Interface (MCI) 287
memory
 allocated to program 293
 determining if cast members are loaded 273
 free 199, 200, 293, 319, 409
 memorySize function 293
 preloading cast members 385, 387
 preloading movies into 388
 ramNeeded function 409
 required to display frames 409
 size of free blocks 199
 unloading cast members from 524
 unloading movies from 525
menu items
 checkMark of menuItem 104
 enabled of menuItem 156
 name of menu 325
 name of menuItem 325
 number of menus 346
 script of menuItem 441
 scripts executed by 441

menu items
 selecting 156
 setting text in 325
menu keyword 294
menus
 defining custom menus 294
 in current movie 346
 installing menus defined in field
 cast members 248
 name of menu property 325
Message window
 displaying expression results in 403
 displaying global variables 462
 writing display to log files 513
messages
 alert command and 59
 custom event messages for sprites 452
 error 204, 206
 event 351
 idle time and 234
 invoking handlers with 89
 passing 38, 363
 sending to child's ancestor 91
 stage system property 476
 tell command 500
methods
 exiting 165
 marking end of handlers 157
 return keyword 423
MIAW 175, 191, 500, 543
MIME files 223, 331
minus sign operator (-) 41
mod (modulus) operator 297
monitors
 centering Stage on 98
 color depth of 113
 size and position of 144
mouse clicks
 assigning scripts for 303, 312
 clickLoc function 107
 clickOn function 108
 determining if mouse button is pressed 303, 309
 doubleClick function 148
 emulateMultiButtonMouse property 156
 lastClick function 265
 lastEvent function 265
 mouseDown function 303

mouse clicks (*continued*)
 mouseUp function 309
 on mouseDown event handler 302, 508
 on mouseEnter event handler 306
 on mouseLeave event handler 306
 on mouseUp event handler 314
 on mouseUpOutSide event handler 314
 on mouseWithin event handler 314
 on rightMouseUp event handler 427
 right mouse button status 426, 427
 stillDown function 486

mouse cursor position 305, 313, 314

mouse position 305, 308

mouse roll function (*lastRoll*) 266

mouse, on timeOut event handler and 351

movie in a window 175, 191, 500, 543
 hiding 68
 minimizing 68

movie names 318, 321, 365

movie properties
 allow zooming 63
 center of member 96
 controller of member 119
 idleHandlerPeriod 234
 idleReadChunkSize 236
 paletteMapping 360
 scoreSelection 440
 scriptsEnabled of member 443
 updateLock 526
 updateMovieEnabled 527
 videoForWindowsPresent 534

movie windows
 drawRect of window property 150
 frontmost movie in 200
 list of 543
 running objects as movies in 543

movies
 color depth settings 496
 deleting frames from 142
 determining version created with 320
 file size of 319
 finding 211
 go loop command 220
 going to frames 219
 idle time and 233
 inserting frames 247
 last frame in 266

movies (*continued*)
 length of in time 322
 movie names 318, 321, 365
 number of casts in 341
 number of menus in 346
 on stopMovie event handler 493
 pausing 118, 366, 367
 playback rates 322
 playing from a marker 221
 playing with play command 373, 376
 preloading 388
 responding to events 298
 run mode of 431
 saving 436, 527
 Score associated with 439
 Shockwave 169, 170, 222
 stopping movie playback 139, 224
 unloading from memory 525
 unused space in 319
 Xtras available to 346

moving
 cast members 316
 sprites 315
 windows 316, 317

multiplication operator (*) 45

N

names of cast members 324
names of casts 324
names of movies 318, 321, 365
naming windows 326
natural logarithm functions 167, 277
negation operator (-) 41
network operations
 canceling 327
 errors during 328
 MIME types and 331
 text returned by 332

network servers, retrieving text from files on 209

new line symbol (\\) 51

not equal operator (<>) 47

number sign (#) 40

numbers
 cast member number of frame scripts 197
 cast numbers 340
 converting expressions to floating-point
 numbers 185

numbers (*continued*)

- displaying number of current frame 192
- exponents 384
- floating-point numbers 185, 186
- largest supported by system 286
- random number generation 411
- script number assigned to sprites 443

O

OBJECT tags 169, 170

opening

- applications 351
- MIME files 223
- Shockwave movies 223
- windows 352

operators

- mod 297
- not 338
- or 354
- sprite...intersects 474

Option key (Macintosh) 353

P

palette channels as puppets 397

palettes

- assigning to cast members 360, 361
- in current frame 194
- remapping 360
- Score color assigned to sprites 439

parameters

- counting parameters sent to handlers 363
- in lists 362

parent scripts

- ancestor property 64
- me keyword 287
- objects created by 348

parentheses operator () 44

Paste command, enabling 155

pasting Clipboard contents into cast members 364

pathname operator (@) 51

pathnames 51, 321, 365, 447, 448

paths

- applicationPath property 67
- browsers and 84
- searched by Director 448

patterns, filling shape cast members with 178

pausing movies 118, 366, 367

pausing playback of cast members 367

playback rates 322

playing AIFF files 472

playing AVI files (Windows) 472

playing digital video tracks 461

playing movies

- after a pause 118

- going to a frame 219

- going to a marker 221

- setting frame rate 194

- with play command 373, 376

playing Shockwave movies from internet 222

playing sound 467, 472

playing time of sound sprites 129

playing WAVE files 472

plus sign (+) 45

points

- coordinates of 378

- identifying 273

- point function 378

- positioning and sizing 282

- type of 238

position

- of cast members 100

- of mouse cursor 305, 313, 314

- of sprites 116, 274, 275, 474

- of Stage on desktop 476, 478, 480

positioning rectangles and points 282

pound sign (#) 40

preloading

- cast members 235

- files from internet 148, 389

- Shockwave Audio cast members 390

printing

- frames 393

- Stage 393

projectors

- exiting 166

- hiding 68

- minimizing 68

property lists

- adding to 58

- deleting values from 143

- findPos command 180

- findPosNear command 181

- position of properties in 180, 181

- property list (*continued*)
 replacing property values from 454, 460
 setaProp command 454
 setProp command 460
property variables 393, 404
proxy servers, setting values of 395
puppets
 palette channels as 397
 puppet sprite blend values 79
 sound channels as 398
 sprites as 396, 399
 tempo channels as 400
purge priority of cast members 402
- Q**
- QuickTime, masks 285
quitting
 Director 408, 464
 handlers 53, 165, 224
 projectors 166
quotation mark (") 51, 409
QUOTE character constant 409
- R**
- radio buttons, checkBoxAccess property 103
random integers 410
random number generation 411
rectangles
 changing dimensions of 245
 coordinates of 414, 415, 473
 defining 413
 inflate rect command 245
 inside function 247
 intersect function 250
 offsetting 350
 positioning and sizing 282
 type of 238
 union rect function 523
rects, identifying 273
redrawing Stage 527
refreshing web pages 88
registration points
 of bitmapped cast members 416
 of sprites 274, 275
regPoint property 416
updating 371
- relative paths, pathname operator (@) 51
remainders of division 297
remapping palettes 360
repeat loops
 exit repeat keyword 167
 next repeat keyword 337
 repeat with keyword 419
 repeat with...down to keyword 420
 repeat with...in list keyword 420
resource files
 displaying resources in 463
 opening 351, 353
retrieving files 211
retrieving text from files on network servers 209
RETURN character constant 423
rollOver tests 265, 428
rounding floating-point numbers 186
- S**
- sampling
 sampleRate of member property 434
 trackNextSampleTime property 514
 trackPreviousSampleTime property 515
saving
 changes to casts 435
 movies 436, 527
Score
 associated with current movie 439
 channels selected in 440
 recording 158
 Score color assigned to sprites 439
 updating 73, 526
Score recording 73
screens
 centering Stage on 98
 identifying mouse clicks on 107
 size and position of 144
script, linked 271
script properties, scriptType of member 444
scripting Xtras 546
scripts
 assigned to cast members 444
 assigning for keys 260, 262
 assigning for mouse clicks 303, 312
 attached to sprites 460
 cast member number of frame scripts 197
 comment delimiter (--) 42

scripts (*continued*)
debugging 462
in linked movies 443
invoking handlers in 89
menu item execution of 441
objects created by parent scripts 348
rollOver tests in frame scripts 428
script number assigned to sprites 443
types of 444
searching for file names 211, 447
sending strings to browsers 167
separating items 255
servers, proxy 395
shapes
 borders of 270
 patterns for 178, 365
 types of 461
Shockwave Audio cast members
 errors and 204, 206
 pausing playback of 367
 percentPlayed of member property 368
 percentStreamed of member property 368
 play member function 377
 preLoadBuffer member command 386
 preloading 390
 specifying URL for 528
 state of 483
 stopping playback of 488
Shockwave movies
 names of external parameters 169
 number of external parameters 169
 opening 223
 playing from internet 222
 values from external parameters 170
short time format 504
size
 chunkSize of member property 104
 fixStageSize property 183
 idleReadChunkSize property 236
 lineSize of member property 270
 of cast members 104, 465
 of free blocks of memory 199
 of monitors 144
 of movies 319
 of Stage 183
 size of member property 465
sizing rectangles and points 282
slash sign (/) 46
sorting lists 466
sound
 fading in 172
 fading out 172
 playing 467, 472
 stopping playback 467, 472
 turning on or off 469
 volume control 471, 540, 541
sound cast member properties, bitRate of member 77
sound channels
 as puppets 398
 cast member 290
 cast members assigned to channels 197
 closing 467
 getting status of 251, 485
 looping 280, 281, 282
 number of samples in 433
 playing sound in 467
 playlist 457
 rewinding 424
 stopping 487
 sound levels 471, 540
 sound properties
 channelCount of member 99
 multiSound 323
 sampleRate of member 434
 soundEnabled property 469
 soundLevel property (Macintosh) 471
 volume of sound 540
 volume of sprite 541
 sound sprites, current playing time of 129
space character 473
SPACE constant 473
sprite properties
 blend of sprite 79
 bottom of sprite 81
 constraint of sprite 116
 cursor of sprite 132
 editable of sprite 155
 ink of sprite 245
 left of sprite 266
 loc of sprite 274
 locH of sprite 274
 locV of sprite 275
 member of sprite 291
 memberNum of sprite 292

sprite properties (*continued*)
moveableSprite of sprite 315
movieRate of sprite 322
movieTime of sprite 322
puppet of sprite 396
right of sprite 425
scoreColor of sprite 439
scriptInstanceList of sprite 442
scriptNum of sprite 443
setTrackEnabled 461
startTime of sprite 482
stopTime of sprite 489
stretch of sprite 494
top of sprite 511
trackCount(sprite) 513
trackEnabled 514
trackNextKeyTime 514
trackNextSampleTime 514
trackPreviousKeyTime 515
trackPreviousSampleTime 515
trackStartTime(sprite) 516
trackStopTime(sprite) 516
trackText 517
tweened of sprite 521
type of sprite 523
visible of sprite 539

sprites
active sprites 108
assigning cast members to 292
beepOn property 72
behavior scripts attached to 442
channel number of 475
clearFrame command 106
clickOn function 108
constrainV function 117
coordinates of 81, 115, 117, 266, 425, 511
counting tracks on digital video sprites 513
cue points and 253, 300
current playing time of 129
currentSpriteNum property 129
cursor resource used when pointer is over 132
custom event messages for 452
deleting frame's sprite contents 106
displaying digital video cast members in 126
editing field sprites 155
key frames and 521
media in digital video sprite tracks 517

sprites (*continued*)
mouseChar function 301
mouseMember function 309
moving 315
moving on Stage 315
on cuePassed event handler 159
on endSprite event handler 159
playing tracks on 514
position of 116, 274, 275, 474
puppet sprite blend values 79
puppets and 396, 399
registration points of 274, 275
rollOver tests 428
script number assigned to 443
scripts attached to 460
specifying location number 291
sprite keyword 473
starting time of movies in sprite channels 516
stop time of tracks 516
stretching 494
trails effect 518
visibility of 539
square brackets ([]) 49
square root function 476
Stage
centering on monitor 98
centerStage property 98
fixStageSize property 183
position of on desktop 476, 478, 480
printing 393
redrawing 527
setting color of 477
size of after loading movies 183
sprite position on 274
updating 527
starting
applications 351
character in selections 451
Score update sessions 73
statements, if...then...else 237
stopping
event messages 351
movie playback 139, 224
playback of cast members 488
sound playback 467, 472
stream status reporting 502
stretching sprites 494

string constant ("") 51
strings
 ASCII codes 102, 347
 char...of keyword 99
 chars function 101
 comparing 117
 converting expressions to 494
 counting characters in 346
 counting items in 342
 counting lines in 343
 counting words in chunk expressions 345
 date last modified 330
 deleting chunk expressions 140
 displaying in browser window 332
 do command 147
 EMPTY character constant 155
 ending character selections 451
 expressions as 494
 field keyword 172
 highlighting 226
 in description pane of Behavior Inspector 204
 in field cast members 502
 integer function 248
 item...of keyword 254
 last function 264
 length function 267
 line...of keyword 268
 numerical value of 531
 offset function 349
 put...after command 403
 put...before command 404
 put...into command 404
 quotation marks and 51
 selecting 450
 sending to browsers 167
 starting character in selections 451
 starts comparison operator 481
 writing to files 459
subtraction operator (-) 41
symbol definition operator (#) 40
symbols
 expressions and 498
 strings and 497
 symbol definition operator (#) 40
system beep elements
 alert command 59
 beep command 72
system clock, time function 504
system properties
 activeCastLib 55
 activeWindow 56
 checkBoxType 103
 deskTopRectList 144
 digitalVideoTimeScale 145
 emulateMultiButtonMouse 156
 floatPrecision 186
 frontWindow 200
 idleLoadMode 235
 idleLoadTag 236
 keyPressed 260
 multiSound 323
 platform 372
 rightMouseDown 426
 rightMouseUp 427
 stage 476

T

TAB character constant 498
Tab key 498
tabbing order, autoTab of member property 70
tempo
 assigned to frames 199
 settings 199
 tempo channels as puppets 400
text
 ASCII codes 102, 347
 char...of keyword 99
 charPosToLoc function 100
 chars function 101
 comparing strings 117
 concatenation operators (& or &&) 42, 43
 converting expressions to strings 494
 counting items in 342
 counting lines in 343
 counting number of characters in 346
 counting words in chunk expressions 345
 deleting chunk expressions 140
 do command 147
 EMPTY character constant 155
 ending character in selections 451
 expressions as strings 494
 field keyword 172
 highlighting 226
 item...of keyword 254

text (*continued*)
last function 264
length function 267
line...of keyword 268
numerical value of strings 531
offset function 349
put...after command 403
put...before command 404
put...into command 404
retrieving from files on network servers 209
returned by network operations 332
selecting 450
selecting words in 545
starting character in selections 451
starts comparison operator 481
strings in field cast members 502
text boxes for cast members 82
ticks
converting time to 504
lastClick function 265
lastKey function 266
lastRoll function 266
movieTime of sprite property 322
number of before timeout occurs 507
ticks function 504
timeoutLength property 507
timer property 509
time
converting frames to duration in time 198
converting to ticks 504
time formats 504
time function 504
units of measurement 145, 510
timeout object
determining name 326
returning 505
sending events to child object 500
timeouts, Lingo for 509
tracks, playing 461
trails effect for sprites 518
transition cast member properties, chunkSize of
member 104
transition cast members
assigned to current frame 199
changeArea of member property 98
duration of 153

transitions
between frames 199, 400
puppetTransition command 400
transitionType of member property 518
types of 518
TRUE logical constant 521
turning digital video cast member
play on or off 534, 539
turning sound on or off 469
typeface 188, 189

U

units of measurement for time 145, 510
unloading
cast members from memory 524
movies from memory 525
updating
frames 526
registration points 371
Score 73, 158, 526
Stage 527
URLs, Shockwave Audio cast members and 528

V

values, comparing 93
variables
global variables 106, 218, 462
local variables 463
property variables 393
voidP property 540
video
deleting frames from movies 142
pausing movies 118
preloading movies 390
stopping movie playback 139, 224
Video for Windows software 534

W

WAVE files 472
web pages, refreshing 88
width, of cast members 542
window properties
 activeWindow 56
 drawRect of window 150
 fileName of window 175
 modal of window 298
 name of window 326
 sourceRect 473
 title of window 510
 titleVisible of window 510
 visible of window 539
 windowList 543
 windowPresent function 543
 windowType of window 544
windows
 active window 56
 closing 109, 191
 coordinates of 415
 displaying strings in brower windows 332
 events outside of 298
 forget window command 191
 frontmost movie in 200
 moving windows behind other windows 316
 moving windows in front of other windows 317
 naming 326
 on activateWindow event handler 55
 on closeWindow event handler 159
 on deactivateWindow event handler 159
 on moveWindow event handler 352
 on openWindow event handler 352
 on zoomWindow event handler 351
 opening 352
 stage property 476
 visibility of 539
 window keyword 542
Windows computers
 beep sound and 72
 platform system property 372
 turning off 464
words in chunk expressions 345
wrapping lines 546

X

XCMDS and XFCNs (Macintosh) 353
XCOD resources 353, 464
Xlibrary files
 closing 110
 displaying Xtras and XObjects in 464
 opening 353
XObjects
 and numerical value of strings 531
 displaying in Xlibrary files 464
 opening 353
Xtras
 available to movie 346
 creating 333
 displaying in Xlibrary files 464
 name of xtra property 326
 numerical value of strings and 531
 objects created by 348
 xtra function 546

Z

zooming, allowing 63
zooming effects 548