

DIRECTOR 5

HELP CONTENTS



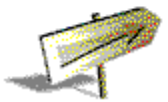
New Features



Basics



Reference



Shortcuts



Lingo Dictionary



FAQs

DIRECTOR 5

HELP CONTENTS



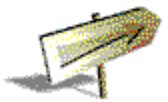
New Features



Basics



Reference



Shortcuts



Lingo Dictionary



FAQs

For information on a command, click the name of the menu in which it appears:

File Edit View Insert Modify Xtras Control Window

Or browse the alphabetical list of menu commands

For information on a window, click its name:

<u>Cast</u>	<u>Markers</u>	<u>Text Inspector</u>
<u>Color Palettes</u>	<u>Memory Inspector</u>	<u>Text</u>
<u>Control Panel</u>	<u>Message</u>	<u>Toolbar</u>
<u>Debugger</u>	<u>Paint</u>	<u>Tool Palette</u>
<u>Field</u>	<u>Score</u>	<u>Video</u>
	<u>Script</u>	<u>Watcher</u>



For information on a command, click the name of the menu in which it appears:

[File](#) [Edit](#) [View](#) [Insert](#) [Modify](#) [Xtras](#) [Control](#) [Window](#)

Or browse the [alphabetical list of menu commands](#)

For information on a window, click its name:

<u>Cast</u>	<u>Markers</u>	<u>Text Inspector</u>
<u>Color Palettes</u>	<u>Memory Inspector</u>	<u>Text</u>
<u>Control Panel</u>	<u>Message</u>	<u>Toolbar</u>
<u>Debugger</u>	<u>Paint</u>	<u>Tool Palette</u>
<u>Field</u>	<u>Score</u>	<u>Video</u>
	<u>Script</u>	<u>Watcher</u>

Menu commands

Select a command or submenu listed alphabetically below:

<u>Align</u>	<u>New Bitmap</u>
<u>Arrange</u>	<u>New Cast</u>
<u>Auto Distort</u>	<u>New Check Box</u>
<u>Auto Filter</u>	<u>New Color Palette</u>
<u>Borders</u>	<u>New Field</u>
<u>Bring to Front</u>	<u>New Movie</u>
<u>Cast</u>	<u>New OLE Object</u>
<u>Cast Member</u>	<u>New Radio Button</u>
<u>Cast Member Properties</u>	<u>New Push Button</u>
<u>Cast Member Script</u>	<u>New Text</u>
<u>Cast Preferences</u>	<u>New Window</u>
<u>Cast Properties</u>	
<u>Cast to Time</u>	<u>Open</u>
<u>Clear</u>	
<u>Close Window</u>	<u>Page Setup</u>
<u>Color Palettes</u>	<u>Paint Preferences</u>
<u>Control Panel</u>	<u>Paint</u>
<u>Control: Push Button, Radio Button, or Check Box</u>	<u>Panel</u>
<u>Control: Field</u>	<u>Paragraph</u>
<u>Convert to Bitmap</u>	<u>Paste Special</u>
<u>Copy</u>	<u>Paste</u>
<u>Create Film Loop</u>	<u>Play</u>
<u>Create Projector</u>	<u>Preferences: Cast</u>
<u>Cut</u>	<u>Preferences: General</u>
<u>Debugger</u>	<u>Preferences: Paint</u>
<u>Disable Scripts</u>	<u>Preferences: Score</u>
<u>Display</u>	<u>Print</u>
<u>Duplicate</u>	<u>Recompile All Scripts</u>
<u>Edit Cast Member</u>	<u>Remove All Breakpoints</u>
<u>Exchange Cast Members</u>	<u>Remove Frame</u>
<u>Exit</u>	<u>Repeat</u>
<u>Export</u>	<u>Replace Again</u>
<u>Field</u>	<u>Reverse Sequence</u>
<u>Film Loop</u>	<u>Revert</u>
<u>Filter Bitmap</u>	<u>Rewind</u>
<u>Find Again</u>	<u>Ruler</u>
<u>Find Cast Member</u>	<u>Run Script</u>
<u>Find Handler</u>	<u>Save All</u>
<u>Find Selection</u>	<u>Save and Compact</u>
<u>Find Text</u>	<u>Save As</u>
<u>Font</u>	<u>Save</u>
<u>Frame Palette</u>	<u>Score Preferences</u>
<u>Frame Script</u>	<u>Score</u>
<u>Frame Sound</u>	<u>Script</u>
<u>Frame Tempo</u>	<u>Select All</u>
<u>Frame Transition</u>	<u>Selected Frames Only</u>
<u>Frame</u>	<u>Send to Back</u>
<u>General Preferences</u>	<u>Show Grid</u>
	<u>Snap To Grid</u>

Grid: Settings
Grid: Show
Grid: Snap To
Ignore Breakpoints
Import
In-Between
In-Between Special
Insert Frame command
Invert Selection
Loop Playback
Marker next
Marker previous
Markers Window
Media Element: Bitmap
Media Element: Color Palette
Media Element: Text
Memory Inspector
Message
Move Backward
Move Forward
Movie Casts
Movie Properties

Sort
Space to Time
Sprite Properties
Sprite Script
Stage
Step Backward
Step Forward
Step Into Script
Step Script
Stop
Transform Bitmap
Text Inspector
Text
Toggle Breakpoint
Toolbar
Tool Palette
Transform Bitmap
Tweak
Undo
Update Movies
Video
Volume
Watch Expression
Watcher
Zoom In
Zoom Out

Using Director Help

This help system contains screens of information on using Macromedia Director. These screens are arranged in topics.

You can navigate through help topics by clicking underlined words (in green):

- Click words with a solid underline to jump to a related topic. (For example, to go to the contents screen click **Contents**).
- Click and hold words with a dotted underline to see pop-up graphics or text. For example: pop-up).

Buttons at the top of the help window help you find help topics:

- Click the **Contents** button of the help screen to see a list of help categories.
- Click the **Search** button to see an index to the Director help topics. The list on the left shows each index entry in the Help system. When you select an index entry, the list on the right shows each topic where the selected keyword is referenced. Double-click a topic to display its help screen.
- Click the **Back** button to see the previously displayed help topic. (This feature is not always available.)
- In Windows 3.1, click the **History** button to see a list of topics you have viewed. In Windows 95, use the Display History Window command in the Options menu.
- In Windows 95, click the **Print** button to print the topic you are viewing. In Windows 3.1, use the Print Topic command in the File menu.
- Click the << and >> buttons to browse backward and forward through related Director help topics. (These features are not always available.)

Acknowledgments and Copyright

This is an example of pop-up help information.

Director 5.0 Help

The Director 5.0 Help system was created by these talented individuals:

Online help expertise from Jeff Swartz
Written by Ben Melnick, Karen Olsen-Dunn, and Joe Schmitz
Designed by Diana J. Wynne
Art direction by Ilene Sandler
Art by Noah Zilberberg
Proofread by Anne Garwood

Special thanks to Dan Sadowski for engineering support and Sally Hebble for QA testing.

Copyright (c) 1994-1996 Macromedia, Inc. All rights reserved. The information in this help system may not be copied, photocopied, reproduced, translated, or converted to any printed, electronic or machine-readable form in whole or in part without prior written approval of Macromedia, Inc.

Last revised March 11, 1996

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

Select a category:

[Animation](#)

[Bitmaps](#)

[Cast](#)

[Color palettes](#)

[Digital video](#)

[Lingo](#)

[OLE](#)

[Projectors](#)

[Score](#)

[Shapes](#)

[Sound](#)

[Stage](#)

[Techniques](#)

[Tempo](#)

[Text](#)

[Transitions](#)

[Xtras](#)



Select a category:

[Animation](#)

[Bitmaps](#)

[Cast](#)

[Color palettes](#)

[Digital video](#)

[Lingo](#)

[OLE](#)

[Projectors](#)

[Score](#)

[Shapes](#)

[Sound](#)

[Stage](#)

[Techniques](#)

[Tempo](#)

[Text](#)

[Transitions](#)

[Xtras](#)



Cast

For an overview of working with casts, see:

[Cast Window](#)
[Understanding internal and external casts](#)

For information on creating cast members, see:

[Creating a new cast member](#)
[Creating text cast members](#)
[Media Element: Bitmap](#)
[Media Element: Color Palette](#)
[Media Element: Text](#)

For information on working with cast members, see:

[Dragging cast members to the score](#)
[Moving cast members between casts](#)
[Moving cast members within the cast window](#)
[Placing cast members on the stage](#)
[Placing cast members over time](#)
[Using registration points](#)

For information on cast member properties, see:

[Cast member properties submenu \(modify menu\)](#)



Score

For information about working in the score, see:

[Selecting cells](#)

[Moving selections within the score](#)

[Moving around the score](#)

[Applying color to cells](#)

[About score window inks](#)

[Turning a channel on and off \(score window\)](#)



Stage

For information about working with the stage, see:

Stage

Resizing sprites on the stage

Placing cast members on the stage

For information on changing the size and location of the stage, see:

Movie properties command (modify menu)

For information on aligning sprites on the stage, see:

Align command (modify menu)

Show Grid (View menu)

Snap to Grid (View menu)

Grid Settings (View menu)



Animation

For information on basic animation, see:

- [**Moving selections within the score**](#)
- [**In-Between command \(Modify menu\)**](#)
- [**In-Between Special command \(Modify menu\)**](#)
- [**Arrange submenu \(Modify menu\)**](#)
- [**Dragging cast members to the score**](#)
- [**Placing cast members on the stage**](#)
- [**Placing cast members over time**](#)
- [**Cast to Time command \(Modify menu\)**](#)
- [**Recording with Cast to Time**](#)
- [**Space to Time command \(Modify menu\)**](#)
- [**Step recording**](#)
- [**Recording with Space to Time**](#)
- [**Real-time recording in a channel**](#)
- [**Understanding onion skinning**](#)
- [**Update Movies command \(Xtras menu\)**](#)
- [**Converting Director 4 movies**](#)
- [**Protecting and compacting with Update movies**](#)
- [**Understanding font mapping**](#)



Bitmaps

For information on working with bitmaps in Director, see:

[Paint window](#)

[Understanding color palettes](#)

[The paint window tools](#)

[Filter submenu \(Xtras menu\)](#)

[Auto Filter command \(Xtras menu\)](#)

[Auto Distort command \(Xtras menu\)](#)

[Understanding shapes and bitmaps](#)

[Creating a new cast member](#)

[Applying a filter to a bitmapped cast member](#)



Text

For information on working with text in Director, see:

Text window

Understanding text and fields

Paragraph command (Modify menu)

Font command (Modify menu)

Creating text cast members

Editing text on stage

Importing text



Shapes

For information on working with shapes, see:

[Creating shapes](#)

[Tool palette](#)

[Understanding shapes and bitmaps](#)

[Drawing with the shape tools](#)



Digital video movies

For information on digital video movies, see:

Importing digital video movies

Digital video window

Using multiple digital video windows



Tempo

For information about setting and changing the tempo of your movies, see:

Frame Tempo command (Modify menu)



Sound

For information on using sound, see:

Importing sound files

Frame Sound command (Modify menu)

Cast Member Properties: Sound (Modify menu)

Controlling sound (Lingo)



Transitions

For information on using transitions, see:

[Cast Member Properties: Transition \(Modify menu\)](#)

[Frame Transition command \(Modify menu\)](#)

[Installing Xtras](#)



Color palettes

For information on color palettes, see:

[Understanding color palettes](#)

[Understanding color depth](#)

[Color palette window](#)

[Cast Member Properties: Palette \(Modify menu\)](#)



Projectors

For information on projectors, see:

[Create Projector command \(File menu\)](#)

[Creating Projectors](#)

[Preparing a movie for distribution](#)



Lingo

For information on specific **Lingo** elements, see the main **Lingo** topic.

For information on using Lingo in scripts, see:

Writing scripts

Types of scripts

Using variables

Event message hierarchy

Working with casts

Puppeting

Sprite properties

Handling text

Controlling sound

Managing memory

Using movie in a window

Child-parent scripts

Using XCMDs

Generating score



Xtras

For information on using and installing Xtras, see:

Using Xtras

Update Movies command (Xtras menu)

Frame Transition command (Modify menu)

Installing Xtras

Filter submenu (Xtras menu)

Auto Filter command (Xtras menu)

Using Auto Filter

Auto Distort command (Xtras menu)

Applying a filter to a bitmapped cast member



OLE

For information on using OLE objects in Director, see:

Using OLE objects in Director
OLE Object command (Insert menu)



Techniques

For information on techniques for using Director, see:

[Converting Director animations into QuickTime movies](#)

[Importing Director movies](#)

[Memory management techniques](#)

[Using registration points](#)



Understanding shapes and bitmaps

Shapes are created in Director using the tool palette. They are 1-bit vector-based graphics that are limited to basic lines, shapes, and patterns. Only text, not lines and shapes, can be converted to bitmaps using the Convert to Bitmap command. Shapes are always editable.

Bitmaps are either imported or are created in the Paint window. You cannot edit bitmaps once you create them, but you can manipulate them using any paint functions. You can create special effects by applying Photoshop filters.

Shapes animate more slowly than bitmaps, but bitmaps use more RAM and disk space. Shapes are most often used as static, invisible buttons or placeholders on the stage.



Creating shapes

Create shapes with the line, rectangle, rounded rectangle, and circle tools. As with the paint tools, clicking the right side of the tool produces a hollow shape. Clicking the left side of the tool produces a shape filled with the current color or pattern in the pattern chip.



Pressing the Shift key while dragging constrains the tool to drawing a perfect square or circle. The line tool is constrained to horizontal, vertical, or 45-degree lines with the Shift key.



Resizing sprites on the stage

If you click a sprite on the stage, a border is displayed around it with a resize handle at the border's right edge. Drag the resize handle to change the sprite's size on the stage.

Selecting cells

Drag across cells to select them. If you have Playback Head Follows Selection checked in the **Score Preferences** dialog box, the movie will advance or rewind as you select in the score. You can see the frames on the stage as you select them in the score.

Action

Shortcut

To select multiple adjacent cells

Drag across cells or Shift-click

To select all the cells in a channel

Double-click the channel number and drag down to select all the cells in adjacent channels.

To select all cells in multiple frames

Drag across the frame numbers at the top of the score.

To select all the cells in a frame

Click a frame number at the top of the score.

To select all adjacent cells that contain the same cast member

Double-click an occupied cell

For discontinuous selections

Control+click


See... Moving selections within the score



Moving selections within the score

You can move the contents of one or more selected cells to a new location within the score.

To move selected cells to a new location within the score:

1. Select the cells. The cursor changes to a hand.
2. To select additional frames, Shift-click them.
3. Drag the frames to the new location and release the mouse.

As you drag, an outline of your selection appears under the cursor. Dragging to the edge of window will auto-scroll the window. To move a copy of the selection, press the Alt key while dragging the selection.

The contents of the frames you move replace the contents of existing frames at the new location.

If Playback Head Follows Selection is checked in the **Score Preferences** dialog box, the selected frames will be displayed on the stage as you drag the selection in the score.



Moving around the score

The playback head

The playback head at the top of the score indicates which frame of the movie is currently on the stage. The playback head travels in the scratch bar as the movie plays. You can drag the playback head or click to move to a specific frame. If Playback Head Follows Selection is checked in the **Score Preferences** dialog box, the playback head travels with any selection you make in the score.

The jump button

To move your view of the score to the current frame of your movie use the jump button, located in the lower left corner of the score window. Whenever you click this button, Director brings the playback head into view and your view of the score jumps to the frame currently on the stage.

The jump to top button

If only the script channel is visible, clicking the jump to top button scrolls the score window to the top of the effects channels.

The frame counter

The frame counter pops up in the middle of the horizontal scroll bar when you drag the scroll box. It indicates how far you have advanced or rewound the movie as you drag the scroll bar.

Move forward and move backward buttons

The move forward and move backward buttons let you move a selection of sprites up or down in the score, to change their foreground priority on the stage. Move forward switches a range of cells you've selected in a channel with the cells in the channel below it. The sprites associated with the selected cells move in front of the other sprites. Move backward switches a range of cells you've selected in a channel with the cells in the channel above it. The sprites associated with the selected cells move behind the other sprites.

Shortcut: Control+Shift+down arrow for move forward, Control+Shift+up arrow for move backward.



Applying color to cells

The cell color selector lets you apply color to score cells. The color only affects the cells in the score; it does not affect how sprites appear on the stage.

To apply color to cells, select them and click a color from the cell color selector. The color is carried with the cell if you move it or copy it.

You show or hide cell colors using the Enable Colored Cells option in the **Score Preferences** dialog box. If you've already applied color to cells, hiding cell colors doesn't remove their color. Score window performance is faster if you hide cell colors.



Step-recording

Step-recording lets you create frame-by-frame animation in a channel. To activate step-recording in a channel, Alt+click the channel number in the score where you want recording to occur. Select additional channels by Alt+clicking their channel numbers.

Note: Step-record mode is automatically turned on if you drag cast members from the cast window to the score or the stage.

A red step-recording indicator appears in the channels that are in step-record mode.

When you choose **Step Forward** from the Control menu or control panel, the sprite in the channel with the step-recording indicator is copied into the next frame of your movie.

Step-recording remains in effect until you Alt+click the channel number again, click out of the channel, drag the playback head, or click Rewind or Back Step.

To step-record animation:

1. Select the place in the score where you want Director to record the animation.
2. Drag a cast member from the cast window to the stage.

A sprite corresponding to the cast member appears on the stage. The thick selection rectangle around the sprite indicates that Director is recording the sprite's position.

The step-record indicator appears next to the channel where Director is recording information about the sprite.

To record a sprite that's already on the stage, click it to select it, and then Alt+click the sprite to start recording. To stop recording, Alt+click it again.

3. Press 3 on the number pad, or click Step Forward control panel.

The movie advances to the next frame. The position of any sprite highlighted with a thick selection rectangle is recorded in the new frame.

If a sprite you're recording is already present in the next cell of the channel where you're recording its position, Director replaces the sprite's old position with its current position.

Note: Recording stops when you move the playback head in any way other than Step Forward.

4. Drag the sprite to reposition it.
You can also stretch the sprite or switch it with a sprite that corresponds to a different cast member.
5. Repeat steps 3 and 4 until you've completed the sequence you want to record.
6. Rewind the movie or click a new frame in the score to stop recording.

Move the playback head to the beginning of the animation and run the movie to see how it looks.

Recording with Space to Time

Use the **Space to Time** command on the Modify menu to move the sprites from several channels in one frame to consecutive cells in a single channel. When you create an animated sequence made up of a series of cast members, you can arrange the sprites on the stage in a single frame to see their positions in relation to one another. Once you've finished arranging the sprites, use Space to Time to convert the frame into an animation.

To use the Space to Time command:

1. Select the place in the score where you want Director to record the animation.
2. Place cast members on the stage where you want them to appear in the animation.
As you position the sprites on the stage, Director places each one in a separate channel in the score.
3. In the score, select all the sprites that are part of the sequence.
4. Choose Space to Time from the Modify menu.
The Space to Time dialog box appears. It lets you set the number of frames you want between each sprite and the one following it.
5. Type the interval you want and then click OK.
Director rearranges the sprites so that instead of being arranged from top to bottom in a single frame, they're arranged in sequence from left to right in a single channel.

If you've spaced the cells so that there are blank cells between them, you need to use one of the In Between commands to fill in the blank cells.



Space to Time is a fast way to set up the points you need for a curved In Between. Arrange the cast members in one frame, choose Space to Time from the Modify menu, and add 10 to 20 cells between each cast member to produce a smooth curve then choose In-Between Special.



Recording with Cast to Time

Use Cast to Time to move cast members to the score and arrange them in subsequent frames in a single channel.

To add a sequence of cast members to the score:

1. Select the place in the score where you want to record the animation.
2. Make the cast window active.
3. Select the series of cast members to be added to the score.
4. Choose Cast to Time from the Modify menu or hold down Alt and drag the cast members to the stage.

The selected series of cast members is added to the score.



Sometimes the series of cast members that's placed in the score jumps about the stage when you play the movie. That's because the registration points of the cast members aren't set properly. See [Using registration points](#).

Real-time recording in a channel (score window)

Real-time recording lets you record the movement of a sprite as you drag it across the stage. The real-time recording technique is especially good for simulating the movement of a cursor.

To activate real-time recording in a channel, press Control-Spacebar while clicking a cell in the desired channel. To activate real-time recording in the first empty score channel, just press Control-Spacebar.

An indicator appears next to the channel number to indicate that real-time recording will occur in that channel. Only one channel at a time can be used for real-time recording.

To record in real time:

1. Select the place in the score where you want Director to record the animation.
2. In the cast window, select the cast member you want to animate.

Make sure you select a cast member in the cast window rather than a sprite on the stage. Also, don't drag the cast member from the cast window to the stage--just select it.

To record a specific range of frames, select the frames, and then click the Selected Frames Only button on the control panel.

3. Hold down the Control key and the spacebar.

The real-time record indicator appears next to the channel where you're recording.

Recording begins as soon as you click the mouse button in the next step, so be prepared to move the mouse.

4. Point to the spot on the stage where you want the animation to start, press the mouse button to begin recording, and drag the pointer across the stage.

Director records the path along which you move the pointer.

If there are sprites in any of the other channels in the first frame of the animation, Director records their positions in each frame where it records the position of the sprite you're moving.

5. Release the mouse button to stop recording.



- To have more control while recording in real time, use the **tempo control** in the control panel to record at a speed that's slower than normal.
- If you select **Trails** in the score, you can also use real-time recording to simulate handwriting.



Turning a channel on and off (score)

Turning a channel off tells Director to ignore the channel during playback. By default, all channels are active. Clicking the button next to a channel turns the channel off, causing Director to ignore that channel when you play the movie.

If you turn the script channel off, Director ignores all scripts during playback. (This is the same as checking the **Disable Scripts** command in the Control menu.)

Channel on/off settings are not saved with the movie.



Understanding internal and external casts

You can create multiple casts for your Director 5.0 movies. There are two types of casts, internal and external. Prior to version 5.0 of Director, all casts were internal except Shared.DIR and only one cast was allowed per movie.

Working with cast members within the cast window is the same regardless of whether the cast is internal or external.

Internal casts

When you create a new movie, Director automatically creates an internal cast. Internal casts are stored inside the movie file. When you save a movie, all internal casts are saved. When you create a projector, they are stored inside the projector file. Internal casts cannot be shared by other movies.

External casts

External casts are stored outside of the movie file. They can be shared between movies or serve as libraries for commonly used movie elements. They are also useful for distributing work in a project team.

When you create an external cast by choosing New Cast from the File menu, you have the choice of linking or not linking the cast to the current movie.

- If you link an external cast to the current movie, Director opens the cast every time you open the movie. If it can't find the cast in the original location, it prompts you to locate the cast file. When you save a movie, all linked external casts are saved as well. The first time you save a movie with a linked external cast file, Director prompts you to enter a file name and choose a location for the file.
- If you don't link an external cast to a movie, you must open it separately with the Open command on the File menu, and save it by activating the window and choosing Save from the File menu. It is not saved along with the movie when you choose Save. If you move a cast member to the stage or score from an unlinked external cast, Director prompts you to link the cast to the current movie.

You can link and unlink existing casts to the current movie with the **Movie Casts command** on the Modify menu.

- **Cast window**
Save command (File menu)
New Cast command (File menu)
Moving cast members between casts



Moving cast members within the cast window

You can rearrange cast members within the cast window by dragging them to a new location. When you select a cast member the pointer changes to a hand shape. Shift-click to select additional adjacent cast members; Control-click to select additional non-adjacent cast members.



Moving cast members between casts

You can move cast members between casts by dragging them, just as you would move them within a cast window.

- When you drag a cast member between internal and linked external casts, the cast member is removed from the source location and placed in the destination.
- When you drag a cast member to or from an unlinked external cast, Director places a copy in the destination and the original remains where it was.
- Alt-drag a cast member to copy it into a new cast window.

• Understanding internal and external casts Moving cast members within the cast window



Placing cast members on the stage

Placing cast members on the stage adds them to the score in the available channels of the current frame. If you place more cast members than you have available channels of the current frame, Director warns you that it will ignore the extra cast members.

When you place cast members on the stage (or in the score), Director adds them as sprites. A sprite is an image of the cast member on the stage or in the score that contains information about the cast member at one point in time.

Select a cast member in the cast window. Shift-click to select additional adjacent cast members; Control-click to select additional non-adjacent cast members. Drag the selection to the stage and release the mouse.

As you drag the cast members over the stage, Director displays an outline indicating the size of the area enclosing all the cast members. (If you drag a sound or a palette, it does not display an outline on the stage, as these cast members have no size dimensions.)

When you drop the cast members on the stage, **step record** is enabled in the channels where the cast members were dropped.

Shortcut: Press Control+Shift+L to place selected cast members in the center of the stage.



Placing cast members over time

By default, when you drag cast members from the cast window, Director places multiple cast members on the stage so that they all occupy the same frame in the score. To place cast members on the stage over time instead of in the same frame, press Alt while dragging the cast members from the cast window to the stage.

As you Alt+drag the cast members over the stage, Director displays an outline indicating the dimensions of the first visual cast member in the selection, which will be placed in the current frame. The rest of the cast members in the selection are placed into adjacent frames.

If the score window is open while you are dragging cast members to the stage, Director outlines the cells in the score where the cast member selection will be dropped. Director restricts you to dropping cast members only into unoccupied cells in the score. (If you instead drag the cast members directly from the cast window into the score, Director does not prevent you from dropping the cast members into occupied cells and overwriting existing sprites.)

When you drag a cast member to the stage, Director uses the first empty cell in the current frame that is closest to the current score selection. (If you drag the cast member directly into the score, you can more precisely specify the cell that will contain the cast member.) You can force Director to use a certain set of empty cells by first selecting them in the score before dragging in the cast members.

Shortcuts:

- Press Control+Alt+L to place selected cast members over time in the center of the stage.
- You can also place selected cast members across time in the score by choosing **Cast to Time** from the Modify menu.

Dragging cast members to the score

Dragging cast members into the score places them into a range of cells and starts step-record mode in the channels where the cast members are dropped.

To place one or more cast members into the score:

1. Click to select the first cast member in the cast window.
2. Shift-click to select additional adjacent cast members; Control+click to select additional non-adjacent cast members.
3. Drag one of the cast members you just selected to the score. Dragging any selected cast member drags the entire selection.

As you drag the cast members over the score, Director outlines the range of cells into which the cast members will be dropped.

Director restricts you to specific regions of the score depending on the type of cast members in the selection. For example, if you drag a sound cast member into the score, Director restricts you to the two sound channels. If you are dragging different types of cast members, Director lets you only drop them into cells of the appropriate type.

You can drag this type of cast member	Into this part of the score
Bitmap, PICT, Text, Field	Sprite channels 1-48
Button, Shape, Film loop, Movie, Digital Video Sound	Sound channel 1 or 2
Palette	Palette channel
Script	Script channel. (Not a movie script.)

When you drop cast members into the score, Director fills the highlighted cells in sequential order by cast number. The first cast member in the selection of a particular type will be dropped into the left-most or top-most cell in the channel.

Any occupied cells in the highlighted region are overwritten by the cast members you drag in.



Creating a new cast member

To create a new cast member, click the Add button. The keyboard shortcut for Add is Control+Shift+A.

Director creates a new window in which you can create the cast member. Whatever you create is placed in the first available cast member position.

The previous and next buttons control which cast member is displayed.



Using rulers

The paint window has vertical and horizontal rulers to help you align and size your artwork.

To use the rulers, choose Rulers from the View menu. The rulers appear at the top and left side of the paint window.

The default unit of measurement is pixels. To change the unit of measurement, click the corner where the rulers meet. With each click the unit of measurement changes.

To change the location of the zero point, drag right or left along the ruler at the top of the window or up or down along the ruler at the side.



Zooming in and out (paint window)

The Zoom commands on the View permit you to zoom in or out at four levels of magnification.

To zoom in on the image in the paint window:

- Choose **Zoom In** (Control+plus sign) from the View menu to zoom in one degree of magnification.

To zoom in on a particular feature of the image, Command-click the image, or position the pointer over the feature before choosing **Zoom In**.

- Choose Zoom from the **View menu** and select a degree of magnification from the submenu.

You can see a full-size view of the image in the box in the upper right corner of the paint window. You can go back to the full-size view any time by clicking inside the box.

Zoom Out reverses the direction of the zoom. To reduce a magnified view, choose Zoom Out (Control+minus) from the View menu.

Shortcut: Switch back and forth between normal size and the last level of magnification you chose by double-clicking the pencil tool or by Command-clicking with any tool.

● **Selecting colors and patterns (paint window)**

Colors and patterns are similar in the way you choose them and in the way you make use of them. You choose both colors and patterns from pop-up palettes (the pop-up colors palette isn't available if you have your monitor set to black and white), and you make use of both with the same set of tools.

The colors that appear on the pop-up palette come from a set of colors known as a palette. A Macintosh or Windows computer is capable of displaying over 16 million colors, but it can only display a given set of colors at any one time, depending on the capabilities of your video card. If the computer you're working with has a 4-bit video card, you can work with palettes of 16 colors. If you're working with an 8-bit video card, you'll be able to work with palettes of 256 colors.

Director has nine standard palettes: two System palettes (for Macintosh and Windows systems), Rainbow, Grayscale, Pastels, Vivid, NTSC, Metallic, and VGA palettes. (The standard palettes aren't all available all the time; which palettes are available depend on the number of colors your monitor is set to display.) The System palette for your platform is the default palette. You can create as many additional palettes as you want.

There are three types of colors you can choose from the current palette: The foreground color is the color that you paint with when the pattern is solid and the ink is Normal. The background color is the secondary color in a pattern. The destination color is the color you want to replace the foreground color with when you use the Switch ink or the Switch Colors button; when you use the Cycle ink, the destination color is the end of the range of colors (beginning with the foreground color) that you want to cycle through; and when you use the Gradient ink, the destination color and the foreground color define the two extremes of the range that make up the color gradient.

Director has three standard palettes of patterns--Grays, Standard, and QuickDraw--as well as a custom palette. The custom palette contains a set of default patterns; you can create new patterns by editing the default patterns in the Pattern Settings dialog box. You can also create a tile--a multicolored pattern that's a duplicate of a small rectangular section of an existing cast member.

● **Drawing with the shape tools**

The shape tools are the line, rectangle, circle, and polygon tools. There are two tools each for the rectangle, circle, and polygon tools: hollow version and a filled version. When you use the hollow version, the shape you draw is only an outline; it is not filled with a pattern or color. When you use the filled version, the shape you draw is filled with the currently selected foreground and background colors and patterns.

If you press the Alt key while drawing with one of the shape tools, the border of the shape is drawn with the current pattern.

If you press the Shift key while drawing with the line tool, the line is constrained to horizontal, vertical, or 45-degree angles.

Double-clicking the shaded side of the rectangle, circle, or polygon tool opens the **Gradients**.

● Using registration points

When you create a cast member in the paint window, it is automatically assigned a registration point in the center. You can see this by creating a simple cast member and clicking the registration tool. When you click the tool, dotted lines appear in the paint window. The intersection of these dotted lines is the registration point of the cast member. Using registration points speeds your ability to quickly and accurately put cast members on the stage and have them all line up with the same point.

The registration tool is used to line up cast members for frame-by-frame animation. When you have a series of cast members, you can align their registration points so you have a fixed reference point for animation. A simple example might be the hands of a clock. The hands are made up of different cast members at various positions around the face of the clock, but they must be anchored to the center of the clock. Setting a registration point on the hands at the point about which they rotate enables them to line up in the proper position on the stage.

Another example of using registration points is a running figure. You can set a registration point at the same spot on the ground so when the series of running figures is animated, they bounce up and down relative to the same point on the ground.

To set registration points:

1. Open the paint window to the first cast member in the series.
2. Select the registration tool.
3. Adjust the registration point or reset it to the default setting by dragging the crosshair outside the window and releasing the mouse button.
4. Repeat this process until the registration points of all the cast members in the series have been reset.

● **Creating tiles from a cast member**

You can use tiles to make multi-colored patterns. There are eight default tiles that are always available for you to use, or you can create your own. The tiles you create are stored with the Director movie they were created in.

Once created, the tile appears in the patterns pop-up menu and can be used like any other pattern. The current foreground and background colors have no effect on the color of the tiles but switching palettes changes the color of the tiles.

To create a tile:

1. Click the Pattern chip and choose Tile Settings from the bottom of the pop-up.
The radio buttons determine whether the tile you select is built-in or made from a cast member.
2. Click Cast Member.
This radio button is dimmed if all the cast members in the movie are 1-bit cast members. Tiles can be made only from cast members with color depths greater than 1 bit.
3. Click the left or right arrows to choose a cast member.
As you click the arrows, the cast members in the paint window appear in the left side of the Tiles dialog box.
4. Choose a size using the Width and Height pop-up menus.
As you choose a width and height for your tile, the dotted box on the left side of the Tiles dialog box changes shape to indicate the tile's size. You can also drag the dotted box to make a tile from a different part of the cast member.
5. Click OK.
The tile you made is displayed in the bottom row of the pattern palette.

You can choose and use the tile just as you would select and use a pattern.

- **Creating text cast members**

There are two main ways to create text cast members:

- Create text cast members directly on stage using the **text tool** on the tool palette. Click the text tool and then drag the pointer on stage to define the width of the text. When you release the mouse button, a text insertion point appears in the area you just defined and you can begin entering text. The new text cast member is placed in the first available position in the cast. The sprite is assigned to the first open score cell in the current frame.

- Create text cast members in the text window by choosing Text from the New Media submenu in the Insert menu or by clicking the text window tool on the toolbar. Text you enter appears in the first available cast position, but it is not automatically placed on the stage.

You can also create text cast members by importing text in the rich text format (RTF). When importing text, Director creates a new cast member each time it encounters a hard page break or a section break in the file. A cast member can include as much text as memory allows.

- **Understanding text and fields**

Importing text

Text Cast Member Properties

● **Understanding text and fields**

There are three ways of working with text in Director. Each has distinct advantages for different applications.

Rich text

Rich text is the type of text you create with the **text tool** or in the **text window**. It is best suited for most applications and provides many features that were not possible in earlier versions of Director.

- It offers paragraph formatting and definable tabs for each separate paragraph in a cast member.
- It is always editable in the authoring environment, but when you create a projector it is bitmapped. Bitmapped text offers the best performance in animations and does not require that users have the same fonts installed on their systems.
- Large fonts can be anti-aliased to improve on-screen appearance. No jagged lines appear at corners or along angles. See anti-alias in the **Text Cast Member Properties** dialog box.
- You can import from any application that stores text in the standard rich text format (RTF). When importing text Director creates a new cast member each time it encounters a hard page break in the file. See **Importing text**.

The text cannot be edited by users in playback or projectors, and it prints at 72 dpi.

Fields

A field is the type of text you create in any of the following ways:

- With the field tool on the tool palette
- By choosing Field from the Controls submenu of the Insert menu
- In the field window, which is opened by choosing the Field command from the Window menu.

In general, you should only use fields if you need text that is editable in playback or projectors, or you need the best available print quality. Prior to version 5.0, fields were the only type of editable text available in Director. The disadvantages of fields are:

- The formatting of text in fields is controlled by the system software, so users must have the same fonts installed on their systems when they play the movie or projector.
- Paragraph formatting and tabs are not available.
- Fields can slow down animation.

Bitmapped text

Bitmapped text is text you create in the paint window, or import from other image editing and paint programs. You cannot edit bitmapped text once you create it, but you can manipulate it using any paint functions. You can, for example, create dramatic effects by applying Photoshop filters to bitmapped text. The print quality of bitmapped text is usually poor.

- **Importing text**
Creating text cast members

Text Cast Member Properties

- **Editing text on stage**

You can edit text directly on stage without opening the text window. Click a text sprite once, and you can move the sprite on the stage just like any other sprite. Click twice to make an insertion point appear within the text. Once the insertion point is visible, you can enter or edit text just like you would in the text window. When you make a change, Director updates all instances of the text cast member.

Change the width of a text or field cast member with the resize handle on the right border.

- **Creating text cast members**
Text Cast Member Properties

- **Importing text**

You can import text from any application that saves text as plain text or in the rich text format (RTF). Choose Import from the File menu, choose Text from the Type pop-up menu, and then double-click the file you want to import.

When importing rich text, Director creates a new cast member each time it encounters in a hard page break or a section break in the file. A cast member can include as much text as memory allows. Director assigns the name of the file you import to the cast member name.

- **Creating text cast members**
Understanding text and fields
Text Cast Member Properties

- **Applying color to text**

You can change the color of selected text using the pop-up color palette in the Font dialog box. To change the background color of text, select the text cast member on stage and then choose a color with the background color chip on the tool palette. This changes the color of the entire cast member. You can't choose more than one background color for a text cast member.

- **Creating a digital video cast member**

To add a digital video movie to the cast, you either paste it, import it, or choose Digital Video from the New Media Element submenu in the Insert menu. Director automatically links the movie to the cast and adds it to the cast window. If you paste the movie, Director first asks you to name the file in which to store the movie.

- **Using multiple digital video windows**

You can create as many digital video windows as memory allows and play them simultaneously. A movie continues playing even if it is not the front-most window.

You can also create multiple views of the same digital video window, to display the same cast member in separate windows. This is useful for editing the movie, since you can play the movie independently in each window, and cut and paste between each window.

Changes to the cast member in one window are automatically reflected in any other windows that display the cast member.

- **About score window inks**

Some score window inks work with both black and white and color artwork and others work only with color. Inks that only work with color are: Bkgnd Transparent, Blend, Darkest, Lightest, Add, Add Pin, Subtract, and Subtract Pin.

To apply an ink, select the cells in the score and then choose the ink from the ink pop-up menu.

When dragging a cast member to the stage, the default ink is the last ink you chose in the pop-up menu.

The default ink is Copy. When you use Copy, a bounding box appears around the sprite. The bounding box is invisible when displayed on a white stage, but when the sprite passes in front of another sprite, or the stage is black, you will see the bounding box.

To eliminate the bounding box that appears around a sprite, select the sprite in the score and choose the Matte ink. Use it sparingly though, because it slows down screen redraw and uses more memory than other inks.

To change the ink for a sprite, select the cell or cells, and choose an ink from the Ink pop-up menu. Your choice replaces the previous ink assigned to the sprite.

Choose an ink carefully, since some inks can result in decreased performance when animating a cast member. For example, the following inks are listed in order of decreasing performance: Copy, Matte, Background Transparent, and Blend.

Note for Adobe Type Manager users: The following inks produce jagged text if you have Adobe Type Manager installed: Not Transparent, Not Reverse, Not Ghost, Blend, Darkest, Lightest, Add, Add Pin, Subtract, and Subtract Pin.

- **Ink pop-up menu**

- ◆ **Importing digital video movies**

Importing a QuickTime or Video for Windows (AVI) movie automatically links the file to the current movie and adds it to the cast window.

Any changes you make to the digital video movie with an editor outside Director are reflected in the linked file.

Note: The imported digital video movie's contents do not become part of the Director movie. If you make a copy of the Director movie, be sure to include all digital video movies that are part of the movie.

- ◆ **Video Window**
Digital Video Cast Member Properties

● **Installing Xtras**

You can add three types of Xtras to Director: filters, transitions, and new cast member types.

Xtras are available from third-party developers. You can install Photoshop filters in Director as Xtras. You can also create your own Xtras if you can program in C.

To install an Xtra, copy the file to the Xtras folder inside the Director application folder. If you use more than one Macromedia product, you should store Xtras in an Xtras folder in the Macromedia folder in the Windows folder. You can create folders inside the Xtras folder and Director will still find them.

- ◆ **Importing sound files**

Imported sounds are added to the cast window. You can import the following sound files:

- ◆ On the Mac, you can import System 7 sounds, AIFF files, and AIFC files. Compressed System 7 sounds can only be played on Macintosh computers.
- ◆ In Windows, you can import WAVE, AIFF files, and AIFC files. Compressed WAVE sounds can only be played on Windows computers.

Dialog box options

File displays sounds stored as files.

Resource displays sounds stored as resources in applications.

Linked lets you create a link to an AIFF or AIFC file on disk. Additionally, in Windows, you can link to a WAVE sound file. You must create a link to a compressed sound file.

- ◆ **Sound Cast Member Properties**
Frame Properties: Sound

● Understanding color palettes

For an entertaining, full-color demonstration of how color palettes affect your work in Director, run the Color Palettes lab from the Director CD.

When your system is set to 8-bit color depth (256 colors) or less, it uses a limited set of colors called a color palette. The color palette determines all the colors that can be shown on the screen at the same time. This includes not only the colors for the images you're working with, but system elements like title bars, dialog boxes, tools, and so on.

If your system is set to display 16-, 24-, or 32-bit color (thousands or millions of colors), color palettes don't affect Director movies; they serve only as a means for selecting colors in the paint window. Because they refer directly to a complete spectrum of colors, 16-, 24-, and 32-bit graphics do not require color palettes to achieve accurate color.

Color graphics with 2-, 4-, or 8-bit color depth don't store any information about the hue, saturation, or brightness of any particular color. They identify colors by referring to positions in the current color palette.

Where palettes come from

When working with 2-, 4-, or 8-bit graphics, many graphics programs create special palettes with the best colors to display a particular image.

When you create 2-, 4-, or 8-bit graphics in any application, the active palette is linked to the file. This palette must be active for the graphic to display the right colors.

Director includes several palettes. The system palettes are the default selections. Any additional palettes you create or import appear as cast members.

Conflicting color palettes in Director movies

Only one palette can be active on your computer at a time. This causes problems in Director when you try to put two bitmapped cast members on stage at the same time that have different palettes; the colors for one of the images will be wrong.

While a movie is playing, the palette channel in the score determines the active palette. When the playback head reaches a frame containing a new palette, it changes the active palette. To make this change using Lingo, see the **PuppetPalette command**.

When a cast member you are placing on stage has a palette different from the currently active palette, Director adds the new palette to the palette channel. The new palette becomes the active palette, and it remains in effect until you set a different palette in the palette channel. For more information about using the palette channel, see **Frame Palette command**

Solving color palette problems

Here are some guidelines for solving problems caused by conflicting color palettes:

- Make sure all cast members on stage at the same time refer to the same palette. This is essential.

- Simplify your work and avoid frequent palette changes by mapping all the images in your movie to as few palettes as possible.
- If possible, create a palette that contains all the colors you need in your movie. You can do this within Director by modifying an existing palette in the color palette window. You can copy and paste colors from one palette to another. For information on using the color palette window, see **Color palettes window**. Outside of Director, you can use a conversion utility named DeBabelizer by Equilibrium software (Macintosh only) to scan groups of images, create an optimal palette, and then remap all of them to the new palette. Some image editing programs also have similar features
- Remap existing cast members to a new color palette using the Transform Bitmap command. See the next section, "Changing the palette of existing cast members."
- As you import cast members, you can remap them to new palettes using the **Image Options**.
- If you don't understand what has been discussed so far, or if you're using images with simple colors, you can avoid all of these complexities by using Transform Bitmap to remap all of your images to the Windows or Macintosh system palette.

Changing the palette of existing cast members

You can remap existing bitmap cast members to different color palettes with **Transform Bitmap**.

- **Understanding color depth**

• **Understanding color depth**

The number of colors a graphic image or a computer system can display is called the color depth.

Color depth is expressed as the number of bits used to specify the color of each pixel, or as the number of colors that can be displayed at once.

Bit Depth	Number of colors
1-bit	Black and white
2-bit	4
4-bit	16
8-bit	256
16-bit	32,768
32-bit	16.7 million

There are several reasons to use 8-bit graphics in your movies:

- Many low-end systems support only 8-bit graphics. If you intend to distribute to a wide audience of low-end users, 8-bit is the safest choice.
- Graphics with lower color depth require less storage space, less memory to display, and animate faster. Graphics with 16-bit color depth or more contain so much information about color that they rapidly use up all available memory.
- Director for Windows 3.1 supports only 8-bit graphics. (Director for Windows 95 supports higher bit depths.)

To change the color depth of bitmap cast members:

1. Select the cast member you want to change.
2. Choose Transform Bitmap from the Modify menu.
3. Choose a new depth setting from the color depth pop-up and then click Transform.

To change the color depth of a movie:

1. Change your monitor's color depth setting in the control panel of your system software.
2. Click Save in the File menu to save the movie or Save As to save the movie with a new name.

• **Understanding color palettes** **Image Options**

● **Applying a filter to a bitmapped cast member**

You can apply a filter to an entire bitmapped cast member, or to a selection in the paint window.

To apply a filter:

1. Open the cast member in the paint window, or select the cast member in the cast window.

You can apply a filter to several cast members at once by selecting them all in the cast window. To apply a filter to a selected portion of a cast member, use the selection rectangle or the lasso in the paint window to select the part you want to change.

2. Choose Filter Bitmap from the Xtras menu.
3. In the Filter Bitmap dialog box, choose a category on the left and a filter on the right. You can also choose All in the category list to view all the filters at once.
4. Click Filter.

Many filters require you to enter special settings. When you choose one of these filters, a dialog box or some other type of control appears after you click Filter.

Some filters have no changeable settings. When you choose one of these, the cast member changes with no further steps.

- **Using OLE objects in Director**

You can place OLE objects in Director movies as cast members. An OLE cast member is treated as a bitmap inside of Director. For this reason, you should use OLE objects primarily for images. Sound and video OLE objects are not effective.

- When you create an OLE object, it's created as an 8-bit cast member. Use the **Transform Bitmap** command to change the bit depth. As you update from the server, the OLE object changes in the cast to the new bit depth.
- Double-clicking an OLE cast member or sprite launches the OLE server so that you can edit it. When you make changes in the OLE server and choose Update from the File menu, the OLE object is automatically updated within Director.
- When you edit an OLE object, Director uses the standard cross-hatching to indicate the object is in use by the server. When you quit the server, the cross-hatching disappears.
- OLE objects are converted to bitmaps when you create a projector. They are no longer linked to their source applications.
- OLE objects remain external files. Be sure to include all OLE files when you move a movie or create a projector.
- OLE objects work only in Windows 95 and Windows NT.

- **OLE Object command (Edit menu)**

● Using Auto Filter

Use Auto Filter to create dramatic animated effects with bitmap filters. Auto Filter applies a filter incrementally to a series of cast members. You can use it either to change a range of selected cast members, or to generate a series of new filtered cast members based on a single cast member. You define a beginning and ending setting for the filter, and then Auto Filter applies an intermediate filter value to each cast member.

To use Auto Filter:

1. Select a bitmapped cast member, or a range of cast members, and then choose Auto Filter from the Xtras menu.

To change only a portion of a bitmapped cast member, use the selection marquee or the lasso in the paint window to select the part you want to change.

2. In the Auto Filter dialog box, select a filter.
3. Click Set Starting Values and use the filter controls to enter filter settings for the first cast member in the sequence.

When you finish working with the filter controls, the Auto Filter dialog reappears.

4. Click Set Ending Values and use the filter controls to enter filter settings for the last cast member in the sequence.
5. Enter the number of new cast members you want to create (the box is not available if you have selected a range of cast members).
6. Click Filter to begin the filtering.

A message appears to show the progress. Some filters are very complex and require extra time for computing.

- **Understanding onion skinning**

Related topics:

Tracing a cast member

Creating new cast members using a series of previous cast members as reference images

Drawing a series of images by using another series of images as reference images

Onion skinning and registration points

Onion skinning derives its name from a technique used by conventional animators who draw on very thin "onion skin" paper so that they can see one or more of the previous images in the animation.

Onion skinning in Director allows you to create or edit animated sequences of cast members in the paint window using other cast members as a reference. Reference images appear dimmed in the background. While working in the paint window you can view not only the current cast member that you're painting, but one or more cast members blended into the image. For a description of the onion skin toolbar, see **Onion Skin (View menu)**.

You can use onion skinning:

- To trace over a image, or to create a series of images all registered with a particular image.
- When drawing each cast member of an animation, to see previous images in the sequence and use those images as a reference while you are drawing new ones.
- To create a series of images based on another "parallel" animation. A series of images serves as the background while you paint a series of foreground images.

● **Tracing a cast member**

Follow these steps to trace a new cast member using another cast member as a background image:

1. Open the paint window and select Onion Skin from the View menu.
2. Open the cast member in the paint window that you want to use as the reference image or background.
3. Click Toggle Onion Skin (the left-most button in the toolbar) to turn onion skinning on.
4. Click Set Background in the toolbar to set the background image.
5. Click the Add button ("+") in the paint window to create a new cast member.
6. Click Show Background in the onion skin toolbar.

The original cast member now appears as a dimmed image in the paint window. All painting operations now take place "on top of" the original cast member's image.

7. Paint the new cast member using the background image as a reference.

- **Creating new cast members using a series of previous cast members as reference images**

You can create new cast members using other cast members as reference images. Use the Preceding and the Following Cast Members buttons in the onion skin toolbar to help you step through a series of cast members created for your animation.

- You can set Following Cast Members to display reference images following the current cast member. Use the Preceding and Following Cast Members settings to help you step through a series of cast members created for your animation.

In the following illustration, the Preceding Cast Members field is set to 2 so that images of two previous cast members appear in the background as a new cast member is painted in the foreground.

● **Drawing a series of images by using another series of images as reference images**

Follow these steps to use a series of images that serve as the background while painting a series of foreground images.

1. Arrange the series of cast members you want to use as your background, in consecutive order in the cast.
Cast members in each of the foreground and the background series must be adjacent to each in the other in the cast, otherwise Director cannot track the background properly as you add cast members to the foreground.
2. Open the paint window and select Onion Skin from the View menu.
The onion skin toolbar appears.
3. Click Toggle Onion Skin in the onion skin toolbar to turn onion skinning on.
Make sure all values in the onion skin toolbar are set to 0.
4. Open the cast member in the paint window that you want to use as the first background cast member in the reference series and click Set Background.
5. Select the position in the cast that you want the first cast member in the foreground series to appear and click the Add button ("+") in the paint window to create a new cast member.
The first cast member in the foreground series can be located anywhere in the cast.
6. Click Track Background in the onion skin toolbar.
The corresponding member of the reference series appears as a reference image.
7. Paint the new cast member using the background image as a reference.
8. When you have finished drawing the cast member, click the Add button ("+") again to create the next cast member.
When Track Background is enabled, Director advances to the next cast member in the reference series and its image appears in the background in the paint window.
9. Repeat step 8 until you have completed drawing the cast members in the series.

- **Onion skinning and registration points**

The current cast member and the reference images must be correctly registered with each other. All images must be drawn with their registration points aligned.

By default, the registration point of a bitmap cast member is always the center of the image. Director freezes the registration point of a bitmap cast member once it has been used as a foreground or background image during onion skinning. For more information about using registration points see, **Using registration points**.

● **Linking to a file**

When you import a PICT file, AIFF sound file, or a Director movie, you have the option to create a link to the file in the cast rather than copying the contents of the file into your movie. This allows you to add a PICT, sound, or Director movie to the current movie without increasing the size of the movie. It also allows you share the same linked file among several movies.

If you change the linked file, the changes are automatically reflected when you reopen your movie. Director looks for any linked cast members when you open the movie. If it cannot find a linked file, a dialog box appears so you can locate the file.

Note: Director always imports digital video movies by linking to the file.

● **Importing PICT files**

If you import a PICT file, Director converts it to a bitmapped cast member and adds it to the cast.

The color depth of an imported PICT image depends upon the color depth of the monitor on which Director's stage appears. For example, if the stage appears on a second monitor when you import the image, the second monitor's color depth determines the image's color depth.

Importing the cast member's palette ensures that the cast member looks as it did when it was originally created.

Remapping the PICT file causes the cast member to appear in the colors closest to the colors in the current palette.

Dialog box options

Linked creates a link to the location of the PICT file on disk. This is useful if you want to add large (24-bit) images to your movie without increasing the movie's size.

Note: If you edit a linked PICT file, changes are reflected in the file on disk when you save the movie. For example, if you import a 24-bit PICT file by linking to the file, and later change its color depth using the Transform Bitmap command, Director alters the bit depth of the PICT file on disk when you save the movie.

● **Importing Director movies**

Importing a movie adds the movie's contents (cast members, scripts, sounds, palettes, text, artwork) to the cast window. Director converts the movie to a film loop in the cast, imports all the movie's cast members, and updates their references to reflect their new positions in the cast. The following items are not imported: movie script, resources, tempos, transitions, and markers in the score. Score scripts are imported but are not activated when you drag the film loop onto the stage.

Note: You cannot link to a movie created with a previous version of Director. To link the movie, first update it to Director 5.0's file format using the **Update Movies** command in the Xtras menu.

Dialog box options

Linked lets you create a link to the movie in the cast window. This option is useful if you want to add a movie to your current movie without increasing the file size of your current movie. The movie is imported as a film loop, without the individual cast members. If this box is not checked, the movie itself will be imported into the cast.

● **Converting Director animations into QuickTime movies**

You can export a Director animation as a QuickTime movie, and then import it into Director so that it becomes a single cast member. You might want to do this for the following reasons:

- Since an imported digital video movie is linked to its source file on disk, you can edit the digital video movie in another application, and your changes will be automatically reflected in the digital video movie inside Director.
- Director provides precise control over the digital video movie's playback, using the Digital Video Cast Member Properties dialog box.
- Other users can use and distribute the QuickTime movie for use in any application that supports QuickTime.

You might not want to convert a Director animation into a digital video movie if the animation includes interactivity. (Digital video movies do not allow interactivity.)

Notes:

- If you lose your transitions when you export a Director movie as a digital video movie, try increasing the Duration and Smoothness settings for the transitions before you export the movie.
- If the exported digital video movie plays before your transitions occur, turn off the digital video movie cast member's Direct to Stage setting in the Cast Member Properties dialog box.

● **Memory management techniques**

You use the same basic memory-management techniques on both Windows and Macintosh:

- Be conservative with memory-intensive cast members such as large internal sounds and bitmaps. If possible, use external sounds instead of internal sounds.
- Keep animations as small as possible. If cast members are too big, you get a memory error message when the movie is played back.
- Create movies in several smaller segments instead of one large movie whenever practical. Smaller segments are more likely to play back without pausing, and can begin and end at logical breaks in the movie.
- Use trails to leave sprite images on stage if the sprite doesn't move and uses no transitions.
- Determine the minimum free memory that the movie will be required to use, and then work backward from that minimum to make sure that your movie fits into available memory. For example, you should take into account all global variables and objects, and determine how much memory they will require. After subtracting this amount from the projector size set by the system, you will know the amount of memory that remains for both sound and graphic cast members in any one frame.
- Determine the available memory before playing a movie by entering `put the memorySize` in the message window. You can also use `put the freeBlock` to see what's happening to memory while you play the movie. You can also use the `traceLoad` command to display cast member names and other information about cast members as they are loaded into memory.
- Using Lingo, include a check of the `freeBlock` before the movie starts to determine if the minimum memory necessary to run is available. If not, display an alert informing the user how much more memory is necessary.
- To determine how much memory a frame of a movie uses, use the `ramNeeded` Lingo function.
- Use Cast Member Properties to tell Director when to purge a cast member from memory. Make sure large cast members have an appropriate purge priority.
- Load cast members into memory on a "when needed" basis (specified using the Cast Properties command on the Modify menu).
- If memory on the target machine is a consideration, create separate versions of your Director movie, and design each one to run on machines with 5 megabytes, 8 megabytes, or 16 megabytes of physical RAM. Create a separate movie that checks the available RAM and then calls the appropriate movie version.
- During authoring, use the Lingo properties the `traceLoad` and the `traceLogFile` to determine when cast members are used in your movie. Set the cast member's unload option in Cast Member Properties to make sure cast members that are needed frequently don't get purged from memory.

Predicting bitmap size in memory

Most cast members are the same size in memory as they are on disk. However, bitmaps can change size when they are loaded into memory, depending on the color resolution of the playback monitor.

A 1-bit color depth image remains a 1-bit color depth image, but 4-bit and 8-bit images adjust their color depth depending on which computer they play on. This has two implications:

- A 4-bit cast member expands to twice its original file size in memory on a 256-color monitor.

- An 8-bit cast member shrinks to half its original file size in memory on a 16-color monitor.

The following formula measures how much memory a bitmap uses:

Number of bytes used by a bitmap=(image's height in pixels x image's width in pixels x color depth in bits)/8

For example, a bitmap image that is 100x100 pixels and has a 4-bit color depth uses 5,000 bytes of memory: $(100 \times 100 \times 4) / 8 = 5,000$.

● **Creating projectors**

To create projectors for any version of Windows, you must use the Windows version of Director; likewise you can only create Macintosh projectors with the Macintosh version of Director.

Note: You can only include Director 5 movies in projectors. Use **Update Movies** to convert older movies to the latest version of Director.

To create a projector:

1. Choose Create Projector from the File menu.
The Create Projector dialog box appears.
2. Double-click the movies and external casts you want to include in the projector.
Director transfers the name of the movie or cast to the file list.
If you want to add more movies to the projector, repeat the step for each one. If you want to include all the movies that are in the folder that's open, click the Add All button.
3. Use the Move Up and Move Down buttons to arrange the movies in the proper order.
Movies play in the order they appear in the list. Be sure to put the starting movie at the top of the list. If your movie contains Lingo that switches between movies, the order of the other movies may not be important.
4. To change any of the projector options, click Options.
The most important option is selecting the type of computer the projector will run on.

Windows projector option

Windows NT and 95
Windows 3.1

Runs on

Windows NT and 95
Windows 3.1--also runs slowly on Windows 95 and NT

Macintosh projector option

Power Macintosh Native
Standard Macintosh
All Macintosh

Runs on

Power Macintosh only
Macintosh older than Power Macintosh--also runs slowly on Power Macintosh
All Macintosh systems at optimal speed, but the projector file is much larger

5. Click Create.
A directory dialog box appears.
6. Name the projector, and then click OK.
Director turns the movies you've selected into a projector.

When you play a projector, it starts the play-only version of Director, plays the movie or movies that are part of the projector, and then quits automatically.

● **Create Projector command**

Preparing a movie for distribution

- **Preparing a movie for distribution**

To distribute a movie to users who don't own Director, you need to create a projector.

A projector is a play-only version of a movie or series of movies. It appears as an application program in your system software. When a projector runs, it automatically begins playing the first movie and quits when the last movie finishes.

You cannot edit a projector in Director. You must edit the movie source file and then create a new projector.

When a projector plays, it accesses all external linked files the same way an ordinary movie does. All linked media--bitmaps, sounds, digital videos, and so on--must be in the same location relative to the projector as they were when the movie was created. To be sure there is no problem when you distribute the movie, place linked files in the same folder as the projector, or in a folder inside the projector folder.

- **Organizing movies in a larger production**

In most cases, you should divide a larger production into a series of smaller movies. You can combine as many movies as you want in a projector, but larger files take longer to save and are cumbersome to work with. Also, movies are easier to change if they are organized in discrete sections.

The best way to organize a larger production is to create a small projector file that launches the movie and then branches to other movies. This saves you the trouble of creating a new projector every time you change one part of a movie.

- **Create Projector command**
 - Creating Projectors**
 - Update Movies**

● **Protecting and compacting with Update Movies**

During a project, use Update Movies to compact groups of files you have been working on that may have become fragmented. (This is the same as using Save and Compact on a file.)

At the end of a project, use it to compact and protect all your movies and casts at once.

A protected movie can be played only from a projector or as movie in a window. To play a protected movie from a projector, you must use Lingo to go to or play the protected movie. Protected casts can only be opened by protected movies.

To protect and compact movies and casts:

1. Choose Update Movies from the Xtras menu.

The Update Movies dialog box appears.

2. Choose options for Action and Original Files.

Action defines whether the selected files are updated or protected. Original files specified if original files are deleted or moved to a new folder you specify.

4. Click OK.

A dialog box appears with which you select files to change.

5. Select movies and casts you want to update and click Add.

Click Add All to add all the movies in the current folder. The movies you select appear in the file list at the bottom of the dialog box. You can update movies in different folders at the same time.

Choose Add All Includes Folders before you click the Add All button to make it include any movies or casts inside folders appearing in the upper list. This option is useful for updating large projects with several levels of folders.

6. Click Update.

Director saves new versions of the selected movies with the same names and locations as the original movies. This ensures that all links and references to other files continue to work properly. Director copies the original movies to the folder you specified, recreating their original folder structure. If you didn't specify a folder for the original movies, Director prompts you to select one.

Note: Director adds a .DXR suffix to protected movies, and .CXT to protected external casts.

● **Update Movies command** **Converting Director 4 movies**

● **Converting Director 4 movies**

You must use Update Movies to convert any Director 4 movie with a shared cast (Shared.dir) to Director 5. To update movies from older versions (pre-4.0), you must first convert them to the Director 4 file format.

Using Update Movies makes the following changes to movie files from Director 4:

- Converts the movie into a Director 5.0 format file.
- Converts a shared cast (Shared.dir) to a linked external cast named Shared.cst. It renumbers the cast members so they begin with number one and updates all score references to the new numbers. Make sure you select Shared.dir as one of the files to be updated while using Update Movies.
- Places transitions in the cast.

Lingo from pre-4.0 versions that was allowed in Director 4 may not work in Director 5. If alert boxes inform you of script errors during Update Movies, this may be the problem. Update Movies converts movies with old Lingo to the new file format, but they will probably not run. To use these movies in Director 5, you have to find and change the old lingo. For information on outdated Lingo, see "Using outdated Lingo" in the introduction to Learning Lingo.

Improved System-Win palette

Director 5 uses an improved System-Win palette. For compatibility, Director still includes the System-Win palette provided in Director 4, which has been renamed "System-Win (Dir4)." You may notice that the Director user interface is a little brighter when it's the active palette.

While your Director 4 movies will play with exactly the colors and artwork as in Director 4, it's easy to convert them to the new System-Win palette. To do so:

1. Make a copy of your movie.
2. Choose Movie Properties from the Modify menu and set the movie palette to System-Win (Dir4).
3. Use Find Cast Member on the Edit menu to find all cast members in your movie that use the System-Win (Dir4) palette.
4. Click Select All.
5. Choose Transform Bitmap from the Modify menu and remap those cast members to System-Win.
6. Check the score for any references to the old palette.
7. Save your movie.

The colors in System-Win that were replaced are more subdued versions of the same colors, so you should find the results to be generally pleasing.

- **Update Movies command**
Protecting and compacting with Update movies

- **Understanding font mapping**

Director stores font, size, and style information for each text cast member. However, when you open a movie created on a Macintosh using the Windows version of Director, Windows may create visually different text than the text displayed on a Macintosh.

Most Windows PCs don't use the same fonts used in your Macintosh movie. If Windows doesn't have one of these fonts, Director substitutes another available font. Macintosh and Windows font sizes can differ. For example, 12-point text on the Macintosh can appear smaller on the PC.

When you create a new movie, Director looks for a file called FONTMAP.TXT in the same folder as the Director application. This file specifies how Director maps fonts between the Macintosh and Windows platforms. If Director finds this file, it uses it to create an internal font map for the movie. If no FONTMAP.TXT file exists, the new movie uses no font map. When you open the movie on the PC, Director uses the movie's internal font map to determine the appropriate substitute Windows fonts for text cast members that were created on a Macintosh. If the movie has no font map, Director substitutes other available fonts. The sample FONTMAP.TXT file at the end of this section provides an example of how Director maps Macintosh fonts to Windows fonts.

-
- If you want to guarantee that field text looks identical on both platforms, convert your field cast members to bitmaps before opening the movie on the other platform. Be aware, though, that bitmapped text uses more disk space and cannot be edited as text.
- Use text cast members if you want identical looking text. Text cast members are not dependent on playback-system fonts, they animate faster than fields, and they use only slightly more memory (but they cannot be edited at playback time).

Buttons, which appear as field cast members, should not be converted to bitmaps. Bitmapping buttons removes certain properties or attributes assigned to button cast members and could affect Lingo scripts that refer to these button properties.

- **Editing a movie's FONTMAP.TXT file**
Using fonts on high-resolution screens

● Editing a movie's FONTMAP.TXT file

When you open a new movie, Director looks for a font map file named FONTMAP.TXT in the same folder as the Director application. This file specifies the font mapping for all new movies. If Director can't find this file, it does not use a font map for new movies.

You can edit an existing movie's font map file to specify which fonts Director substitutes when you open the movie on the PC. Rather than creating this file from scratch, you can save the movie's internal font map table in a text file, and then edit this file as necessary.

To define the font mapping information for a movie, it's more convenient to edit the FONTMAP.TXT file before you begin authoring a movie, since Director automatically uses the information stored in the FONTMAP.TXT file when you open a new movie. (If you've already created the movie, you can still edit the font map file, but you will then have to manually load the file into the movie to have Director apply it to the movie.)

Note: If you edit a text, field, or button cast member on the PC in a movie created on a Macintosh, the cast member loses its original Macintosh font information. Similarly, if you edit the text, field, or button cast member on the Macintosh for a movie created on the PC, the cast member loses its original Windows font information. If you plan to edit a movie on both the Macintosh and Windows platforms, make sure that the font mapping file specifies that each Macintosh font has only one substitute font on the PC, and vice versa. This one-to-one font mapping ensures that Director will be able to assign the appropriate substitute font when you edit a text cast member on one platform and then open the movie on the other platform.

A movie's FONTMAP.TXT file might look like this:

```
; This is a sample FONTMAP.TXT file
; Comments are denoted by using ";" or "--" to start the line
; The format for Font Mapping is:
; Platform:FontName => Platform:FontName [MAP (NONE | ALL)] [OLDSIZE =>
  NEWSIZE]
-- The format for specific Character Mapping is
-- Platform: => Platform:  OLDCHAR => NEWCHAR ...
; Here are sample mappings for the standard Mac fonts:
Mac:Chicago      => Win:"MS Sans Serif"
Mac:Courier       => Win:"Courier New"
Mac:Geneva        => Win:System Map All
Mac:Helvetica     => Win:Arial
Mac:Monaco        => Win:Terminal
Mac:"New York"   => Win:"MS Serif" Map None
Mac:Symbol        => Win:Symbol
Mac:Times         => Win:"Times New Roman" 14=>12 18=>14 24=>18 30=>24
; Here are sample mappings for the stock Windows fonts
Win:Arial         => Mac:Helvetica Map All
Win:"Courier"     => Mac:Courier
Win:"Courier New" => Mac:Courier
Win:"MS Serif"    => Mac:"New York" Map None
Win:"MS Sans Serif" => Mac:Chicago
Win:Symbol        => Mac:Symbol
Win:System        => Mac:Geneva
Win:Terminal      => Mac:Monaco
Win:"Times New Roman" => Mac:"Times" 12=>14 14=>18 18=>24 24=>30
```

Note: From Windows to Mac, Courier and Courier New map onto Courier. When coming back to Windows only Courier New will be used.

Here is a sample character mapping for the bullet char

Mac: => Win: 165=>149

Win: => Win: 149=>165

Note:

- Comment lines must begin with two dashes (--) or a semicolon (;)
- Only one font mapping definition can be specified on a line
- Arguments must be separated by spaces or tabs
- If a font name consists of more than one word, it must be enclosed in quotation marks

- **Defining the font mapping for a new movie**
Changing the font mapping for an existing movie

● Defining the font mapping for a new movie

To define the font mapping for a new movie:

1. Using any application that can edit text, open the sample FONTMAP.TXT file that's in the same folder as the Director application.

When you installed Director, this file was placed in the same folder as the Director application. If the file is missing, you can either re-install it or create it from scratch. See the end of this section for an example of a FONTMAP.TXT file.

2. For each Macintosh font remapping entry, type on one line:

```
Mac:MacFontName=>Win:WinFontName [MAP (NONE|ALL)]  
[MACfontsize=>WINfontsize]
```

where MacFontName is the name of the Macintosh font, and WinFontName is the name of the Windows font being substituted for the Macintosh font.

The two arguments enclosed in brackets are optional. MAP ALL or MAP NONE specifies whether you want to remap characters with ASCII values greater than 127 or just pass them through. The default is MAP ALL.

You can specify how you want the characters to be remapped, as described in Step 3. The sample FONTMAP.TXT file contains mappings for a few commonly used graphical characters.

The last argument, [MACfontsize=>WINfontsize], consists of one or more pairs of numbers, separated by a space, that let you map a Macintosh font size to a Windows font size.

Because font sizes appear smaller on a PC, you might want to map Macintosh font sizes to larger Windows font sizes.

3. For each Macintosh special character that you want to remap, type:

```
Mac:=>Win: OLDCHAR=>NEWCHAR OLDCHAR=>NEWCHAR
```

where OLDCHAR is the ASCII value of the Macintosh special character, and NEWCHAR is the ASCII value of the Windows character being substituted for it. You can enter as many remapping pairs as you want by separating each one with a space.

You can only remap characters whose ASCII values are greater than 127 and less than 255.

If you didn't specify MAP ALL for any of the font remapping entries, as described in Step 2, you can skip this step.

4. Save the file as ASCII text, in the same folder as the Director application.
5. Open a new movie in Director.

When you open a new movie, Director looks for the font map file named FONTMAP.TXT in the same folder as the Director application. All new movies will use the font mapping information in the FONTMAP.TXT file. You can edit this file on a movie-by-movie basis, as necessary.

Existing movies continue to use the font map information (if any) stored within the movie rather than the font mapping specified in the FONTMAP.TXT file.

● **Changing the font mapping for an existing movie**

To change the font mapping for an existing movie:

1. Using any text editing application, edit the FONTMAP.TXT file as described in "Editing a movie's FONTMAP.TXT file," earlier in this section.
Save this file using any name of your choice.
2. Open the movie whose font mapping you want to change.
3. Choose Movie Properties on the Modify menu.
4. Click Load from File.
This option lets Director load the font mapping assignments specified in the font map file.
5. In the dialog box, select the font map file you just edited and click Open.
6. Click OK in the Movie Info dialog box.
7. Save the movie and close it.
8. Open the movie again.

The movie now uses the font map information specified in the font mapping file.

- **Using fonts on high-resolution screens**

Windows Setup program offers a Large font option, which uses a larger font on higher-resolution screens. When using the Large font option, text in TrueType or System font on an 800x600 pixel or larger display can wrap differently than it does on the Macintosh.

To avoid problems with the Large font option when playing back movies on screens larger than 800x600 pixels, use Windows' Small font option whenever possible. When it is necessary to use the Large font option, first test the movie on a large screen to verify that text looks appropriate.

• Using XCMDs and XFCNs

Related topics:

[Differences between Xobjects and XCMDs](#)

[Differences between XObjects and XCMDs](#)

[Learning to use XCMDs](#)

[Using an XCMD or XFCN](#)

[XCMDs and callbacks](#)

[XCMD and XFCN callback requests](#)

Lingo lets you use HyperCard's XCMDs and XFCNs in your movies. Using XCMDGlue--part of Director's *Standard.xlib library of XObjects-- you can access XCMDs and XFCNs from Lingo scripts. This lets you can extend Director's capabilities by using the many XCMDs and XFCNs available from HyperCard.

Most XCMDs and XFCNs work automatically with XCMDGlue, but some may not. When the XCMD's primary purpose is to perform a HyperCard-specific action--such as handling cards, HyperTalk scripts, or other parts of the HyperCard interface--the XCMD or XFCN might generate an error message when used in Director.

XCMDs and XFCNs are closely related. For convenience, Director Help refers to them collectively as XCMDs.

Note: XCMDs provide an interface to external code modules but are not capable of ensuring that the external code modules themselves perform as intended. You must make sure that the external code modules perform correctly to have them produce the desired results in Director.

- **Differences between XObjects and XCMDs**

XCMDGlue works differently from XObjects. You don't create instances of XCMDGlue to work with specific XCMDs. Instead, XCMDGlue acts as an interpreter between Lingo and the XCMD.

A major difference between XCMDs and XObjects is that an XObject can have multiple instances:

- One XObject can be used to create a number of independent objects, each capable of performing different operations.
- An XCMD cannot create new instances, so it can perform only one function at a time.

For these cases, you can use Lingo to create a special mechanism which may solve the problem. For further information, see the **XCMDs and callbacks** topic.

● Learning to use XCMDs

Related topics:

[Opening XCMD resources](#)

[Viewing XCMD resources](#)

[Closing XCMD resources](#)

Like using XObjects, using an XCMD involves three basic steps:

1. Opening the XCMD
2. Exchanging messages with the XCMD to perform some function
3. Closing the XCMD.

One of the best ways to learn about XCMDs is to use them in Director's message window. In this section, you'll see how to open, view the contents of an XCMD resource by exchanging a message with the XCMD, and close an XCMD.

● Opening XCMD resources

XCMDs can be located in two places: in an external file or in a Director movie.

When an XCMD resource is stored in the current movie's resource fork, the XCMD is automatically opened when the movie is opened. This is similar to the way *Standard.xlib is automatically opened when you launch Director. You can copy XCMD resources into your Director movie using a resource editor like ResEdit.

When an XCMD resource is stored in an external file such as a resource file or stack, you can open it with the openXlib command. If the file is in another folder, you must specify a full pathname to the folder. The easiest way to access the file is to place it in the same folder as your Director movie or the Director application.

To open an XCMD using the openXlib command:

1. Launch Director.
2. Open the message window and type openXlib followed by the name of the XCMD resource file.
3. Press Return.

The resource file you specified opens.

One resource file can contain multiple XCMDs. When you use the openXlib command, all XCMDs stored in the specified XCMD resource file are opened. The XCMD resource file can be a HyperCard stack, a resource file, or even a TeachText document containing XCMD resources. Notice that this is the same command used to open regular XObjects.

- **Viewing XCMD resources**

After you've opened the XCMD, you can use the showXlib command to display all open resource files that contain XCMDs as well as XObjects.

To display a list of all open resource files that contain XCMDs and Xobjects, type showXlib in the message window, and then press Return.

To display the contents of a specific XCMD resource file, type showXlib followed by the name of the resource file, and then press Return.

● **Closing XCMD resources**

The `closeXlib` command lets you close all open resource files that contain XCMDs and XObjects.

To close all open resource files that contain XCMDs and XObjects, type `closeXlib` in the message window, and then press Return.

To close a specific resource file that contains XCMDs, type `closeXlib` followed by the name of the resource file in the message window, and then press Return.

• Using an XCMD or XFCN

In many cases, once you open an XCMD, you can use the XCMD in your Lingo scripts the same way you would use it in a HyperTalk script. XCMDGlue does everything else by converting the XCMD for you. For example, the following handler would let you use the MIDIplay XCMD (from Opcode Systems) to play a MIDI file from Director:

```
on startMIDIplayback
  openXlib (the pathname & "MIDIplay")
  -- opens the MIDIplay XCMD
  -- Use Lingo's "pathname" function to find
  -- resource files
  -- in the same folder as your movie
  MIDIplay "open","MyDrive:MyFolder:myMIDIfile"
  -- opens the MIDI file to be played
  MIDIplay "start"
  -- starts playback of the MIDI file
end startMIDIplayback
```

This handler stops the playback of the MIDI file:

```
on stopMIDIplayback
  MIDIplay "stop"
  closeXlib (the pathname & "MIDIplay")
end stopMIDIplayback
```

● **XCMDs and callbacks**

Related topics:

Using a callback handler

Defining the callback factory

Creating the callback object

Specifying the callback handler

Not all XCMDs can be used with XCMDGlue in a completely transparent manner. Occasionally, XCMDGlue is unable to properly convert the XCMD. When you attempt to use an XCMD's syntax in a script, an error message is displayed.

Certain XCMDs may call on HyperCard to internally perform some tasks while the XCMD is executing. Most of these are conversion routines and are used to conveniently convert information to and from different formats. The remaining callbacks either involve the HyperTalk interpreter or access information stored in HyperCard-specific entities such as fields, or they do both. The table of HyperCard callback requests at the end of this appendix lists specific technical information regarding these callbacks.

Lingo automatically supports all callbacks that are not overly specific to HyperCard. Still, some HyperCard-specific callbacks are supported when Lingo provides a direct equivalent. The remaining callbacks that are not automatically supported (a total of nine) are so specific to HyperCard that they cannot be resolved automatically unless the application calling the XCMD is virtually identical to HyperCard. Even in such cases, it is still possible to use an XCMD by using a user-defined mechanism called a callback handler.

• Using a callback handler

A callback handler uses a Lingo factory to accept and respond to messages that correspond to HyperCard callback requests. A factory is a set of scripts that can be used to create an object. In Director 4.0, the functionality of factories has largely been replaced by parent scripts. For more information on parent scripts, see Chapter 10, "Parent Scripts and Child Objects." In this specific case, however, a factory provides the best way to respond to callbacks. This section shows you the steps necessary to create a callback factory, and to call that factory from a handler.

Essentially, a callback handler provides a mechanism that some XCMDs already expect to be available. The XCMD expects that when it sends or receives a callback message, something will be there to receive it and possibly return another message. (Usually HyperCard does this.) A callback handler defined in Lingo simply intercepts and returns these messages when appropriate. Whether you choose to use this information depends on your understanding of the purpose of the callback.

Fortunately, when XCMDGlue does not understand a callback request, it indicates the name of the callback in the error message. Once you know which callback your XCMD needs to deal with, you can create a callback handler for it.

There are three basic steps to creating a callback handler:

1. Defining a callback factory
2. Creating the callback object
3. Specifying the XCMD to be used with the callback object (with the `setCallBack` command that is part of XCMDGlue).

● Defining the callback factory

The first step in creating a callback factory is to define it. The following example factory includes methods for all the callbacks that are not supported by XCMDGlue. This factory does not attempt to do anything with the callback requests other than create a record of them in the message window. As you'll see later, you can use this information to process callbacks. This factory should be placed in a movie script:

```
factory callBackFactory
method mNew
me (mPut, 1, "SendCardMessage")
me (mPut, 2, "EvalExpr")
me (mPut, 3, "StringLength")
me (mPut, 4, "StringMatch")
me (mPut, 5, "SendHCMMessage")
me (mPut, 6, "ZeroBytes")
me (mPut, 7, "PasToZero")
me (mPut, 8, "ZeroToPas")
me (mPut, 9, "StrToLong")
me (mPut, 10, "StrToNum")
me (mPut, 11, "StrToBool")
me (mPut, 12, "StrToExt")
me (mPut, 13, "LongToStr")
me (mPut, 14, "NumToStr")
me (mPut, 15, "NumToHex")
me (mPut, 16, "BoolToStr")
me (mPut, 17, "ExtToStr")
me (mPut, 18, "GetGlobal")
me (mPut, 19, "SetGlobal")
me (mPut, 20, "GetFieldByName")
me (mPut, 21, "GetFieldByNum")
me (mPut, 22, "GetFieldByID")
me (mPut, 23, "SetFieldByName")
me (mPut, 24, "SetFieldByNum")
me (mPut, 25, "SetFieldByID")
me (mPut, 26, "StringEqual")
me (mPut, 27, "ReturnToPas")
me (mPut, 28, "ScanToReturn")
me (mPut, 31, "FormatScript")
me (mPut, 32, "ZeroTermHandle")
me (mPut, 33, "PrintTEHandle")
me (mPut, 34, "SendHCEvent")
me (mPut, 35, "HCWordBreakProc")
me (mPut, 36, "BeginXSound")
me (mPut, 37, "EndXSound")
me (mPut, 38, "RunHandler")
me (mPut, 39, "ScanToZero")
me (mPut, 40, "GetXResInfo")
me (mPut, 41, "GetFilePath")
me (mPut, 42, "FrontDocWindow")
me (mPut, 43, "PointToStr")
me (mPut, 44, "RectToStr")
me (mPut, 45, "StrToPoint")
me (mPut, 46, "StrToRect")
me (mPut, 47, "GetFieldTE")
me (mPut, 48, "SetFieldTE")
```

```

me (mPut, 49, "GetObjectName")
me (mPut, 50, "GetObjectScript")
me (mPut, 51, "SetObjectScript")
me (mPut, 52, "StackNameToNum")
me (mPut, 53, "Notify")
me (mPut, 54, "ShowHCAAlert")
me (mPut, 100, "NewXWindow/GetNewXWindow")
me (mPut, 101, "CloseXWindow")
me (mPut, 102, "SetXWIdleTime")
me (mPut, 103, "XWHasInterruptCode")
me (mPut, 104, "RegisterXWMenu")
me (mPut, 105, "BeginXWEdit/EndXWEdit")
me (mPut, 106, "SaveXWScript")
me (mPut, 107, "GetCheckPoints")
me (mPut, 108, "SetCheckPoints")
me (mPut, 109, "XWAllowReEntrancy")
me (mPut, 110, "SendWindowMessage")
me (mPut, 111, "HideHCPalettes")
me (mPut, 112, "ShowHCPalettes")
me (mPut, 113, "XWAlwaysMoveHigh")
me (mPut, 200, "GoScript")
me (mPut, 201, "StepScript")
me (mPut, 202, "AbortScript")
me (mPut, 203, "CountHandlers")
me (mPut, 204, "GetHandlerInfo")
me (mPut, 205, "GetVarInfo")
me (mPut, 206, "SetVarValue")
me (mPut, 207, "GetStackCrawl")
me (mPut, 208, "TraceScript")

method mEvalExpr x
  put "mEvalExpr:" && x

method mSendHCMessage x
  put "mSendHCMessage:" && x

method mSendCardMessage x
  put "mSendCardMessage:" && x

method mGetFieldByName card, name
  put "mGetFieldByName:" && card && name

method mGetFieldByNum card, num
  put "mGetFieldByNum:" && card && num

method mGetFieldByID card, id
  put "mGetFieldByID:" && card && id

method mSetFieldByName card, name, value
  put "mSetFieldByName:" && card && name && value

method mSetFieldByNum card, num, value
  put "mSetFieldByNum:" && card && num && value

method mSetFieldByID card, id, value
  put "mSetFieldByID:" && card && id && value

```



```
method mUnknown which
  put me(mGet, value(which)) into callBackName
  put "mUnknown:" && which && "(" & ¬
  callBackName & ")"
```

You do not need to specify every callback handled in this factory. You are required to define methods only for the callbacks that are indicated in error dialogs generated by the XCMD. For example, the mEvalExpr callback may be the only callback you need to account for.

As indicated in this example, the put statements in each method are optional. They are there to let you know what the XCMD or XFCN is attempting to tell HyperCard. You can use this information in any way you want. Sometimes, a callback requires a value (message) to be sent back to HyperCard. If you know what that value should be, use return at the end of the specific callback method's script. For example, if a callback required HyperCard to return TRUE or FALSE you could use a method similar to the following:

```
method callBackMethod
  if test then return TRUE else return FALSE
end callBackMethod
```

Some XCMDs use a large amount of processor time. In this situation, using a put statement in your script slows down whatever the XCMD does, because the put statement has to be evaluated and written into the message window. You can optimize the callback factory in this case by removing the put statements.

When a callback error occurs, the XCMD usually stops running after you click OK in the error dialog box. However, because of the design of certain XCMDs, the XCMD sometimes continues to execute. You still need to create a callback handler for these XCMDs. Otherwise, unexpected results could occur.

- **Creating the callback object**

After you have defined a callback factory, you can create a factory object using the following statement:

```
put callbackFactory(mNew) into callbackObject
```

• **Specifying the callback handler**

Finally, you specify the callback handler with the following statement:

```
setCallBack XCMD/XFCNname, callbackObject
```

The `setCallBack` command is part of the `XCMDGlue` `XObject`.

The `XCMD` or `XFCN` should now function properly. If you later use other elements of the `XCMD`'s syntax, you might still need to deal with other callbacks. You can accomplish this easily by adding the appropriate method to your callback factory.

● XCMD and XFCN callback requests

The following are HyperCard's callback requests. The symbol in the rightmost column identifies which level of support is provided for each callback.

Number	HyperCard callback	Type*
1	SendCardMessage	-
2	EvalExpr	-
3	StringLength	4
4	StringMatch	4
5	SendHCMesssage	-
6	ZeroBytes	4
7	PasToZero	4
8	ZeroToPas	4
9	StrToLong	4
10	StrToNum	4
11	StrToBool	4
12	StrToExt	4
13	LongToStr	4
14	NumToStr	4
15	NumToHex	4
16	BoolToStr	4
17	ExtToStr	4
18	GetGlobal	4
19	SetGlobal	4
20	GetFieldByName	-
21	GetFieldByNum	-
22	GetFieldByID	-
23	SetFieldByName	-
24	SetFieldByNum	-
25	SetFieldByID	-
26	StringEqual	4
27	ReturnToPas	4
28	ScanToReturn	4
31	FormatScript	-
32	ZeroTermHandle	-
33	PrintTEHandle	-
34	SendHCEvent	-
35	HCWordBreakProc	-
36	BeginXSound	-
37	EndXSound	-
38	RunHandler	-
39	ScanToZero	4
40	GetXResInfo	-
41	GetFilePath	-
42	FrontDocWindow	-
43	PointToStr	-
44	RectToStr	-
45	StrToPoint	-
46	StrToRect	-
47	GetFieldTE	-
48	SetFieldTE	-
49	GetObjectName	-

50	GetObjectScript	-
51	SetObjectScript	-
52	StackNameToNum	-
53	Notify	-
54	ShowHCAAlert	-
100	NewXWindow/GetNewXWindow	-
101	CloseXWindow	-
102	SetXWIdleTime	-
103	XWHasInterruptCode	-
104	RegisterXWMenu	-
105	BeginXWEdit/EndXWEdit	-
106	SaveXWScript	-
107	GetCheckPoints	-
108	SetCheckPoints	-
109	XWAllowReEntrancy	-
110	SendWindowMessage	-
111	HideHCPalettes	-
112	ShowHCPalettes	-
113	XWAlwaysMoveHigh	-
200	GoScript	-
201	StepScript	-
202	AbortScript	-
203	CountHandlers	-
204	GetHandlerInfo	-
205	GetVarInfo	-
206	SetVarValue	-
207	GetStackCrawl	-
208	TraceScript	-

* 4: Automatically supported by Lingo

-: Requires a callback handler. Some messages and expressions (such as EvalExpr) may be evaluated by XCMDGlue in a manner compatible with HyperTalk. Other messages and expressions (such as GetFieldByName) always assume HyperCard entities for which there are no counterpart in Director.

Click a category to see a list of frequently asked questions:

[Technical support](#)

[Multiple platforms](#)

[General](#)

[Macintosh](#)

[Microsoft Windows](#)

[Digital video](#)

[Lingo](#)

[Made with Macromedia](#)



Click a category to see a list of frequently asked questions:

[Technical support](#)

[Multiple platforms](#)

[General](#)

[Macintosh](#)

[Microsoft Windows](#)

[Digital video](#)

[Lingo](#)

[Made with Macromedia](#)



FAQs -- multiple platforms



How do I take Director 5 movies from the Macintosh to Windows? Do I need to buy Director 5 for Windows?



What about Director and Windows 95? Netscape? OS/2? Blackbird?



FAQs -- general

- My projector does not center on the stage, even though I have told it to do so in Movie Info. Why?
- When I import graphics into Director and apply Background Transparent ink effect to them, the edges are not clean.
- Why are my buttons being ignored when I have a Wait for Digital Video Movie to Finish in Channel or other wait settings in the tempo channel?
- How can I expand Director's printing capabilities?
- Why do I get an out of memory error when importing a FLC or FLI on Windows, or a PICS file on Macintosh?
- How does Director work with a database?
- How can I play a Director movie on a hard disk and keep its content on a CD-ROM?



FAQs -- Macintosh

- I have Director for Macintosh, and it is quitting with a Type 1 Error.
- I am using Director for Macintosh, and all I want to do is make the area behind the stage black.

FAQs -- Microsoft Windows

- When I use Director for Windows, I get a General Protection Fault.
- Why do my 24-bit images look great in the cast window but look dithered when I put them on the stage? I am using Director for Windows.
- How do I play MPEG movies in Director for Windows?
- How can I send MCI commands from Lingo? Where is a list of MCI commands?
- When I run my Director for Windows projector, I get an error message that says "Lingo.ini not found."
- When I run my Director for Windows projector, it says "Handler not Defined: #FileIO."
- In Director for Windows, when I play a QuickTime movie or an AVI file with sound, I cannot play another sound file simultaneously. Why?
- Is there a way to embed a custom icon for a Director for Windows projector?
- Which installer do you recommend for Director for Windows projectors? Do you recommend a compiler for DLLs?
- How do I find the letter of a CD drive in Director for Windows?



FAQs -- digital video

- Where have the transitions and sounds gone when I export my Director movie to digital video?
- In Director for Windows, why does the controller of my QuickTime for Windows movie stay on the stage when I jump to a new frame? Or, on the Macintosh, why does the last frame of the digital video file stay on the screen when I jump to a new frame?
- How can I make digital video files play as well as they do outside of Director?

FAQs -- Lingo

- How do I find the movie script?
- How do I use a custom cursor in Director?
- What is a mask cast member, and how do I make one?
- I have a `rollOver` test in a frame that works properly, but when I jump to a new frame, that `rollOver` area is still being evaluated even though the sprite is no longer there. This also happens with the cursor of sprite property.
- Where is a list of `keyCodes`?

FAQs -- Made with Macromedia

- Basic information regarding distribution of "Made with Macromedia" titles.
- Who needs to comply with the Made with Macromedia logo requirements?
- What if the Macromedia runtime is an insubstantial part of a commercially distributed software product that was not Made with Macromedia?
- I am using Shockwave™ to add multimedia to my web site. Am I required to use the Made with Macromedia logo on my "Shocked" Web Site?
- I qualify for either full or partial marking of the Made with Macromedia logo. What are the steps I need to take to comply with this agreement?



Technical support

If you think you need technical support, either online, by fax, or by phone:

1. Please read everything relevant to the problem in the manuals and the online help.
2. Check the index for more references to the topic. More information on a procedure or feature may be found in a separate section.
3. If something used to work, think about what may have changed. Perhaps you installed new software or changed some settings.
4. Try creating a new file and reproducing the problem there. If the problem goes away in the new file, compare the new file with your old file to find and eliminate the differences.

Note: Most of the problems you encounter can be solved by following the five steps listed above.

If you still need help:

If you still need help at this point, a little advance preparation can save you time and money, and allow the support representative to help you faster. Consult the following checklist before contacting technical support:

- Please try to define the problem so that you can repeat the steps that led to the problem and specifically identify when and how the problem occurred. The support representative will need to know exactly what the problem is in order to provide help.
- Be able to provide the following information:
 - Product name, version number, and product registration number
 - Type of computer, such as 386, 486 or Pentium, local-bus or non-local bus, Quadra or PowerMac
 - Amount of memory installed
 - Amount of free hard disk space
 - Screen resolution (screen size in pixels, for example, 1024 by 768)
 - Screen color depth (number of colors or bits, for example, 256 colors or 8-bit color)
 - Graphics card manufacturer, model name, and driver version number.
 - Sound card manufacturer and model name
 - DOS and Windows or Macintosh System version numbers
 - A list of external devices connected to the computer
 - Brief description of the problem or error, and the specific text of any error messages

These steps will help us pinpoint and solve your problem more quickly.

Technical support:

Inside the U.S. and Canada

Outside the U.S. and Canada

Contacting Macromedia



Contacting Macromedia

Technical Support

Sales: Call 800-288-4797

Source & Center

Call 800-396-0129 or 415-252-7999.

Contact Source & Center for training, consulting services, purchasing Priority Access technical support, referrals for multimedia development, referrals to Macromedia Authorized Graphics/Imaging Centers (MAGIC) and to user groups, and authorization programs for trainers, developers and service bureaus.

Macromedia International User Conference

Call 415-252-7999

Success Stories

pr@macromedia.com

fax 415-626-1502

Product Suggestions and Feedback

director@macromedia.com

fax 415-626-0554.

Contact the Director Product Team with product suggestions and feedback about Director.

World Wide Web

<http://www.macromedia.com/>

Made with Macromedia program

Call 415-252-2000.

Macromedia offers Director 5 developers the ability to distribute applications created in Director without paying royalties. Our new Macromedia licensing policy allows you to distribute your Director projects royalty-free, provided you include the Made with Macromedia logo as described in our guidelines.



Technical support inside the United States and Canada

Debugging, designing, creating

Please note that technical support can answer installation, configuration, and general usage questions about the product. For help in debugging, designing, or creating your application, contact Macromedia Professional Consulting Services at 415-252-2245.

Online services

Information about Director is available in Macromedia's forums on CompuServe, America Online, Microsoft Network, and also at various sites on the Internet. For an up-to-date list of all of the resources available online, call MacroFacts, Macromedia's 24-hour fax information line and request document 3503. In the United States and Canada, call 800-449-3329. From elsewhere, call 415-863-4409.

- **CompuServe:** To reach the Macromedia forum on CompuServe, use the command Go Macromedia. On CompuServe, we provide message areas for discussion of multimedia development and support of our products, as well as libraries that contain useful utilities and examples--including drivers, models, DLLs, and XCMDs.
- **America Online:** In the United States, to reach the Macromedia forum on America Online, use the keyword Macromedia. On America Online, we provide message areas for discussion of multimedia development and support of our products, as well as libraries that contain useful utilities and examples--including drivers, models, DLLs, and XCMDs.
- **Microsoft Network:** To reach the Macromedia forum on Microsoft Network, use the command Goto Macromedia. On Microsoft Network, we provide message areas for discussion of multimedia development and support of our products, as well as libraries that contain useful utilities and examples--including drivers, models, DLLs, and XCMDs.

MacroFacts

Macromedia's 24-hour fax information line, providing instant access to Macromedia's products and services. In the United States and Canada, call 800-449-3329. From elsewhere, call 415-863-4409.

Contacting Technical Support

Technical Support fax: 415-703-0924

Technical Support phone: 415-252-9080

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



Technical support outside the United States and Canada

Online services

Information about Director is available in Macromedia's forums on CompuServe, Microsoft Network, and also at various sites on the Internet. For an up-to-date list of all of the resources available online, call MacroFacts, Macromedia's 24-hour fax information line and request document 3503. You can reach MacroFacts by calling 415-863-4409.

- **CompuServe:** To reach the Macromedia forum on CompuServe, use the command Go Macromedia. On CompuServe, we provide message areas for discussion of multimedia development and support of our products, as well as libraries that contain useful utilities and examples--including drivers, models, DLLs, and XCMDs.
- **Microsoft Network:** To reach the Macromedia forum on Microsoft Network, use the command Goto Macromedia. On Microsoft Network, we provide message areas for discussion of multimedia development and support of our products, as well as libraries that contain useful utilities and examples--including drivers, models, DLLs, and XCMDs.

MacroFacts

Macromedia's 24-hour fax information line, providing instant access to Macromedia's products and services. You can reach MacroFacts by calling 415-863-4409.

If you're looking for assistance by fax or phone internationally, please contact the vendor or the distributor from which you acquired Director.

For additional help, contact the Macromedia office in your area:

- **Macromedia Europe (including Europe, the Middle East, and Africa)**

4 Wellington Business Park
Dukes Ride, Crowthorne Berkshire
England
United Kingdom RG45 6LS

44.1344.76.1111
44.1344.76.1149 (fax)
44.1344.750.517 (fax)

- **Macromedia Japan**

Serom Building 3F
Shinsen-Cho 11-7
Shibuya-Ku
Tokyo
Japan 150

81.3.3462.5790
81.3.3462.5794 (fax)

- **Macromedia Asia Pacific**

9 Minto St
East Kew, Victoria
Australia 3102

61 3 9859 8325
61 3 9859 4162 (fax)

◆ **For Pacific and Latin America**

Macromedia
600 Townsend Street
San Francisco, CA 94103
U.S.A

415-252-2267
415-626-0554 (fax)



How do I take Director 5 movies from the Macintosh to Windows? Do I need to purchase Director 5 for Windows?

Yes, you need to have Director for Windows to make an Director executable (projector) for the Windows platform. Director movies play on both platforms, but the projectors themselves are platform-specific.

In general, you do not need to duplicate your work. Most of what you do in Director will be exactly the same on both platforms. You can open the same movie on the Macintosh and Windows machine. Even the Lingo is the same. Most of the other issues you will deal with will include your content: making sure that your cast members work on both platforms as well.

Our best advice would be to work with both platforms from the beginning of your project. You should be successful if you test early, test often, and test on all target machines.



What about Director and Windows 95? Netscape? OS/2? Blackbird?

Director for Windows 5 runs on Windows 95, Windows NT, and Windows 3.1.

There have been announcements made of the development of Director playback technologies for Netscape, OS/2 Warp, Blackbird, and others.

The best resource on the latest information of Director and Macromedia developments is the press release area of our CompuServe and America Online forums, and our web page: <http://www.macromedia.com>.



My projector will not center on the stage, even though I have told it to do so in the Movie Info dialog box. Why?

If you would like to avoid all of this, you can also use a Lingo command in your `on startMovie` handler to: "set the centerStage to TRUE".



When I bring my graphics into Director and apply Background Transparent ink effect to them, the edges are not clean.

Imported images may have very light shades around the edges that look white to the eye, but are slightly darker than true white. Background Transparent ink effect subtracts one color only. The default for this background color is white, the first indexed position of a palette. You can change the background color of a sprite by selecting the sprite on the stage and selecting a different color in background chip of the Tools window.

Some techniques used to work with these light colors are:

1. Use the eyedropper tool in the paint window to determine the position in the palette the light color. Select the image in the paint window with the selection rectangle tool in Shrink or No Shrink modes. Select the paintbrush and the switch ink effect to change that color to a true white.
2. Often images that are anti-aliased against a white background get a halo around them because the image has feathered edges with intermediate colors between the object color and the white background. One good solution is to recreate the image without anti-aliasing, or edit the edges by hand.



Why are my buttons being ignored when I have a Wait for Digital Video Movie to Finish in Channel X, or other wait settings in the tempo channel?

The tempo channel is intended for use in straightforward, linear presentations. It is quick and easy, and perfect for slide shows. If you begin to use Lingo to add interactive control of your movie, it is good practice to use Lingo alone.

Say, for example, you have written a sprite script for a button to "go to the frame + 1". If you also have a Wait for Digital Video Movie to Finish in Channel: X set in the tempo channel, Director will not know which message to execute. The tempo channel and Lingo compete for control of score navigation.

The good news is you can script everything in the tempo channel with Lingo and maintain interactivity at the same time.



How can I expand Director's printing capabilities?

From within the Director application, you can print for authoring purposes of previewing the stage, your scripts, cast members and more. For many reasons, one being to keep the size of the projector small, Director projectors have simple printing engines.

You can use the Lingo `printFrom` command to print the stage in a projector at 25, 50 or 100%. It is also possible to use third-party solutions to enhance the printing capabilities of a projector. Please take a look on our CompuServe and America Online forums or our web page for information and demonstration copies of those available. One of these most commonly used at this time is the PrintOMatic Xtra.



Why do I get an out of memory error when importing a FLC or FLI on Windows, or a PICS file on Macintosh?

When you import a FLC/FLI or PICS file into Director, a 27-frame file for example, it is similar to bringing in 27 full screen graphics at one time. If you have been working for a while, you could run out of memory if you try to import all of these graphics at once. This is a very common error when importing large 3D files.

What you will need to do is this:

1. Import one small section of the PICS (Mac) or FLC/FLI (Windows) file at a time.
2. Save the file by choosing Save and Compact (this will re-write the file and your changes).
3. Go into About Director, and choose to purge the memory.
4. Repeat this sequence again, or a few more times, depending on the size of the file.

Once the entire animation is in Director, you can put it back together on the stage with "Cast To Time." You may want to make sure in the paint window that the registration points of all of the images are the same. Double-click the registration tool to make sure it is in the center of each image.

To find more 3D animation discussions and techniques, take a look at the technical support KnowledgeBase file, available on our CompuServe and America Online forums, and web page.



How does Director work with a database?

You have a couple of options when working with databases and Director. You can either create your own database structure by using Lingo's lists. This is an easy and fast, cross-platform solution.

If you would like to communicate with an external database, you can use one of many Xtras available from third parties to do so. Some issues you may need to consider if using this approach include: cross-platform compatibility issues for the database, how to distribute the database, and the overhead of running multiple applications on the target machine. Information on these third party utilities can be found on our CompuServe and America Online forums and web page: <http://www.macromedia.com>.



I have Director for Macintosh, and it is quitting with a Type 1 Error.

Type 1 Error is a bus error, one of the most common errors on the Macintosh. It could mean a number of things, in order of severity. If you receive a Type 1 Error while running Director, you should run through this list of steps:

1. Allocate more memory to Director:
 - Quit Director.
 - Select the Director application icon in the Director folder (do not launch it).
 - Select Get Info from the File Menu.
 - Allocate more memory to the Preferred size for Director.
 - Try the task you were doing before.
2. If that does not work, disable all of your extensions:
 - Restart your computer and hold down the Shift key.
 - Keep the Shift key down until a message comes up saying "Extensions Disabled."

If the error goes away, you should work through your extensions to see which one(s) might be conflicting. We do not keep a list of extensions known to be incompatible chiefly because of the difficulty in creating such a list. It would be more descriptive to make a list of extensions that, under some circumstances, can lead to problems in Director. Sometimes two extensions that cause no problems by themselves will cause Macintosh applications to go haywire if they are both active at the same time. Similarly, extension conflicts can be the result of load order.

Director does not require any extensions, except QuickTime if you are using QuickTime movies. If you find that there is an extension conflict on your machine, you can use an Extension Manager (one ships with System 7.5) to disable all of your extensions except QuickTime while working in Director.

3. If that does not work, reinstall Director.
4. If that does not work, reinstall the System Software.
A full list of all the Macintosh System errors is available from Apple.



I am using Director for Macintosh, and all I want to do is make the area behind the stage black.

To do this in a projector, simply choose Full Screen in the Projector Options dialog box.



When I use Director for Windows, I get a General Protection Fault.

If you get a General Protection Fault when starting up Director in module DIR5WIN.DLL, then you might have the HP Deskjet printer driver set up as default in the Printers control panel. Selecting a different printer driver or making "none" the default should fix the problem.

If the General Protection Fault occurs in this module at another time during development, or in a different module, it is most likely a display driver issue. Switching to the Microsoft "VGA" display drivers will test if this is the problem.



Why do my 24-bit images look great in the cast window but look dithered when I put them on the stage? I am using Director for Windows 3.1.

Director for Windows 3.1 displays bitmap sprites on stage in 8-bit color depth (256 colors) or less. We do so to abide by a Microsoft standard set in Windows 3.1 to ensure the highest level of compatibility with a broader range of video display drivers. Director 5 run on Windows 95 and NT supports 24-bit color.

This limitation does not apply to digital video, like QuickTime for Windows and Video for Windows. Digital video goes through a different display buffer than the other sprites on the Director stage. One option you have is to convert your bitmaps to one-frame digital video files, as they are optimized to color depths greater than 8-bit. Because they play directly to the screen buffer, however, you will not be able to overlay sprites on top of them during playback.

The best technique, however, to maintain the score's 48 channel layering effects, would be dither your graphics to a custom 8-bit palette.



How do I play MPEG movies in Director for Windows?

Director uses the Lingo MCI command to pass strings to an MPEG playback controller such as the ReelMagic card. Initially, when Director for Windows shipped, you could only play the MPEG file and not control it. You can now get the "handle" to the MPEG window by using DLLGLue XObject, available on our CompuServe and America Online forums, and web page. Sample scripts come with the DLL Glue XObject, and a help file on MCI, called "MCISTRWH.HLP," is available in our CompuServe and America Online forums, and web page.



How can I send MCI commands from Lingo? Where is a list of MCI commands?

Director can send MCI commands from Lingo with the `mci "string"` command. The command you insert in the "string" area is documented with the device you are trying to control. See the [mci](#) help topic for information on using the Lingo `mci` command. Microsoft has a series of help files and technical documents on CompuServe and in their multimedia developer's kit. An MCI help file, called MCISTRWH.HLP, is located in our libraries on our CompuServe and America Online forums, as well as our web page: <http://www.macromedia.com>.



When I run my Director for Windows projector, I get an error message that says "Lingo.ini not found."

The two most frequent causes of this problem include:

1. A particular Cirrus Logic video driver that overwrites other areas of memory. Solution: Get newer Cirrus Logic video drivers from Cirrus Logic or use the generic Microsoft SVGA (640x480x256) driver.
2. A multimedia shell is in use on a low-end machine, thus, reducing system resources and available RAM below a workable level. The solution is to run straight Program Manager instead.



When I run my Director for Windows projector, it says "Handler not Defined: #FileIO."

Upon launching, the Director for Windows projector looks for a text file called "LINGO.INI". This file can be useful for last minute instructions for your completed Director project.

The projector looks for it in its own directory and then scans the DOS path, and when it finds a LINGO.INI file, it follows its instructions. The default LINGO.INI (in the Director application directory) has many comments and one solitary command: "openXlib 'fileio'." Comment this out and put a copy of the file in the same directory as your projector, and it will never ask for FILEIO.DLL again.

If by chance a LINGO.INI file from a different application created with Director remains on that machine, the projector will follow the instructions it finds first. (There was a LINGO.INI file associated with the Director Player for Windows projector as well.) You might get other errors at that time. Your safest bet is to supply a text file in the same directory as your projector, and call it "LINGO.INI." Even if it is blank, the projector will read it first and continue without any errors.

If you are using the FileIO XObject with Director for Windows, you will need to supply the FILEIO.DLL in the same directory as your projector. FileIO is not embedded into the Director Windows application as it is on the Macintosh. For more information on what to distribute with your Director for Windows projector, please refer to the "PROJECT.WRI" file in the Director for Windows application directory.



In Director for Windows, when I play a QuickTime movie or an AVI file with sound, I cannot play another sound file simultaneously. Why?

The fact is that Windows supports only one sound channel. Director for Windows can play more than one sound at a time (.WAV or .AIF) because it mixes them together. Macromedia created a technology called MACROMIX.DLL that allows Director to do this. In Director for Windows 4.0, it was necessary that the sounds needed to be of the same sampling rate (11.025, 22.05, 44.1 kHz) and sample sizes (8-bit and 16-bit) in order to mix them together. This is no longer true in Director for Windows 5.

What Director cannot do is mix sounds for QuickTime for Windows or AVI movies with audio files. Whichever sound gets to the sound port first, wins. The second sound will not be able to start playing until the first sound is completely finished with the sound channel.

The Director for Windows "README.WRI" file recommends to keep at least one frame between video and audio sources. Other techniques used to make sure the first sound is completely finished include:

```
for a puppetSound:  puppetSound 0
for a digital video:  set the movieRate of sprite X to 0
```



Is there a way to embed a custom icon for a Director for Windows projector?

The best (least risky) way to do this in Windows is to change the icon inside the PROJECTR.SKL file. It should be installed into the same directory as DIRECTOR.EXE.

1. Make a backup copy of PROJECTR.SKL.
2. Open PROJECTR.SKL as an .EXE in your favorite resource editor (One recommended is AppStudio).
3. Edit the icon. It's in the icon resource called APPICON.
4. Save the file. Make sure it is still called PROJECTR.SKL and is in the same directory.
5. Create a projector with your own icon.
6. Optional: restore the original version of PROJECTR.SKL.

AppStudio comes with Microsoft's Visual C++.

Note: Borland Resource Workshop versions 4.0 and 4.0.2 will NOT work for this procedure. (It comes with Borland C++.) The Director projector skeleton includes half a megabyte of code, but BRW will not recognize those code elements -- it only understands resources. Thus, you will run into "unexpected file format" errors trying to save your updated PROJECTR.SKL file as an EXE.



Which installer do you recommend for Director for Windows projectors? Do you recommend a compiler for DLLs?

There are many installer and compiler products on the market. We cannot, as a policy, officially recommend third-party software or hardware. It seems, from the feedback of many developers, that this decision is based on personal preference. Your best bet would be to ask the developers themselves on our on-line services, or contact a software reseller for this information.



Where have the transitions and sounds gone when I export my Director movie to digital video?

Many of the score-based transitions of Director will not export to QuickTime on the Macintosh, or AVI on Windows. You can experiment with increasing the chunk size and duration of some of them, but many will not convert with the file. When a frame-based Director animation is converted to a time-based QuickTime or AVI file, many of these transitions are lost along the way.

As for sounds, the export to AVI from Director for Windows does not support sound. Exporting sound with the Director animation to QuickTime on the Macintosh can vary in reliability. Success in doing this is affected by the sound format, available resources on the machine, where the sound is placed in the score and many other factors, none of which are constant.

The best technique is to export only the animations from Director. You should add the transitions, and sounds if you would like, in your favorite digital video editing tool. SoundEdit 16 is a nice complement when editing the sound of a QuickTime movie on a Macintosh. You can choose keyframes in the QuickTime movie, synchronize parts of the wave form to those frames, and save the QuickTime movie once again.



In Director for Windows, why does the controller of my QuickTime for Windows file stay on the stage when I jump to a new frame? Or, on the Macintosh, why does the last frame of the digital video file stay on the screen when I jump to a new frame?

In Windows, digital video plays directly to the screen buffer, on top of Director's compositing engine. You can turn this property off on the Macintosh, but not in Windows. It is a good technique on the Macintosh, however, to select Direct to Stage in the Cast Member Properties dialog box for the QuickTime movie. This way, the video can play on the top layer, above the other sprites.

When the video plays direct to stage, Director does not know that the movie is there, and therefore does not redraw the stage when jumping to a new frame. You will see an artifact of the video when you jump to a new frame if you do not remove it yourself.

There are many techniques you can use to redraw the stage yourself. Here is one example:

```
on exitFrame
  set the visible of sprite X to FALSE
  --X is the channel where the video is placed
  updatestage
end
```



How can I make digital video files play as well as they do outside of Director?

If you are having trouble with the performance of a digital video file in Director, you should:

1. Play the digital video file in Apple's MoviePlayer or the Windows Media Player. Make sure it runs to your satisfaction outside of Director.
2. After importing the file into Director, play it in the video window. You can access this window by double-clicking on the digital video file in the Cast window. If the file plays poorly from there, it will not play well in the score. If you are having troubles with the file, you should return to the MoviePlayer or Media Player to see if the file is intact.
3. If you are on the Macintosh, select Direct to Stage in the Digital Video Cast Member Info dialog box for the QuickTime movie. (Digital video is always Direct to Stage on Windows.)
4. Place the video in the score. In the script channel for that frame, type:

```
on exitFrame
  go to the frame
end
```

It is important to use this "go to the frame" loop to keep Director's playback head moving in that frame.

If you would like to loop in that frame until the digital video is done playing, you can use this script:

```
on exitFrame
  if the movieRate of sprite channelNumber
    then go to the frame
end
```

For a list of all of the digital video Lingo terms, see the [Lingo digital video](#) topic.



How do I find the movie script?

With the functionality of Director 5 to support more than one movie script, there is no one way to find "it," since there can be more than one of them. Once you create a movie script, you will find it in the cast window, but some ways to find it initially include:

- Press Ctrl-Shift-U.
- In the Window menu, select the script window.
- If you're in a score script, click the Add (+) button.

How do I use a custom cursor in Director?

In order to use a custom cursor in Director, there are only a few steps to follow. The Director *Learning Lingo* book covers this issue. For some reason, though, it still stumps many people, so here is a list of the rules to follow:

1. The cast member you use for your custom cursor must be 1-bit (black and white). You can verify this in the bottom left hand corner of the paint window.
2. Your Lingo syntax needs to be correct. You have two options of how to do this. Here are some sample scripts to illustrate this:

You can use the "cursor" command:

```
on startMovie
  cursor [5]
  --your cursor is in cast #5 and would be
  --active for the entire movie
end
```

You can use the "cursor of sprite" command:

```
on enterFrame
  if rollover (2) then
    --the sprite you rollover is in channel 2
    set the cursor of sprite 2 to [5]
    --your cursor is in cast #5
  end if
  --only when sprite 2 is rolled over
end

on exitFrame
  go to the frame
end
```



What is a mask cast member, and how do I make one?

When you use a custom cursor, the areas that are white in the black & white area will be transparent when rolling over other sprites. In order to make the white areas opaque, you will need to make a mask cast member. The Macintosh system watch cursor is a good example of this.

Here is one technique to do so:

1. Duplicate your custom cursor cast member in the cast window.
2. Double-click that cast member and bring it up in the paint window.
3. Select the pencil tool (after zooming in) and draw a one-pixel circumference around the bitmap, making it one pixel thicker than the other bitmap.
4. Take the paint bucket tool and fill the white areas with black.

The paint bucket tool may do the trick. In order to have the white area be opaque over other sprites, there needs to be opposition of black and white pixel when the two cursors are used together.

5. Again, make sure this cursor is set to 1-bit after editing.
6. Make sure this cast member is in the cast position following the custom cursor.
7. Add it to your syntax. For example:

```
set the cursor of sprite 2 to [5,6]
--where the cursor cast member is in cast #5
--and the mask cast member is in cast #6
```

●
I have a `rollOver` test in a frame that works properly, but when I jump to a new frame, that `rollOver` area is still being evaluated even though the sprite is no longer there. This also happens with the `cursor of sprite` property.

These tests for the mouse position, or "hot" areas, actually refer to sprite channel characteristics, and not to individual sprites. If the channel used to hold the sprite being rolled over alternates between empty and full, unexpected results may occur.

There are at least three ways of working with this:

1. One is to keep channels full in all frames. You can move all the frames next to one another and delineate them with markers.
2. Another approach would be to create a one-bit "dot" cast member using the tools window. (This would take up very little memory.) Put an instance of the "dot" in the empty frames of the sprite channel being used to test for the rollOver.
3. You can also scoot the sprite off-stage, up above the menu bar, to give it an off-screen position before removing it from the stage.

This issue is similar to setting the `cursor of sprite`, as mentioned in the Lingo Dictionary. The cursor property will stay in effect until you turn it off by setting the cursor to zero.

● Where is a list of keyCodes?

The **keyCode** function returns the numerical code for the last key pressed. (This keyboard code is the key's numerical value, not the ASCII value.)

There is not a list of keyCodes available in the Director documentation. It is easy to generate a keyCode yourself, though, by creating a one-frame test movie with these scripts:

In the movie script, type:

```
on startMovie
  set the keyDownScript to "put the keyCode"
end
```

In the frame script of frame 1, type:

```
on exitFrame
  go to the frame
end
```

When you play the movie, leave the message window open and the keyCode for any key you type will appear in the message window. Make sure that the message window is not the active window, or the keys you press will not be evaluated correctly.

To test for keys in the numeric keypad, you will need to test the keyCodes from a projector and put the keyCode into a text field. It is necessary to test for certain modifier keys specifically with Lingo (i.e. the `controlDown`, the `shiftDown`, etc.)

These keyCodes are standard on Macintosh keyboards, but might not be standard across IBM-compatible keyboards. We have not, however, come across a nonstandard keyboard here in technical support.

• How do I find the letter of a CD drive in Director for Windows?

Crossing drive volumes is more difficult in Windows than on the Mac. On the Mac, you can just say:

```
play movie "MyCD:Data:MovieA"
```

Any mounted volume with that name and path will work automatically. On Windows, though, the path may be "G:\MyCD\Data\MovieA" or "D:\MyCD\Data\MovieA" or whatever. Each end user's machine could have the hard drive on a different drive letter.

The following handler will help in finding the actual drive letter for the CD drive. It assumes that you have a uniquely named file that you pass to the handler (here called "weirdfil.txt") at the root level of your CD drive. It will successively search for the file by that name on the root of each drive letter (A to Z) in the alphabet, then return the letter name of the CD volume it finds the file on.

The script would look like:

```
put CheckDrive("weirdfil.txt") into myCD
```

The return would be the drive letter followed by a colon. This handler begins looking at drive C, since the chances of both A and B drive being floppy drives is very high.

```
on CheckDrive weirdfil
-- Note: use your actual filename instead of "weirdfil"
-- throughout this handler
repeat with I = 66 to 90
  set drive = numToChar( I )
  set myThisPath = string(drive & ":\"& weirdfil)
  set myFile = fileIO(mNew, "read", myThisPath)
  if objectP(myFile) then
    myFile(mDispose)
    return drive&":"
  exit
end if
end repeat
alert "Please check that"&&QUOTE&weirdfil&QUOTE&&-
"is on your CD drive."
end
```


● **How can I play a Director movie on a hard disk and keep its content on a CD-ROM?**

Here are some tips on how to play your Director movie from a hard disk while keeping its content on the CD-ROM:

1. Make sure linked cast members are all on one volume, and save the movie onto the same volume. This allows a relative pathname to be constructed at playback.
2. In the movie script, determine the drive letter of the CD-ROM, probably using a custom Xtra.
3. Construct a string starting with the drive letter of the CD-ROM, and the rest of the path duplicating the directory hierarchy where the movie was saved. This allows the relative paths constructed in the first step to remain usable.
4. setAt the searchPath, 1, <<the constructed string>>

This tells Director to search for files on the CD-ROM, beginning at the named directory.

If some of your content or subsidiary movies are on the hard disk, you will want to search the hard disk first, before looking at the CD-ROM. Most likely, these searches will be relative to the directory that the movie was launched from. This is how the searching will work, as set up above.

If you know that all your content and subsidiary movies are on the CD-ROM, you can dispense with searching the hard disk, saving at least one disk access:

5. set the searchCurrentFolder to FALSE

This will bypass searches relative to the current directory before scanning through the searchPath.

- **Basic information regarding distribution of "Made with Macromedia" titles**

Macromedia's royalty-free licensing policy means that you can distribute applications created with Director or Authorware to millions of end-users on multiple platforms--free. Simply include the Made with Macromedia logo on your product's packaging and credit screen, complete the Run time distribution agreement, and register your product with Macromedia to qualify.

This is meant to give a quick overview of the Made with Macromedia Run-time distribution agreement and answer frequently asked questions. It is not, however, a replacement for the actual agreement. Please refer to the Made with Macromedia (MwM) folder on your product CD for the Run-Time distribution agreement, logos, and logo usage guidelines.

- **Who needs to comply with the Made with Macromedia logo requirements?**

Any user of Authorware or Director who creates an End-user Product and distributes it outside of his own organization or anyone who causes an End-user Product to be created and distributed outside of his own organization. You must fill out Exhibit A of the Run time distribution agreement and place the Made with Macromedia logo on the outer most front, side, or back of the packaging and within the software on either a splash or credits screen.

- **What if the Macromedia runtime is an insubstantial part of a commercially distributed software product that was not Made with Macromedia?**

You do not need to place the Made with Macromedia logo on the outside of the packaging, but you will need to put the logo onscreen within the software that makes use of a Macromedia run-time. You must sign and return the Run-time distribution agreement, fill out Exhibit C, "Product Qualifying for Limited Markings," and place the Made with Macromedia logo on the splash or credits screen only. (You do not have to place it on the packaging.)

- **I am using Shockwave™ to add multimedia to my web site. Am I required to use the Made with Macromedia logo on my "Shocked" Web Site?**

No. This is a benefit to help inform your viewers that your site contains cutting edge multimedia content. Viewers will be directed to the Shockwave Plug-In so they can view the Macromedia-created movies within your web site. Macromedia will publish a list of shocked web sites on our web site, thus increasing the visibility of your site.

●
I qualify for either full or partial marking of the Made with Macromedia logo. What are the steps I need to take to comply with this agreement?

1. Complete, sign, and return one copy of the Run-time Distribution Agreement and either Exhibit A or C that are located in the Made with Macromedia folder of the Director or Authorware product CD. Exhibit A is for products that qualify for full marking and Exhibit C is for products that qualify for limited, software only, markings. The agreement becomes effective upon receipt by Macromedia. For multiple products or future products, you need only fill out and return an additional copy of Exhibit A or C. Each additional copy of Exhibit A or C will become effective upon receipt by Macromedia.
2. If you qualify for full markings, you need to place the Made with Macromedia logo on the outside of the packaging and on screen within the software. Logos are located in the Made with Macromedia (MwM) folder of your Authorware or Director product CD. See Exhibit B of the Run Time Distribution Agreement for detailed size and location guidelines. If you qualify for limited markings, you must place the Made with Macromedia logo on screen only within the software guidelines outlined in Exhibit B.
3. Incorporate the following copyright statement into the copyright screen of the end-user product.
 - (If Authorware was used to create the Publisher Product)
AUTHORWARE ® COPYRIGHT © 1993 Macromedia, Inc.
 - (If Director was used to create the Publisher Product)
DIRECTOR ® COPYRIGHT © 1994 Macromedia, Inc.
4. Send Macromedia two (2) copies of the final, packaged end-user software within 30 days of ship to:

Macromedia Developer Relations
600 Townsend Street
San Francisco, CA 94103

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Click a letter (above) to view Lingo elements alphabetically.

Or select one of the categories below to see a list of relevant Lingo elements:

Cast members

Casts

Code structures & syntax

Computer & monitor

Digital video

External files

Fields

Frames

Interface elements

Lingo that has changed in 5.0

Lingo that is outdated

Lists

Math & logical operators

Memory management

Movie control

Movie in a window

Navigation

New Lingo elements

Operators

Parent scripts

Puppets

Score generation

Sprites

Sound

Strings

Time

User interaction

Variables

For general information on using Lingo, see the **Lingo basics** topic.



Click a letter (above) to view Lingo elements alphabetically.

Or select one of the categories below to see a list of relevant Lingo elements:

[Cast members](#)

[Casts](#)

[Code structures & syntax](#)

[Computer & monitor](#)

[Digital video](#)

[External files](#)

[Fields](#)

[Frames](#)

[Interface elements](#)

[Lingo that has changed in 5.0](#)

[Lingo that is outdated](#)

[Lists](#)

[Math & logical operators](#)

[Memory management](#)

[Movie control](#)

[Movie in a window](#)

[Navigation](#)

[New Lingo elements](#)

[Operators](#)

[Parent scripts](#)

[Puppets](#)

[Score generation](#)

[Sprites](#)

[Sound](#)

[Strings](#)

[Time](#)

[User interaction](#)

[Variables](#)

For general information on using **Lingo**, see the **[Lingo Basics](#)** topic.

The Lingo menu appears when you click and hold the Lingo button in the script window. This menu displays the complete set of Lingo commands that you can use to create scripts for your movie.

Choosing an element from the Lingo menu enters it into a script at the insertion point. This saves you from typing the command and also eliminates typing.

Note: You can use the Copy command from the Windows Help Edit menu to copy Lingo examples from the Help topics; then you can paste the example into the Director script window and modify it for your use.

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

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

New Lingo elements

The following elements are new in Director 5.0 or have had functionality added since Director 4.0:

<u>activeWindow</u>	<u>name of CastLib</u>
<u>autoTab of member</u>	<u>new</u>
<u>beginRecording</u>	<u>number of CastLib</u>
<u>border of member</u>	<u>number of castLibs</u>
<u>boxDropShadow</u>	<u>number of members of castLib</u>
<u>boxType of member</u>	<u>on activateWindow</u>
<u>buttonType</u>	<u>on closeWindow</u>
<u>cancelIdleLoad</u>	<u>on moveWindow</u>
<u>case</u>	<u>on resizeWindow</u>
<u>the castLibNum of sprite</u>	<u>on rightMouseDown</u>
<u>center of member</u>	<u>on rightMouseUp</u>
<u>changeArea of member</u>	<u>on zoomWindow</u>
<u>channelCoun of member</u>	<u>openWindow</u>
<u>charPosToLoc</u>	<u>otherwise</u>
<u>chunkSize of member</u>	<u>pageHeight of member</u>
<u>clearFrame</u>	<u>paletteMapping</u>
<u>crop of member</u>	<u>pattern</u>
<u>deleteFrame</u>	<u>paletteRef</u>
<u>desktopRectList</u>	<u>the platform</u>
<u>digitalVideoTimeScale</u>	<u>preLoadMode of CastLib</u>
<u>digitalVideoType of member</u>	<u>preLoadMovie</u>
<u>dropShadow of member</u>	
<u>duplicate(list)</u>	<u>rect of member</u>
<u>duplicateFrame</u>	<u>rightMouseDown, the</u>
<u>duration of member</u>	<u>rightMouseUp, the</u>
<u>editable of member</u>	<u>sampleRate</u>
<u>emulateMultiButtonMouse</u>	<u>sampleSize</u>
<u>end case</u>	<u>save castLib</u>
<u>endRecording</u>	<u>score</u>
<u>fileName of castLib</u>	<u>scoreSelection</u>
<u>filled of member</u>	<u>scriptsEnabled</u>
<u>finishIdleLoad</u>	<u>scriptType</u>
<u>frameLabel</u>	<u>scrollByLine</u>
<u>framePalette</u>	<u>scrollByPage</u>
<u>frameScript</u>	<u>scrollTop of member</u>
<u>frameSound1</u>	<u>selection of castLib</u>
<u>frameSound2</u>	<u>setCallBack</u>
<u>frameTempo</u>	<u>shapeType</u>
<u>frameTransition</u>	<u>sound of member</u>
<u>frontWindow</u>	<u>wordWrap of member</u>
<u>height of member</u>	<u>timeScale of member</u>
<u>idleHandlerPeriod</u>	<u>trackCount(member)</u>
<u>idleLoadDone</u>	<u>trackCount(sprite)</u>
<u>idleLoadMode</u>	<u>trackEnabled</u>
<u>idleLoadPeriod</u>	<u>trackNextKeyTime</u>
<u>idleLoadTag</u>	

idleReadChunkSize
insertFrame
keyPressed
lineCount of member
linePosToLocV
lineSize of member
loc of sprite
locToCharPos
locVToLinePos
loop of member
margin of member
media of member
member
memberNum of sprite

trackNextSampleTime
trackPreviousKeyTime
trackPreviousSampleTime
trackStartTime(member)
trackStartTime(sprite)
trackStopTime(member)
trackStopTime(sprite)
trackText
trackType (member)
trackType (sprite)
transitionType of member
type of member
unloadMovie
updateFrame
updateLock
windowPresent



Lingo that is outdated in version 5.0

The following elements are obsolete and no longer supported:

- birth
- instance
- factory
- closeDA
- openDA
- when...then constructs

Lingo that has changed in version 5.0

The following terms have been revised to keep terminology clear now that Director has multiple casts. The older terms are still supported, but they will become obsolete and should be avoided:

Director 4.0 Term

backColor of cast
 cast
 castmembers
 castNum of sprite
 castType of cast
 center of cast
 controller of cast
 crop of cast
 depth of cast
 duplicate cast
 duration of cast
 erase cast
 fileName of cast
 foreColor of cast
 frameRate of cast
 height of cast
 hilite of cast
 loaded of cast
 loop of cast
 modified of cast
 move cast
 name of cast
 number of cast
 number of castmembers
 palette of cast
 picture of cast
 preLoad of cast
 preLoadCast
 purgePriority of cast
 scriptText of cast
 size of cast
 sound of cast
 text of cast
 textAlign of field
 textFont of field
 textHeight of field
 textSize of field
 textStyle of field
 video of cast
 width of cast

Director 5.0 Term

backColor of member
member
number of members
memberNum of sprite
type of member
center of member
controller of member
crop of member
depth of member
duplicate member
duration of member
erase member
fileName of member
foreColor of member
frameRate of member
height of member
hilite of member
loaded of member
loop of member
modified of member
move member
name of member
number of member
number of members
palette of member
picture of member
preLoad of member
preLoadMember
purgePriority of member
scriptText of member
size of member
sound of member
text of member
alignment of member
font of member
lineHeight of member
fontSize of member
fontStyle of member
video of member
width of member

- **Child-parent scripts**

A child object is a self-contained, independent copy of the handlers and variables in a parent script. Each group of child objects has its own behaviors and values for variables and properties:

- The handlers work the same as other handlers in Director.
- The property settings and variables can be set and tested the same way as the Lingo properties and variables you've already worked with.

A parent script contains three types of Lingo:

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

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

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

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

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

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

See "Parent and child scripts" in *Learning Lingo* and the ancestor and parent scripts in the movie MECH for more information about child objects.

- **actorList** and **ancestor** properties; **new** function; **me** keyword

● **Event message hierarchy**

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

This message:	Is sent to this series of scripts:
mouseDown, mouseUp	Primary event handler, sprite, cast member, frame, and then movie script
keyDown, keyUp	Primary event handler, sprite, cast member, frame, and then movie script
enterFrame, exitFrame	Frame and then movie script
idle	Primary event handler and then movie scripts
startMovie and stopMovie handler	Movie scripts
activateWindow, closeWindow, openWindow, resizeWindow, zoomWindow	Movie scripts
Custom handler calling statements	Movie scripts

For more information about strategies for placing handlers, see Chapter 1, "Script Basics," in *Learning Lingo*.

● **on activateWindow, on closeWindow, on enterFrame, on exitFrame, on idle, on keyDown, on keyUp, on mouseDown, on mouseUp, on openWindow, on resizeWindow, on startMovie, on stopMovie, on zoomWindow**

- **Puppeting**

By making a score channel a puppet, you can tell Director to ignore the score's settings for that channel and control the channel directly from Lingo. In a sense, the channel is a puppet and Lingo pulls the strings.

- For sprites and sounds, the channel remains a puppet until you use a statement to return control to the score. The statement you use depends on whether the channel is a sprite channel or a sound channel.
- For tempos and palettes, the puppet condition lasts until the playback head enters a frame that has a new palette or tempo setting.
- For transitions, the puppet condition applies only to the specific instance in which the puppet transition is used.

For details about using one of these commands, see that command's entry in the online help.

- **puppetPalette**, **puppetSound**, **puppetSprite**, **puppetTempo**, and **puppetTransition** commands

● Handling text

Director has two types of cast members dedicated to text:

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

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

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

For more information about handling text and fields, see Chapter 6, "Working with Text & User Input," in *Learning Lingo*.

- **Controlling sound**

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

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

- **sound playFile, puppetSound, sound fadeIn, and sound fadeOut commands; the soundBusy and soundEnabled properties**

• Using movie in a window

A movie in a window is a distinct Director movie that can be opened by the current movie. Because it's a Director movie, the movie in a window can keep its Lingo in effect.

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

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

The following lists the Lingo elements you use to achieve different tasks for playing a movie in a window. For details about an element, see its entry in the online help:

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

For an example of how these elements are used together to create a movie in a window, see Chapter 9, "Movies in a window," in *Learning Lingo*.

• **open window**, **close window**, **moveToFront**, and **moveToBack** commands; the **fileName of window**, **the titleVisible of window**, **the visible of window**, and **the windowType of window** properties

• Using variables

Director remembers and updates values by using variables. As the name implies, a variable contains a value that can be changed or updated as the movie plays. By changing the value of a variable as the movie plays, you can do things such as store information the user enters or record whether a specific event has happened.

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

Assigning values to variables

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

```
put "Mary" into theName
```

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

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

Creating variables

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

Global variables

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

You make a variable global by using the term `global` before the variable name in every handler that uses the global variable.

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

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

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

```
global gName  
put "Mary" into gName
```

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

```
on nameChange
```

```
global gName
put "John" into gName
end
```

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

To display all current global variables and their current values, use the `showGlobals` command in the message window.

Local variables

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

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

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

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

- **global** keyword, **clearGlobals**, **showGlobals**, **put**, and **set** commands

● **Types of scripts**

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

There are four types of scripts:

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

Sprite scripts can respond to mouseDown and mouseUp events and to keyDown and keyUp events if the sprite is a field.

Frame scripts can respond to mouseDown and mouseUp events, keyDown and keyUp events (if the sprite is a field), and enterFrame and exitFrame events.

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

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

- *Movie scripts* aren't assigned to a specific object but are available to the entire movie. Scripts assigned to the movie can respond to mouseDown and mouseUp events, keyDown and keyUp events (if the sprite is a field), enterFrame and exitFrame events, startMovie and stopMovie events, and idle events.

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

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

- **Event message hierarchy**

- **Managing memory**

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

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

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

- **preLoad, preLoadMember commands; purgePriority of member property**

- **Working with casts**

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

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

When you want to replace a movie's content by replacing its casts, you can make the casts external casts and overwrite the cast files with new cast files. Of course, each new cast file must have the same name as the cast file it replaces.

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

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

- **castLib** keyword; **purgePriority of member**

- **Sprite properties**

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

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

backColor	memberNum
blend	moveableSprite
castLibNum	pattern
constraint	scoreColor
cursor	scriptNum
editable	spriteBox
foreColor	stretch
height	trails
in	top
lineSize	visible
locH	width
locV	

- **the** and **sprite** keywords; **put**, **puppetSprite**, and **set** commands

• **Generating score**

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

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

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

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

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

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

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

Command	Result
<code>clearFrame</code>	Deletes everything in the current frame, but remains in the frame.
<code>deleteFrame</code>	Deletes the current frame. The next frame then becomes the current frame.
<code>duplicateFrame</code>	Duplicates the current frame and its content. The duplicate frame is inserted after the current frame and then becomes the current frame.

<code>insertFrame</code>	Inserts a copy of the current frame following the current frame. The new frame then becomes the current frame.
<code>updateFrame</code>	Enters the changes made to the current frame. The command then makes a copy of the current frame, inserts it as the next frame, and then advances to the new frame.

For more information about generating score from Lingo, see Chapter 11, "Creating movies from Lingo," in *Learning Lingo*.

- **clearFrame**, and **deleteFrame**, **duplicateFrame**, **insertFrame**, and **updateFrame** commands; **the updateLock** movie property

• **Writing scripts**

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

To:	Do this:
Open a new score script	Choose New Script from the Script pop-up menu; or select a score cell and then click the script preview button. When you open a new score script, the script receives the number of the first available location in the cast window.
Open a new movie script	Click the plus sign in the script window
Assign a score script to one or more cells in the score	Select the cells in the score and then choose the score script number from the Script pop-up menu.
Enter and edit text	Click in the script window and follow basic text editing techniques to type, select, delete, and copy text.
Remove a score script from the cell	Select the cell and then choose Clear Script from the Script pop-up menu.
Change a script window's type	Open the script's Cast Member Properties and then choose a type from the Type pop-up menu.
Cycle through the scripts in the script window	Use the forward and reverse arrows at the top of the script window to advance or back up to the script.
Open the script assigned to a cast member	Click Script in the Cast Member Properties dialog box; select the script in the cast window and then click the script button at the top of the cast window.
Open a score, movie, or parent script	Double-click the script in the cast window.
Open a score script from the score	Select the cell and then click the script preview button.
Open a score, movie, or parent script	Select the script and choose Duplicate from the Edit menu

• **Script window** and **Types of scripts**

● Using Xtras

Xtras are software modules that extend Director's functionality. An Xtra file can contain one or more such modules. There are four types of Xtras:

- Transition Xtra cast members, which supply transitions in addition to predefined transitions available in the Frame Properties:Transition dialog box. After they are used in the score's transition channel, they appear in the cast window the same as any cast member.

An Xtra transition cast member can have its own custom properties, properties dialog box, animated thumbnail, cast window icon, and About box. Open the dialog box that sets properties by opening the Frame Properties:Transition dialog box and then clicking Options.

- Xtra cast members, which can be a wide range of objects such as databases, text managers, or special graphics. They appear in the Insert menu after the Xtra is loaded. An Xtra can create more than one menu item if it is designed to do so.

Add an Xtra cast member to a cast by choosing the Xtra from the Insert menu. Cast member Xtras are sometimes called sprite Xtras because they can be assigned to the score after they are in the cast window.

An Xtra cast member can have its own custom properties, properties dialog box, media editor, animated thumbnail, cast window icon, and About box. Open the dialog box that sets the Xtra's properties by opening the cast member's Properties dialog box and then clicking Options. Open the Xtra's media editor by double-clicking the cast member's thumbnail in the cast window.

- Lingo Xtras, which add Lingo elements to Director's built-in Lingo.
- Tool Xtras, which you can use during authoring. To open a tool Xtra, choose it from the Xtras menu.

Providing Xtras

When Director launches, it automatically registers Xtras that are in either of two places:

- The Xtras folder in the same folder that contains the Director application or projector
- One of the following folders, depending on which platform Director is running on:
For Windows 95 and Windows NT, the Xtras folder in the Program Files\Common Files\Macromedia\Xtras folder

For Windows 3.1, the Windows\Macromed\Xtras folder

For Macintosh and Power Macintosh, the System Folder:Macromedia:Xtras folder.

To make an Xtra available, place its file in one of these folders before you launch Director. (The Xtra can be in a folder within the Xtras folder up to five layers deep.) Director also automatically closes these Xtras when the application quits.

You can also open Lingo Xtras after Director is running by using the `openXlib` command. The Lingo Xtra can be in any folder if you open it this way. However, you must use the `closeXlib` command to close the Xtra after Director is finished with it.

Xtras aren't packaged in projectors. The Xtras must be in an Xtras folder in the same folder as the projector or an Xtras folder that is valid for the current operating system.

If an Xtra that the movie uses is missing, an alert appears when the movie or external cast file opens. For missing Xtra transition cast members, the movie performs a simple cut transition instead. For other missing Xtra cast members, Director displays a red "X" on the stage as a placeholder for the missing Xtra.

Copies of the same Xtra can have different filenames or have the same filename but reside in different folders. If they are used in the same movie, Director detects that such Xtras are duplicates and displays an alert. You can avoid this situation by just deleting

any duplicate Xtras if this occurs.

Creating Xtras by using the new function

You can create a new instance of an Xtra by using the `new` function. The specific way you do this depends on the Xtra's type.

You can create new transition Xtras and cast member Xtras just as you can built-in cast members. For transition cast members, use `new` and the symbol `#transition`. Other cast member Xtras have their own symbols specified by the developer.

For example, a QuickDraw 3D cast member could be given the symbol `#quickDraw3D`. In this case, to create a new cast member, you'd use the statement `new(#quickDraw3D)`.

This statement creates a new instance of the Xtra cast member which has the symbol `#math`:

```
new(#math)
```

After the cast member is created, you can assign it content the same way as you do for other castmembers.

You create new instances of Lingo Xtras by using the `new` function and the term `xtra` as the first parameter. For example, this statement creates a new instance of the Lingo Xtra `stringReader`:

```
set string1 = new(xtra "stringReader")
```

For instances of Lingo Xtras created by using the `new` function, you must set the variable that contains the Xtra to 0 before you use the `closeXlib` command to delete the Lingo Xtra.

See Chapter 11, "Authoring from Lingo," for more information about creating cast members from Lingo.

Checking which Xtras are available

Lingo can tell you how many Xtras are available, the name of each, and what each Xtra contains.

The `number of xtras` property indicates how many Xtras are available in the current movie.

The `name of xtra` property determines the name of a specific Xtra. The `name of xtra` property can be tested and set.

For example, the following repeat loop displays the name of each Xtra in the message window:

```
repeat with counter = 1 to (the number of xtras)
  put the name of xtra counter
end repeat
```

The `showXLib` command displays each Xtra file and its contents. For example, suppose that a Lingo Xtra Friends is in the folder `c:\Xtra Reserve`. If the Xtra file Friends contains the modules Fred and Joe, the `showXlib` command would give the following results:

```
c:\xtra reserve
  xtra Fred
  xtra Joe
```

Use `mMessageList` to display message with information about the Xtra. For example, the statement `put mMessageList(xtra "Fred")` displays information about the Xtra Fred.

For more information on Xtras

For a listing of Xtras available for Director, see the Macromedia Web site at <http://www.macromedia.com>. You will also find information on the Xtra Developer's Kit (XDK).



Lingo elements--navigation

Click a Lingo element for more information:

[continue](#)

[delay](#)

[go loop](#)

[go next](#)

[go previous](#)

[go](#)

[marker](#)

[pause](#)

[play](#)

[play done](#)



Lingo elements--movie control

Click a Lingo element for more information:

[movieName](#)

[on startMovie](#)

[on stopMovie](#)

[play](#)

[play done](#)

[saveMovie](#)

[score](#)

[stage](#)

[stageBottom](#)

[stageColor](#)

[stageLeft](#)

[stageRight](#)

[stageTop](#)

[switchColorDepth](#)

[updateMovieEnabled](#)



Lingo elements--user interaction

Click a Lingo element for more information:

[clickLoc](#)

[clickOn](#)

[commandDown](#)

[controlDown](#)

[cursor](#)

[cursor of sprite](#)

[doubleClick](#)

[key](#)

[keyCode](#)

[keyDownScript](#)

[keyUpScript](#)

[lastClick](#)

[lastKey](#)

[lastRoll](#)

[loc of sprite](#)

[mouseDownScript](#)

[mouseH](#)

[mouseUpScript](#)

[mouseV](#)

[moveableSprite of sprite](#)

[on keyDown](#)

[on keyUp](#)

[on mouseDown](#)

[on mouseUp](#)

[optionDown](#)

[rollOver](#)

[shiftDown](#)



Lingo elements--computer & monitor

Click a Lingo element for more information:

[beep](#)

[beepOn](#)

[colorDepth](#)

[desktopRectList](#)

[floatPrecision](#)

[machineType](#)

[mci](#)

[multiSound](#)

[quit](#)

[restart](#)

[romanLingo](#)

[version](#)



Lingo elements--memory management

Click a Lingo element for more information:

[cancelIdleLoad](#)
[finishIdleLoad](#)
[freeBlock](#)
[idleLoadDone](#)
[idleReadChunkSize](#)
[loaded of member](#)
[memorySize](#)
[movieFileFreeSize](#)
[movieFileSize](#)
[preLoad](#)
[preLoad of member](#)
[preLoadEventAbort](#)
[preLoadMember](#)
[preLoadMode of CastLib](#)
[preLoadRAM](#)
[purgePriority of member](#)
[ramNeeded](#)



Lingo elements--casts

Click a Lingo element for more information:

castLib

fileName of castLib

findEmpty

name of CastLib

number of CastLib

number of castLibs

number of members of castLib

preLoadMode of CastLib

save castLib



Lingo elements--cast members

Click a Lingo element for more information:

[center of member](#)

[crop](#)

[duplicate member](#)

[erase member](#)

[fileName of member](#)

[height of member](#)

[importFileInto](#)

[media of member](#)

[member](#)

[new](#)

[number of member](#)

[rect of member](#)

[type of member](#)



Lingo elements--sprites

Click a Lingo element for more information:

[backColor of sprite](#)

[constrainH](#)

[constraint of sprite](#)

[constrainV](#)

[cursor of sprite](#)

[foreColor of sprite](#)

[height of sprite](#)

[ink of sprite](#)

[left of sprite](#)

[loc of sprite](#)

[memberNum of sprite](#)

[moveableSprite of sprite](#)

[paletteRef](#)

[puppetSprite](#)

[scriptNum of sprite](#)

[stretch of sprite](#)

[type of sprite](#)

[updateStage](#)

[visible of sprite](#)



Lingo elements--frames

Click a Lingo element for more information:

[frame](#)
[frameLabel](#)
[framePalette](#)
[frameScript](#)
[frameSound1](#)
[frameSound2](#)
[frameTempo](#)
[frameTransition](#)
[label](#)
[marker](#)
[on enterFrame](#)
[on exitFrame](#)
[puppetPalette](#)
[puppetTempo](#)
[puppetTransition](#)



Lingo elements--score generation

Click a Lingo element for more information:

[beginRecording](#)

[clearFrame](#)

[deleteFrame](#)

[duplicateFrame](#)

[endRecording](#)

[insertFrame](#)

[updateFrame](#)



Lingo elements--external files

Click a Lingo element for more information:

[closeDA](#)

[closeResFile](#)

[closeXlib](#)

[copyToClipboard](#)

[fileName of castLib](#)

[fileName of member](#)

[getNthFileNameInFolder](#)

[importFileInto](#)

[movieFileFreeSize](#)

[moviePath](#)

[open](#)

[openDA](#)

[openResFile](#)

[openXlib](#)

[pathName](#)

[searchCurrentFolder](#)

[searchPath](#)

[setCallback](#)

[showResFile](#)

[showXlib](#)

[sound playFile](#)

[xFactoryList](#)



Lingo elements--movie in a window

Click a Lingo element for more information:

[closeWindow](#)

[drawRect of window](#)

[fileName of window](#)

[forget window](#)

[modal of window](#)

[moveToBack](#)

[moveToFront](#)

[open window](#)

[rect of window](#)

[title of window](#)

[visible of window](#)

[window](#)

[windowList](#)



Lingo elements--parent scripts

Click a Lingo element for more information:

[actorList](#)

[ancestor](#)

[birth](#)

[new](#)

[property](#)



Lingo elements--lists

Click a Lingo element for more information:

[\[\] \(list brackets\)](#)

[add](#)

[addAt](#)

[addProp](#)

[append](#)

[count](#)

[deleteAt](#)

[deleteProp](#)

[findPos](#)

[findPosNear](#)

[getaProp](#)

[getAt](#)

[getLast](#)

[getOne](#)

[getPos](#)

[getProp](#)

[getPropAt](#)

[listP](#)



Lingo elements--code structures & syntax

Click a Lingo element for more information:

[case](#)

[end case](#)

[end if](#)

[exit](#)

[if ... then ...](#)

[if ... then ... else](#)

[repeat while](#)

[end repeat](#)



Lingo elements--strings

Click a Lingo element for more information:

[&](#)

[&&](#)

[char...of](#)

[chars](#)

[contains](#)

[delete](#)

[EMPTY](#)

[item...of](#)

[itemDelimiter](#)

[last](#)

[length](#)

[line...of](#)

[number of items in](#)

[number of lines in](#)

[stringP](#)

[word...of](#)



Lingo elements--math & logical operators

Click a Lingo element for more information:

<u>- (minus sign)</u>	<u>abs</u>
<u>() (parenthesis)</u>	<u>and</u>
<u>* (multiplication)</u>	<u>atan</u>
<u>+ (addition)</u>	<u>charToNum</u>
<u>/ (division)</u>	<u>cos</u>
<u>> (greater than)</u>	<u>exp</u>
<u>> = (greater than or equal to)</u>	<u>FALSE</u>
<u>< (less than)</u>	<u>float</u>
<u>< = (less than or equal to)</u>	<u>floatP</u>
<u><> (not equal)</u>	<u>integer</u>
<u>= (equal sign)</u>	<u>integerP</u>
	<u>mod</u>
	<u>or</u>
	<u>sqrt</u>
	<u>TRUE</u>



Lingo elements--fields

Click a Lingo element for more information:

[alignment of member](#)

[char...of](#)

[chars](#)

[charToNum](#)

[contains](#)

[delete](#)

[dropShadow of member](#)

[editable of member](#)

[editable of sprite](#)

[font of member](#)

[fontSize of member](#)

[fontStyle of member](#)

[foreColor of member](#)

[height of member](#)

[hilite](#)

[item...of](#)

[lineHeight of member](#)

[locToCharPos](#)

[locVToLinePos](#)

[number of chars in](#)

[number of items in](#)

[number of words in](#)

[offset](#)

[scrollByPage](#)

[string](#)

[text of member](#)

[wordWrap of member](#)

[word...of](#)



Lingo elements--digital video

Click a Lingo element for more information:

[controller of member](#)

[duration of member](#)

[frameRate of member](#)

[loop of member](#)

[movieRate of sprite](#)

[timeScale of member](#)

[trackCount\(member\)](#)

[trackCount\(sprite\)](#)

[trackEnabled](#)

[trackNextKeyTime](#)

[trackNextSampleTime](#)

[trackPreviousKeyTime](#)

[trackPreviousSampleTime](#)

[trackStartTime\(member\)](#)

[trackStartTime\(sprite\)](#)

[trackStopTime\(member\)](#)

[trackStopTime\(sprite\)](#)

[trackText](#)

[trackType \(member\)](#)

[trackType \(sprite\)](#)



Lingo elements--sound

Click a Lingo element for more information:

[puppetSound](#)

[sound close](#)

[sound fadeIn](#)

[sound fadeOut](#)

[sound of member](#)

[sound playFile](#)

[sound stop](#)

[soundBusy](#)

[soundEnabled](#)

[volume of sprite](#)



Lingo elements--interface elements

Click a Lingo element for more information:

[alert](#)

[buttonType](#)

[checkMark of menuitem](#)

[enabled of menuitem](#)

[installMenu](#)

[menu](#)

[name of menu](#)

[name of menuitem](#)

[number of menuitems](#)

[number of menus](#)

[script of menuitem](#)

Lingo elements--operators

Click a Lingo element for more information:

(pound sign)

- (minus sign)

-- (comment delimiter)

& (concatenator)

&& (concatenator)

* (multiplication)

+ (addition)

/ (division)

< (less than)

< = (less than or equal to)

<> (not equal)

= (equal sign)

> (greater than)

> = (greater than or equal to)

[] (list brackets)

" (string constant)

- (continuation symbol)

() [parentheses]



Lingo elements--puppets

Click a Lingo element for more information:

[puppet of sprite](#)

[puppetPalette](#)

[puppetSound](#)

[puppetSprite](#)

[puppetTempo](#)

[puppetTransition](#)

[updateStage](#)



Lingo elements--time

Click a Lingo element for more information:

[date](#)

[delay](#)

[framesToHMS](#)

[HMStoFrames](#)

[startTimer](#)

[ticks](#)

[time](#)

[timer](#)

[timeoutKeyDown](#)

[timeoutLapsed](#)

[timeoutMouse](#)

[timeoutScript](#)



Lingo elements--variables

Click a Lingo element for more information:

[clearGlobals](#)

[property](#)

[put](#)

[set...to and set...=](#)

[showGlobals](#)

[showLocals](#)



Lingo elements--A

Click a Lingo element for more information:

[abbr](#)

[abbrev](#)

[abbreviated](#)

[abort](#)

[abs](#)

[activateWindow](#)

[activeWindow](#)

[actorList](#)

[add](#)

[addAt](#)

[addProp](#)

[after](#)

[alert](#)

[alignment of member](#)

[ancestor](#)

[and](#)

[append](#)

[atan](#)

[autoTab of member](#)



Lingo elements--B

Click a Lingo element for more information:

[backColor of member](#)

[backColor of sprite](#)

[BACKSPACE](#)

[beep](#)

[beepOn](#)

[before](#)

[beginRecording](#)

[birth](#)

[blend of sprite](#)

[border of member](#)

[bottom of sprite](#)

[boxDropShadow of member](#)

[boxType of member](#)

[buttonStyle](#)

[buttonType](#)



Lingo elements--C

Click a Lingo element for more information:

[cancelIdleLoad](#)

[case](#)

[the castLib](#)

[the castLibNum of sprite](#)

[center of member](#)

[centerStage](#)

[changeArea of member](#)

[channelCount](#)

[char...of](#)

[charPosToLoc](#)

[chars](#)

[charToNum](#)

[checkBoxAccess](#)

[checkBoxType](#)

[checkMark of menuItem](#)

[chunkSize of member](#)

[clearFrame](#)

[clearGlobals](#)

[clickLoc](#)

[clickOn](#)

[close window](#)

[closeDA](#)

[closeResFile](#)

[closeWindow](#)

[closeXlib](#)

[crop](#)

[colorDepth](#)

[colorQD](#)

[commandDown](#)

[constrainH](#)

[constraint of sprite](#)

[constrainV](#)

[contains](#)

[continue](#)

[controlDown](#)

[controller of member](#)

[copyToClipboard](#)

[cos](#)

[count](#)

[crop of member](#)

[cursor](#)

[cursor of sprite](#)

Lingo elements--D

Click a Lingo element for more information:

<u>date</u>	<u>digitalVideo member video</u>
<u>deactivateWindow</u>	<u>digitalVideo sprite movieRate</u>
<u>delay</u>	<u>digitalVideo sprite movieTime</u>
<u>delete</u>	<u>digitalVideo sprite startTime</u>
<u>deleteAt</u>	<u>digitalVideo sprite stopTime</u>
<u>deleteFrame</u>	<u>digitalVideo sprite volume</u>
<u>deleteOne</u>	<u>digitalVideoTimeScale</u>
<u>deleteProp</u>	<u>digitalVideoType</u>
<u>depth of member</u>	<u>directToStage of member</u>
<u>desktopRectList</u>	<u>do</u>
<u>digitalVideo</u>	<u>done</u>
<u>digitalVideo member center</u>	<u>dontPassEvent</u>
<u>digitalVideo member controller</u>	<u>doubleClick</u>
<u>digitalVideo member crop</u>	<u>down</u>
<u>digitalVideo member directToStage</u>	<u>drawRect of window</u>
<u>digitalVideo member duration</u>	<u>dropShadow of member</u>
<u>digitalVideo member frameRate</u>	<u>duplicateFrame</u>
<u>digitalVideo member loop</u>	<u>duplicate(list)</u>
<u>digitalVideo member pausedAtStart</u>	<u>duplicate member</u>
<u>digitalVideo member preload</u>	<u>duration of member</u>
<u>digitalVideo member sound</u>	



Lingo elements--E

Click a Lingo element for more information:

[editable of member](#)

[editable of sprite](#)

[else](#)

[EMPTY](#)

[emulateMultiButtonMouse](#)

[enabled of menuItem](#)

[end](#)

[end case](#)

[end repeat](#)

[endRecording](#)

[ENTER](#)

[enterFrame](#)

[erase member](#)

[exit](#)

[exit repeat](#)

[exitFrame](#)

[exitLock](#)

[exp](#)

Lingo elements--F

Click a Lingo element for more information:

[factory](#)

[fadeIn](#)

[fadeOut](#)

[FALSE](#)

[field](#)

[fileName of castLib](#)

[fileName of member](#)

[fileName of window](#)

[filled](#)

[findEmpty](#)

[findPos](#)

[findPosNear](#)

[finishIdleLoad](#)

[fixStageSize](#)

[float](#)

[floatP](#)

[floatPrecision](#)

[font of member](#)

[fontSize of member](#)

[fontStyle of member](#)

[foreColor of member](#)

[foreColor of sprite](#)

[forget window](#)

[frame](#)

[frameLabel](#)

[framePalette](#)

[frameRate of member](#)

[frameScript](#)

[frameSound1](#)

[frameSound2](#)

[framesToHMS](#)

[frameTempo](#)

[frameTransition](#)

[freeBlock](#)

[freeBytes](#)

[frontWindow](#)



Lingo elements--G

Click a Lingo element for more information:

[getaProp](#)

[getAt](#)

[getLast](#)

[getNthFileNameInFolder](#)

[getOne](#)

[getPos](#)

[getProp](#)

[getPropAt](#)

[global](#)

[go](#)

[go loop](#)

[go next](#)

[go previous](#)



Lingo elements--H

Click a Lingo element for more information:

halt

height of member

height of sprite

hilite

hilite of member

HMStoFrames



Lingo elements--I

Click a Lingo element for more information:

[idle](#)

[idleHandlerPeriod](#)

[idleLoadDone](#)

[idleLoadMode](#)

[idleLoadPeriod](#)

[idleLoadTag](#)

[idleReadChunkSize](#)

[if](#)

[ilk](#)

[importFileInto](#)

[in](#)

[inflate rect](#)

[ink of sprite](#)

[items](#)

[insertFrame](#)

[inside](#)

[inside point](#)

[installMenu](#)

[instance](#)

[integer](#)

[integerP](#)

[intersect](#)

[intersect rect](#)

[intersects](#)

[into](#)

[item...of](#)

[itemDelimiter](#)



Lingo elements--J

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



Lingo elements--K

Click a Lingo element for more information:

[key](#)

[keyCode](#)

[keyDown](#)

[keyDownScript](#)

[the keyPressed](#)

[keyUp](#)

[keyUpScript](#)



Lingo elements--L

Click a Lingo element for more information:

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

Lingo elements--M

Click a Lingo element for more information:

[machineType](#)

[map](#)

[map point](#)

[map rect](#)

[margin](#)

[marker](#)

[mAtFrame](#)

[max](#)

[maxInteger](#)

[mci](#)

[mDescribe](#)

[mDispose](#)

[me](#)

[media of member](#)

[member](#)

[member backColor](#)

[member memberType](#)

[member depth](#)

[member fileName](#)

[member foreColor](#)

[member height](#)

[member hilite](#)

[member loaded](#)

[member name](#)

[member number](#)

[member palette](#)

[member picture](#)

[member purgePriority](#)

[member rect](#)

[member regPoint](#)

[member scriptText](#)

[member text](#)

[member width](#)

[memberNum of sprite](#)

[memorySize](#)

[menu](#)

[menuItem](#)

[mRespondsTo](#)

[menuItems](#)

[menus](#)

[method](#)

[mGet](#)

[min](#)

[mInstanceRespondsTo](#)

[mMessageList](#)

[mName](#)

[mNew](#)

[mod](#)

[modal of window](#)

[modified of member](#)

[mouseCast](#)

[mouseChar](#)

[mouseDown](#)

[mouseDownScript](#)

[mouseH](#)

[mouseItem](#)

[mouseLine](#)

[mouseUp](#)

[mouseUpScript](#)

[mouseV](#)

[mouseWord](#)

[move member](#)

[moveableSprite of sprite](#)

[moveToBack](#)

[moveToFront](#)

[moveWindow](#)

[movie](#)

[movieFileFreeSize](#)

[movieFileSize](#)

[movieName](#)

[moviePath](#)

[movieRate of sprite](#)

[movieTime of sprite](#)

[mPerform](#)

[mPut](#)

[multiSound](#)



Lingo elements--N

Click a Lingo element for more information:

<u>name of CastLib</u>	<u>number of CastLib</u>
<u>name of member</u>	<u>number of castLibs</u>
<u>name of menu</u>	<u>number of chars in</u>
<u>name of menuItem</u>	<u>number of items in</u>
<u>name of window</u>	<u>number of lines in</u>
<u>name of xtra</u>	<u>number of member</u>
<u>new</u>	<u>number of members</u>
<u>next</u>	<u>number of members of castLib</u>
<u>next repeat</u>	<u>number of menuItems</u>
<u>not</u>	<u>number of menus</u>
<u>nothing</u>	<u>number of words in</u>
	<u>number of xtras</u>
	<u>numToChar</u>



Lingo elements--O

Click a Lingo element for more information:

<u>objectP</u>	<u>on resizeWindow</u>
<u>of</u>	<u>on rightMouseDown</u>
<u>offset</u>	<u>on rightMouseUp</u>
<u>offset rect</u>	<u>on startMovie</u>
<u>on</u>	<u>on stepMovie</u>
<u>on activateWindow</u>	<u>on stopMovie</u>
<u>on closeWindow</u>	<u>on timeOut</u>
<u>on deactivateWindow</u>	<u>on zoomWindow</u>
<u>on enterFrame</u>	<u>open</u>
<u>on exitFrame</u>	<u>openWindow</u>
<u>on idle</u>	<u>open window</u>
<u>on keyDown</u>	<u>openDA</u>
<u>on keyUp</u>	<u>openResFile</u>
<u>on mouseDown</u>	<u>openXlib</u>
<u>on mouseUp</u>	<u>optionDown</u>
<u>on moveWindow</u>	<u>or</u>
<u>on openWindow</u>	<u>otherwise</u>

Lingo elements--P

Click a Lingo element for more information:

<u>pageHeight of member</u>	<u>preLoad</u>
<u>palette of member</u>	<u>preLoad of member</u>
<u>paletteMapping</u>	<u>preLoadEventAbort</u>
<u>paletteRef</u>	<u>preLoadMember</u>
<u>param</u>	<u>preLoadMode of CastLib</u>
<u>paramCount</u>	<u>preLoadMovie</u>
<u>pass</u>	<u>preLoadRAM</u>
<u>pasteClipboardInto</u>	<u>previous</u>
<u>pathName</u>	<u>printFrom</u>
<u>pattern</u>	<u>property</u>
<u>pause</u>	<u>puppet of sprite</u>
<u>pausedAtStart of member</u>	<u>puppetPalette</u>
<u>pauseState</u>	<u>puppetSound</u>
<u>perFrameHook</u>	<u>puppetSprite</u>
<u>pi</u>	<u>puppetTempo</u>
<u>picture of member</u>	<u>puppetTransition</u>
<u>pictureP</u>	<u>purgePriority of member</u>
<u>platform</u>	<u>put</u>
<u>play</u>	<u>put...after</u>
<u>play done</u>	<u>put...before</u>
<u>playFile</u>	<u>put...into</u>
<u>point</u>	
<u>power</u>	



Lingo elements--Q

Click a Lingo element for more information:

[quickTimePresent](#)

[quit](#)

[QUOTE](#)



Lingo elements--R

Click a Lingo element for more information:

[ramNeeded](#)

[random](#)

[randomSeed](#)

[rect](#)

[rect of member](#)

[rect of sprite](#)

[rect of window](#)

[rect point](#)

[regPoint of member](#)

[repeat while](#)

[repeat with](#)

[repeat with...in list](#)

[repeat with...down to](#)

[resizeWindow, on](#)

[restart](#)

[result](#)

[RETURN](#)

[return](#)

[rightMouseDown](#)

[rightMouseDown](#)

[rightMouseUp](#)

[rightMouseUp](#)

[right of sprite](#)

[rollOver](#)

[romanLingo](#)

Lingo elements--S

Click a Lingo element for more information:

<u>sampleRate</u>	<u>sort</u>
<u>sampleSize</u>	<u>sound close</u>
<u>save castLib</u>	<u>sound fadeIn</u>
<u>saveMovie</u>	<u>sound fadeOut</u>
<u>score</u>	<u>sound of member</u>
<u>scoreColor of sprite</u>	<u>sound playFile</u>
<u>scoreSelection</u>	<u>sound stop</u>
<u>script of menuItem</u>	<u>soundBusy</u>
<u>scriptNum of sprite</u>	<u>soundEnabled</u>
<u>scriptsEnabled</u>	<u>soundLevel</u>
<u>scriptText of member</u>	<u>sourceRect</u>
<u>scriptType</u>	<u>sprite</u>
<u>scrollByLine</u>	<u>spriteBox</u>
<u>scrollByPage</u>	<u>sprite...intersects</u>
<u>scrollTop</u>	<u>sprite...within</u>
<u>searchCurrentFolder</u>	<u>sqrt</u>
<u>searchPath</u>	<u>stage</u>
<u>searchPaths</u>	<u>stageBottom</u>
<u>selection</u>	<u>stageColor</u>
<u>selection of castLib</u>	<u>stageLeft</u>
<u>selEnd</u>	<u>stageRight</u>
<u>selStart</u>	<u>stageTop</u>
<u>setaProp</u>	<u>startMovie</u>
<u>setAt</u>	<u>starts</u>
<u>setCallback</u>	<u>startTimer</u>
<u>setProp</u>	<u>stepMovie</u>
<u>set...to and set...=</u>	<u>stillDown</u>
<u>setTrackEnabled</u>	<u>stop</u>
<u>shapeType</u>	<u>stopMovie</u>
<u>shiftDown</u>	<u>stretch of sprite</u>
<u>short</u>	<u>string</u>
<u>showGlobals</u>	<u>stringP</u>
<u>showLocals</u>	<u>switchColorDepth</u>
<u>showResFile</u>	<u>symbolP</u>
<u>showXlib</u>	
<u>shutDown</u>	
<u>sin</u>	
<u>size of member</u>	

Lingo elements--T

Click a Lingo element for more information:

[TAB](#)

[tan](#)

[tell](#)

[text of member](#)

[the](#)

[then](#)

[ticks](#)

[time](#)

[timeoutKeyDown](#)

[timeoutLapsed](#)

[timeoutLength](#)

[timeoutMouse](#)

[timeoutPlay](#)

[timeoutScript](#)

[timer](#)

[timeScale of member](#)

[title of window](#)

[titleVisible of window](#)

[trackType \(sprite\)](#)

[trails of sprite](#)

[transitionType of member](#)

[TRUE](#)

[to](#)

[top of sprite](#)

[trace](#)

[traceLoad](#)

[traceLogFile](#)

[trackCount\(member\)](#)

[trackCount\(sprite\)](#)

[trackEnabled](#)

[trackNextKeyTime](#)

[trackNextSampleTime](#)

[trackPreviousKeyTime](#)

[trackPreviousSampleTime](#)

[trackStartTime\(member\)](#)

[trackStartTime\(sprite\)](#)

[trackStopTime\(member\)](#)

[trackStopTime\(sprite\)](#)

[trackText](#)

[trackType \(member\)](#)

[type of member](#)

[type of sprite](#)

Lingo elements--U

Click a Lingo element for more information:

[union rect](#)

[unLoad](#)

[unLoadMember](#)

[unloadMovie](#)

[updateFrame](#)

[updateLock](#)

[updateMovieEnabled](#)

[updateStage](#)



Lingo elements--V

Click a Lingo element for more information:

[value](#)

[version](#)

[video of member](#)

[videoForWindowsPresent](#)

[visible of sprite](#)

[visible of window](#)

[voidP](#)

[volume of sound](#)

[volume of sprite](#)



Lingo elements--W

Click a Lingo element for more information:

[when...then](#)

[while](#)

[width of member](#)

[width of sprite](#)

[window](#)

[windowList](#)

[windowPresent](#)

[windowType of window](#)

[with](#)

[within](#)

[word...of](#)

[wordWrap of member](#)



Lingo elements--X

Click a Lingo element for more information:

[xFactoryList](#)

[xtra](#)

[xtras](#)



Lingo elements--Y

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



Lingo elements--Z

Click a Lingo element for more information:

[zoomBox](#)

[zoomWindow](#)

- **# (pound sign)**

Syntax: #*symbolName*

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

The valid operations on symbols are:

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

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

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

Example:

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

```
set state = #Playing
```

- **symbolP** function

- **- (minus sign)**

Syntax (negation): *-expression*

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

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

Syntax (subtraction): *expression1 - expression2*

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

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

Example 1 (negation):

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

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

The result is -5.

Example 2 (subtraction):

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

```
put 5 - 2
```

The result is 3, which is an integer.

Example 3 (subtraction):

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

```
put 3.25 - 1.5
```

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

- **-- (comment delimiter)**

Syntax: -- [*comment*]

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

Example:

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

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

• **& (concatenator)**

Syntax: *expression1* & *expression2*

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

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

Example 1:

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

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

The result is the string "abracadabra".

Example 2:

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

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

• **&& (concatenator)**

Syntax: *expression1* && *expression2*

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

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

Example 1:

This statement concatenates the strings "abra" and "cadabra", and inserts a space between the two:

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

The result is the string "abra cadabra".

Example 2:

This statement concatenates the strings "Today is" and today's date in the long format, and inserts a space between the two:

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

If today's date were Tuesday, March 15, 1996, the result would be the string `Tuesday, March 15, 1996`.

• () (parentheses)

Syntax: (*expression*)

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

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

Examples:

These statements use the grouping operator to change the order in which operations occur. The result appears below each statement:

```
put (2 + 3) * (4 + 5)
```

```
-- 45
```

```
put 2 + (3 * (4 + 5))
```

```
-- 29
```

```
put 2 + 3 * 4 + 5
```

```
-- 19
```


● * (multiplication)

Syntax: *expression1* * *expression2*

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

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

Example 1:

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

```
put 2 * 3
```

The result is 6, which is an integer.

Example 2:

This statement multiplies the floating point numbers 2.0 and 3.1414, and then displays the result in the message window:

```
put 2.0 * 3.1416
```

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

- **+ (addition)**

Syntax: *expression1* + *expression2*

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

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

Example 1:

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

```
put 2 + 3
```

The result is 5, which is an integer.

Example 2:

This statement adds the floating point number 2.5 and 3.25, and then displays the result in the message window:

```
put 2.5 + 3.25
```

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

• / (division)

Syntax: *expression1* / *expression2*

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

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

Example 1:

This statement divides the integer 22 by 7, and then displays the result in the message window:

```
put 22 / 7
```

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

Example 2:

This statement divides the floating point number 22.0 by 7.0, and then displays the result in the message window:

```
put 22.0 / 7.0
```

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

- **< (less than)**

Syntax: *expression1* < *expression2*

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

This operator can compare strings as well as integers and floating point numbers.

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

- **<= (less than or equal to)**

Syntax: *expression1* <= *expression2*

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

This operator can compare strings as well as integers and floating point numbers.

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

- **<> (not equal)**

Syntax: *expression1* <> *expression2*

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

This operator can compare strings as well as integers and floating point numbers.

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

- **= (equal sign)**

Syntax: *expression1* = *expression2*

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

This operator can compare strings as well as integers and floating point numbers.

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

- **> (greater than)**

Syntax: *expression1* > *expression2*

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

This operator can compare strings as well as integers and floating point numbers.

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

- **>= (greater than or equal to)**

Syntax: *expression1* >= *expression2*

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

This operator can compare strings as well as integers and floating point numbers.

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

• [] (list operators)

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

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

There are four types of lists:

- Unsorted linear lists
- Sorted linear lists
- Unsorted property lists
- Sorted property lists.

Each entry in a linear list is a single value that has no other property associated with it. Each entry in a property list consists of a value and a property. The property appears before the value and is separated from the value by a colon. You cannot store a property in a linear list. When using strings as entries in a list, enclose the string in quotation marks.

For example, `[6, 3, 8]` is a linear list. The numbers have no properties associated with them. However, `[#gears:6, #balls:3, #ramps:8]` is a property list. Each number has a property, in this case a piece of machinery, associated with it. This property list could be useful for tracking how many of each piece of machinery are currently on the stage in the mechanical simulation. Properties can appear more than once in a property list.

Lists can be sorted in alphanumeric order. A sorted linear list is ordered by the values in the list. A sorted property list is in order by the properties in the list. You sort a list by using the appropriate command for a linear list or property list.

Property lists are case-sensitive.

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

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

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

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

Example 1:

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

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

Example 2:

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

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

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

Example 3:

This statement creates an empty linear list:

```
set x = []
```

Example 4:

This statement creates an empty property list:

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

• **add, addAt, append, count, deleteAt, findPos, findPosNear, getaProp, getAt, getLast, getPos, setAt, setaProp, sort** commands; **ilk, list, min, max** functions

- **" (string constant)**

Syntax: "

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

Example:

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

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

- **↵ (continuation symbol)**

Syntax: *first part of a statement on this line ↵
 second part of statement on next line ↵
 third part of statement*

This special character, when used as the last character in a line, indicates that the statement continues on the next line. Lingo then interprets the lines as one continuous statement. You can do this on several successive lines.

Create this character by pressing Alt+Enter.

Example:

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

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

- **abbr, abbrev, abbreviated**

- the **date** and **time** functions.

● **abort**

Syntax: `abort`

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

The `abort` command does not quit Director.

Example:

This statement has Lingo 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
```

• **abs**

Syntax: abs (*numericExpression*)

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

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

Example 1:

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

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

Example 2:

This statement determines whether the absolute value of the difference between the current mouse position and the value of the variable `startV` is greater than 30. If it is, the foreground color of sprite 6 is changed.

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


- **activeWindow**

Syntax: the activeWindow

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

Example: to be provided

- **actorList**

Syntax: `the actorList`

This property is a list of all child objects currently in the movie. All objects in the `actorList` receive an `enterFrame` message when the playback head enters a new frame.

You can clear child objects from the `actorList` by setting the `actorList` to `[],` which is an empty list.

Director doesn't clear the contents of the `actorList` when branching to another movie, which could cause unpredictable behavior in the new movie. If you don't want child objects in the current movie to be carried over into the new movie, insert a statement that clears the `actorList` in the `on startMovie` handler of the new movie.

Example 1:

This statement creates a child object from the parent script `Moving Ball`. All three values are parameters that the script requires:

```
add the actorList, new(script "MovingBall", 1, 200, 200)
```

Example 2:

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

```
put the actorList
```

Example 3:

This statement clears the `actorList`:

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

- **new command; parent-child scripts**

- **add**

Syntax: `add linearList, value`

This command adds the value specified by *value* to the linear list specified by *linearList*. For a sorted list, the value is placed in its proper order. For an unsorted list, the value is added to the end of the list.

Example 1:

This statement adds the value 2 to the list named `bids`. The resulting list is [3, 4, 1, 2]:

```
set bids = [3, 4, 1]
add bids, 2
```

Example 2:

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

```
add bids, 2
```

• **addAt**

Syntax: `addAt list , position , value`

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

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

Example:

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

```
set bids = [3, 2, 4, 5, 6, 7]
addAt bids, 4, 8
```

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

• addProp

Syntax: `addProp list , property , value`

This command adds the property specified by *property* and its value specified by *value* to the property list specified by *list*. For an unsorted list, the value is added to the end of the list. For a sorted list, the value is placed in its proper order.

When the property already exists in the list, Lingo creates a duplicate property. You can avoid duplicate properties by using the `setaProp` command to change the new entry's property.

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

Example 1:

This statement adds the property named `kayne` and its assigned value 3 to the property list named `bids`, which contains [`#gee: 4, #ohasi: 1`]. Because the list is sorted, the new entry is placed in alphabetical order:

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

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

Example 2:

This statement adds the entry `kayne: 7` to the list named `bids`, which now contains [`#gee: 4, #kayne: 3, #ohasi: 1`]. Because the list already contains the property `kayne`, Lingo creates a duplicate property:

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

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

- **after**

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

- **alert**

Syntax: `alert message`

This command causes a system beep and displays an alert dialog box containing the string specified by *message*, and an OK button. This command is useful for providing error messages in your movie. The message can contain up to 255 characters.

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

Example 1:

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

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

Example 2:

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

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

- **string**

- **alignment of member**

Syntax: the alignment of member *whichCastmember*
 the alignment of member *whichCastmember*

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

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

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

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

Example:

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

```
put the alignment of member "Rokujo Speaks" into ↵
  alignment
```

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

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

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

- **text of member** property; **font of member**, **lineHeight of member**, **fontSize of member**, and **fontStyle of member** text properties; **& (ampersand)** and **&& (double ampersand)** field operators

• ancestor

Syntax: `property ancestor`

The ancestor property allows child objects to use handlers that are not contained within the parent script. The `ancestor` property is typically used with two or more parent scripts. This is useful when you want child objects to share certain behaviors that are inherited from an ancestor, while differing in other behaviors that are inherited from the parents.

The `ancestor` property is typically assigned in the child object's `new` handler within the parent script. When you send a message to a child object that does not have a defined handler, that message is forwarded to the script defined by the ancestor property.

The ancestor property can be changed "on the fly." This allows you to significantly change behaviors and properties for a large group of objects with a single command.

For a complete discussion of this topic, see "Parent Scripts and Child Objects," in *Learning Lingo*.

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

```
set propertyVariable to value
```

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

```
set the legCount of me to 4
```

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

```
put the legCount of me into myLegCount
```

The following scripts present an example of using the ancestor property. Each of these scripts is a cast member. Using the ancestor script `Animal` and the parent scripts `Dog` and `Man`, they interact with one another to define objects.

Example 1:

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

```
property breed, ancestor

on new me
    set breed = "Mutt"
    set the ancestor of me to new(script "Animal")
```

```
    set the legCount of me to 4
    return me
end new
```

Example 2:

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

```
property race, ancestor

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

Example 3:

The third script Animal stores the property variable `legCount` for each child object and defines the `eat` handler:

```
property legCount

on new me
    return me
end new

on eat me, what
    put "Eating " & what
end
```

Example 4:

The fourth script creates a child object of Man, displays its variables in the message window, and calls the `eat` handler and displays it in the message window. Since the `eat` handler is not in the parent script Man, Lingo finds the `eat` handler in the ancestor script Animal:

```
set manChild to new(script "man")
put the legCount of manChild
--> 2

put the race of manChild
--> "African"

eat manChild, "apple"
--> "Eating apple"
```

Example 5:

This parent script creates a set of independent timers. Each timer's "countdown" time is controlled by the `intervalInSeconds` argument. The action that each timer performs when its time is up is controlled by the `actionItem` argument.

Each timer that is created repeatedly performs its assigned task each time its assigned time passes.

You can use the Lingo "actorList" to service objects each time the playback head advances. By creating objects from a common parent script, each object can have separate time settings and separate actions that occur when they hit their target time.

```
-- script name: "timer script"
property interval, currentSegment, startTime, actionItem

on new me, intervalInSeconds, handlerToExecute
  set interval = intervalInSeconds * 60
  set currentSegment = 0
  set startTime = the ticks
  set actionItem = handlerToExecute
  return me
end new

on AdvanceClock me
  set currentSegment to currentSegment + 1
  do actionItem
end AdvanceClock
```

This can be called by syntax such as:

```
add the actorList, new(script "timer script", 2, "beep")
add the actorList, new(script "timer script", 15, ↵
"set the stageColor to random(255)")
```

The first statement has the object beep every 2 seconds, while the second randomly changes the color of the screen every fifteen seconds. The advantage of using parent scripts is that all the data is self-contained for each object. You can dispose of each object by removing it from the actorList. If you are using many timers, then giving each an identifying property allows you to iterate through the actorList, identifying that unique object.

Example 6:

These statements make the parent script Animal the ancestor script for the child object designated by `me`:

```
set theDog to new(script "Animal")
set the ancestor of me to theDog
```

- **new** function; **me** and **property** keywords

- **and**

Syntax: *logicalExpression1* and *logicalExpression2*

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

The precedence level of this logical operator is 4.

Example 1:

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

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

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

Example 2:

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

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

- **not** and **or** logical operators

- **append**

Syntax: `append list, value`

This command adds the specified value to the end of the list, regardless of the list's type. This differs from the `add` command, which adds a value to a sorted list in accordance with the list's order.

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

Example:

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

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

- **add** command

- **atan**

Syntax: `atan (number)`

This function calculates the arctangent of the specified number. The result is between $\pi/2$ and $+\pi/2$.

Example:

This expression gives the arctangent of $\pi/4$ radians:

```
atan (pi()/4.0)
```

- **autoTab of member**

Syntax: the autoTab of member *whichCastmember*

This field cast member property determines whether the editable field that follows the field cast member specified by *whichCastmember* becomes the active field after the user presses Tab.

- When the autoTab of member is TRUE, pressing Tab makes the next editable field on stage the active field.
- When the autoTab of member is FALSE, pressing Tab doesn't make the next editable field on stage the active field.

Example:

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

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

● **backColor of member**

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

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

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

Example:

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

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

Example:

This handler converts a list that contains percentages of red, green, and blue that make up a color. It returns an integer that represents the color on a 32-bit color monitor:

```
on convert theList
  set red to getAt(theList, 1) * 255 * 65536 * / 100.00
  set green to getAt(theList, 2) * 255 * 256 / 100.00
  set blue to getAt(theList, 3) * 255 * 1 * / 100.00
  return red + green + blue
end
```

This statement calls the `convert` handler to obtain the `backColor` value for pure blue on a 32-bit monitor and displays the result in the message window:

```
put convert ([0, 0, 100])
```

Example:

This handler converts a list that contains red, green, and blue values from 0 to 255 and generates an integer that represents the color on a 16-bit color monitor:

```
on convert16 theList
  set normedList = theList /8
  set red to integer(getAt(normedList, 1)) * 1024
  set green to integer(getAt(normedList, 2)) * 32
  set blue to integer(getAt(normedList, 1))
  return red + green + blue
end
```

This statement calls the `convert16` handler to change the `backColor` of field cast member 1 to bright green:

```
set the backColor of member 1 to convert16([0, 255, 0])
```


- **backColor of sprite**

Syntax: the backColor of sprite *whichSprite*

This sprite property determines the background color of the sprite specified by *whichSprite*. The sprite must be a puppet before you can set its background color using Lingo. Setting the backColor using a Lingo script is the same as choosing the background color from the tool palette when the sprite is selected on the stage.

The background color applies only to 1-bit bitmap and shape cast members. It does not affect how a field or button cast member is displayed. An 8-bit bitmap is affected, but generally not in a useful way.

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

When you set this property within a script while the playback head is not moving, be sure to use the command `updateStage` to redraw the stage. If you are changing several sprite properties--or several puppet sprites--you only have to use one `updateStage` command at the end of all the changes.

One use of the backColor of sprite property that works consistently with 8-bit bitmap sprites is specifying which color is to be made transparent using the score ink effect Background Transparent. This is particularly useful when creating or importing anti-aliased graphics or objects from a 3D rendering program for use over video.

Using a black stage color that is defined as the overlay color by the video card, as well as having images that are anti-aliased against a black background, usually works best. This will produce a dark gray shadow behind the graphic when used over a video source. This is the least objectionable shadow color.

The backColor of sprite property can be tested and set.

Example 1:

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

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

Example 2:

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

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

- **foreColor** sprite property; **stageColor** property

● BACKSPACE

Syntax: BACKSPACE

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

Example:

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

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

- **beep**

Syntax: beep [*numberOfTimes*]

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

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

Example 1:

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

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

Example 2:

This handler causes up to three beeps whenever a key is pressed:

```
on keyDown  
    beep random(3)  
end
```

• beepOn

Syntax: the beepOn

This property determines whether the computer beeps when the user clicks outside an active sprite--a sprite that has a script associated with it. If the `beepOn` property is set to `TRUE`, clicking outside active sprites results in a beep.

The `beepOn` property can be tested and set. The default value is `FALSE`.

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

Example 1:

This statement sets the `beepOn` property to `TRUE`:

```
set the beepOn to TRUE
```

Example 2:

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

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

- **before**

-

put...before command

- **beginRecording**

Syntax: beginRecording

This keyword starts a score update session. Only one update session in a movie can be active at a time.

Example:

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

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

- **endRecording, updateFrame, generating score**

● blend of sprite

Syntax: the blend of sprite *whichSprite*

Using this sprite property, you can set or determine the puppet sprite's blend value. Blend values can be from 0 to 100, which correspond to the blend values in **Sprite Properties** dialog box.

Example 1:

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

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

Example 2:

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

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


- **border of member**

Syntax: the border of member *whichCastmember*

This field cast member property indicates the width, in pixels, of the border around the specified field cast member.

Example:

This statement makes the border around the field cast member Title ten pixels wide:

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

- **bottom of sprite**

Syntax: the bottom of sprite *whichSprite*

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

The `bottom of sprite` property can be tested, but not set directly. Use the `spriteBox` command to set the bottom vertical coordinate of a sprite.

Example:

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

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

Note: Sprite coordinates are measured in numbers of pixels, starting with (0,0) at the upper left corner of the stage. Stage coordinates are measured in numbers of pixels, starting with (0,0) at the upper left corner of the monitor.

- **height**, **left**, **locH**, **locV**, **right**, **top**, and **width** sprite properties; **spriteBox** command

• **boxDropShadow of member**

Syntax: the boxDropShadow of member *whichCastMember*

This field cast member property determines the size, in pixels, of the drop shadow for the box of the field cast member specified by *whichCastmember*.

Example:

This statement makes the drop shadow of field cast member Title ten pixels wide:

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

- **boxType of member**

Syntax: the boxType of member *whichCastmember*

This field cast member property determines how the type of text box for the cast member is specified by *whichCastmember*. The possible values are #adjust, #scroll, #fixed, and #limit.

Example:

This statement makes the box for field cast member Editorial a scrolling field:

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

- **buttonStyle**

Syntax: the `buttonStyle`

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

The `buttonStyle` property can have these values:

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

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

Example 1:

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

```
set the buttonStyle to 1
```

Example 2:

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

```
put the buttonStyle into buttonStyleValue
```

- **checkBoxAccess** and **checkBoxType** properties

● **buttonType**

Syntax: the buttonType of member *whichCastmember*

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

Example:

This statement makes the button cast member Editorial a checkbox:

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

- ◆ **cancelIdleLoad**

Syntax: `cancelIdleLoad loadTag`

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

Example:

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

```
cancelIdleLoad 20
```

- ◆ **idleLoadTag**

- **case**

Syntax: `case case expression of
expression1 : Statement(s)
expression2 :
multipleStatements
.
.
.
expression3, expression4 :
Statement(s)
{otherwise statement(s)}
end case`

This keyword starts a multiple branching logic structure that is more efficient than repeated use of if-then statements.

Lingo compares the value in *case expression* to the expressions in lines beneath it. The comparison starts at the beginning and continues through each line in order until Lingo encounters an expression that matches *case expression*. When a matching expression is found, Lingo executes the corresponding statement or statements that follow the colon after the matching expression. When only one statement follows the matching expression, the matching expression and its corresponding statement appear on the same line. Multiple statements are on indented lines immediately below the matching expression.

When there is more than one possible match that causes Lingo to execute the same statements, the expressions must be separated by commas. (The line containing *expression3* and *expression4* is an example of such a situation.) If the optional *otherwise* statement is included at the end of the case structure, the statement following *otherwise* is executed if there are no matches.

Example:

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
```


● castLib

Syntax: castLib *whichCast*

This keyword indicates that the cast specified in *whichCast* is a cast.

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

Examples:

The following statement displays the number of the cast Buttons in the message window:

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

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

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

● **castLibNum of sprite**

Syntax: the castLibNum of sprite *whichSprite*

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

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

Example:

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

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

- **center of member**

Syntax: the center of member *whichCastmember*

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

- When the `crop of member` is FALSE, the `center of member` has no effect.
- When the `crop of member` is TRUE and the `center of member` is TRUE, cropping occurs around the center of the digital video cast member.
- When the `crop of member` is TRUE and the `center of member` is FALSE, cropping starts at the top left corner of the sprite that refers to the digital video cast member.

Example:

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

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

- **crop of member** digital video cast member property

- **centerStage**

Syntax: the centerStage

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

- If the centerStage is TRUE, the stage is centered.
- If the centerStage is FALSE, the stage is not centered.

This property is useful for checking stage location before a movie plays from a projector. Place handlers that use this property in the preceding movie before using the `go to movie` command.

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

Example 1:

This statement sends the movie to a specific frame if the stage is not centered:

```
if the centerStage = FALSE then -  
  go to frame "off center"
```

Example 2:

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

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

- **fixStageSize** property

- **changeArea of member**

Syntax: the changeArea of member *whichCastMember*

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

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

Example:

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

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

- **channelCount of member**

Syntax: the channelCount of member *whichCastmember*

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

Example:

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

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

- **char...of**

Syntax: char *whichCharacter* of *chunkExpression*
 char *firstCharacter* to *lastCharacter* of *chunkExpression*

This chunk expression keyword identifies a character or a range of characters in a chunk expression. Chunk expressions are used to refer to any character, word, item, or line in any source of text such as field cast members and variables that hold strings.

- The expression *whichCharacter* identifies a specific character.
- The expressions *firstCharacter* and *lastCharacter* identify a range of characters.

The expressions must be integers that specify a character or range of characters in the chunk. Characters include letters, numbers, punctuation marks, spaces, and control characters like TAB and RETURN.

You can test and set the `char...of` keyword.

Example 1:

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

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

Example 2:

This statement displays the entire string \$9.00:

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

Example 3:

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

```
put "?????" into char 1 to 5 of word 2 of line 3 -  
of member "quiz"
```

- **mouseCast**, **mouseItem**, **mouseLine**, and **mouseWord** integer functions; **word...of**, **item...of**, and **line...of** chunk expression keywords; **number of chars in** chunk function; and **chars**, **length**, and **offset** functions

● charPosToLoc

Syntax: `charPosToLoc (member whichCastMember, nthCharacter)`

This function gives the point in the specified field cast member that is closest to the character specified by *nthCharacter*. This is useful for determining the location of individual characters.

Values for `charPosToLoc` are in pixels from the top left corner of the field cast member. The *nthChar* parameter is 1 for the first character in the field, 2 for the second character, and so on. The point is the point in the entire field cast member, not the part of the field cast member that appears on the stage.

Example:

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

```
put charPosToLoc(member "Headline", 50) into location
```


- **chars**

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

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

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

Example 1:

This statement identifies the sixth character in the word Macromedia:

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

Example 2:

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

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

Example 3:

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

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

- **char..of** chunk expression keyword; **length** and **offset** functions; **number of chars** **in** chunk function

- **charToNum**

Syntax: `charToNum(stringExpression)`

This function identifies the ASCII code number that corresponds to the first character of *stringExpression*.

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

Director treats upper- and lower-case letters the same if you compare them using the equals sign (=) operator. However, it treats them differently if you use the < or > operator. For example, the statement `put ("M" = "m")` gives the result 1 or TRUE. However the statement `put ("M" < "m")` or the statement `put ("M" > "m")` gives the result 0 or FALSE. This can create confusion when comparing characters. You can avoid problems by using `charToNum` to give the ASCII code for a character and then using the ASCII code to refer to the character.

Example 1:

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

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

Example 2:

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

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

- **numToChar** function

- **checkBoxAccess**

Syntax: the checkBoxAccess

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

0--Lets the user set checkboxes and radio buttons on and off. This is the default.

1--Lets the user set checkboxes and radio buttons on but not off.

2--Prevents the user from setting checkboxes and radio buttons at all; the buttons can only be set by scripts.

The `checkBoxAccess` property can be tested and set. The default value is 0.

Example 1:

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

```
set the checkBoxAccess to 1
```

Example 2:

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

```
put the checkBoxAccess into oldAccess
```

- hilite of member property; checkBoxType property

- **checkBoxType**

Syntax: the checkBoxType

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

0--Creates a standard checkbox that has "x" when the checkbox is selected. This is the default.

1-- Creates a checkbox that has a black rectangle when the checkbox is selected.

2-- Creates a checkbox that has a filled black rectangle when the checkbox is selected.

The `checkBoxType` property can be tested and set. The default value is 0.

Example 1:

This statement sets the `checkBoxType` property to 1, which creates a black rectangle in checkboxes when the user clicks them.

```
set the checkBoxType to 1
```

- hilite of member property; checkBoxAccess property

- **checkMark of menuItem**

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

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

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

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

The `checkMark of menuItem` property can be tested and set. The default value is FALSE.

Example:

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

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

- **installMenu** command; **enabled of menuItem**, **name of menuItem**, **number of menuItems**, and **script of menuItem** menu item properties; **name of menu** and **number of menus** menu item properties; **menu** keyword

- **chunkSize of member**

Syntax: the chunkSize of member *whichCastMember*

This transition cast member property, which determines the transition's chunk size, does the same as setting the smoothness slider in the Frame Properties: Transition dialog box. It can be tested and set. Values are the number of pixels in each chunk of the transition and can be any value from 1 to 128 pixels.

Example:

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

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

- **clearFrame**

Syntax: clearFrame

This command erases everything already in the current frame's sprite and effects channels. It works 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
      set the frameScript to 25
      updateFrame
    end repeat
  endRecording
end
```

- **Score generation**

• **clearGlobals**

Syntax: `clearGlobals`

This command sets all user-defined global variables to 0.

Example:

If you initialize a global variable with a string or value:

```
global foo  
  
put "Director" into foo
```

The global variable `foo` contains the string `Director` until another string or value is put into the global, or until the `clearGlobals` command is issued. This can be useful when initializing global variables, or when opening a new movie that requires a new set of global variables.

```
clearGlobals
```

When this command is issued, the global variable `foo` contains 0.

● clickLoc

Syntax: the clickLoc

This function identifies the last place on the screen where the mouse was clicked. The location is given as a point.

The value of clickLoc usually changes frequently. If you want to record a specific value of clickLoc, it is a good idea to assign it to a variable at the time the click occurs.

Example:

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

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

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

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

- **clickOn**

Syntax: the clickOn

This function returns the last active sprite clicked by the user. An active sprite is a sprite that has a sprite script associated with it.

When you want 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`.

The `clickOn` function has no effect in a repeat loop.

To detect a click on the stage, test whether the `clickOn` equals 0.

The value of `clickLoc` usually changes frequently. If you want to record a specific value of `clickLoc`, it is a good idea to assign it to a variable at the time the click occurs.

Example 1:

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

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

Example 2:

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

```
set the foreColor of sprite (the clickOn) to -  
random(255)-1
```

- **doubleClick, mouseDown, and mouseUp** functions

- **close window**

Syntax: `close window windowIdentifier`

This command closes the window specified by *windowName*.

- To specify a window by name, use the syntax `close window "name "`, where you replace name with the name of a window. Be sure to use the complete path name.
- To specify a window by its number in the `windowList`, use the syntax `close window number`, where you replace *number* with the window's number in the window list.

Lingo permits you to attempt to close a window that is already closed.

Example 1:

This statement closes the window named Panel:

```
close window "Panel"
```

Example 2:

This statement closes the window that is number 5 in the window list:

```
close window 5
```

- **open window** command; **windowList** function

- **closeResFile**

Syntax: closeResFile [*whichFile*]

On the Macintosh, this command closes the resource file specified by the string expression *whichFile*. When the resource file is in a different folder than the current movie, *whichFile* must specify a pathname. When no file is specified, all open resource files are closed. In Windows, this command performs no operation and generates no error message.

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

Example 1:

This statement closes all open resource files:

```
closeResFile
```

Example 2:

This statement closes the file Special Fonts if it is in the same folder as the movie or in a folder that is in Director's search path:

```
closeResFile "Special Fonts"
```

Example 3:

This statement closes the file Special Fonts in the folder Special Tools on the same disk as the movie. The disk is identified by the variable currentDrive:

```
closeResFile currentDrive & ¬  
":Special Tools:Special Fonts"
```

- **openResFile** and **showResFile** commands

- **closeXlib**

Syntax: closeXlib [*whichFile*]

This command closes the Xlibrary file specified by the string *whichFile* . If the Xlibrary is in another folder than the current movie, *whichFile* must specify a pathname. If no file is specified, all open Xlibraries are closed.

Xtras and XObjects are stored in Xlibrary files. Xlibrary files are resource files that contain XCOD (XObjects) resources. HyperCard XCMDs and XFCNs can also be stored in Xlibrary files.

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

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

Example 1:

This statement closes all open Xlibrary files:

```
closeXlib
```

Example 2:

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

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

Example 3:

This statement closes the Xlibrary Transporter Xtras in the folder New Xtras on the same disk as the movie. The disk is identified by the variable `currentDrive`:

```
closeXlib currentDrive & ~  
":New XObjects:Transporter XObjects"
```

- **openXlib** and **showXlib** commands

• **colorDepth**

Syntax: the colorDepth

This property determines the color depth of the computer's monitor.

- On the Macintosh, using this property lets you check the color depth of different monitors and change it when appropriate.
- In Windows, using this property lets you check the monitor's color depth, but not change it. (Changing color depth in Windows requires using the System Setup program or installing the proper video card driver.)

Possible values are the following:

1--Black-and-white

2--4 colors

4--16 colors

8--256 colors

16--32,768 colors

32--16,777,216 colors

When you assign a color depth higher than the monitor's color depth no color depth change occurs.

On computers with more than one monitor, the `colorDepth` property refers to the monitor that the stage is on. If the stage spans more than one monitor, the `colorDepth` indicates the greatest depth of those monitors; setting the `colorDepth` attempts to put all those monitors to the specified depth.

The `colorDepth` property can be tested and set. On the Macintosh, the default value is the value set in the Monitors control panel.

Example 1:

This statement makes playing the segment "Full color" dependent on whether the monitor color depth is set to 256 colors:

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

Example 2:

This statement uses the `colorQD` function to check whether the monitor can display color, and then sets the color depth to 256 colors if it is:

```
if the colorQD = TRUE then set the colorDepth to 8
```

- colorQD function; switchColorDepth property

- **colorQD**

Syntax: the colorQD

This function indicates whether the Color QuickDraw software is available on a Macintosh.

- The `colorQD` is TRUE (1) for a color-capable Macintosh. For any computer using Windows, the `colorQD` is always TRUE regardless of what is on the computer.
- The `colorQD` is FALSE (0) for a black-and-white-only Macintosh.

Note: A machine capable of displaying color may not have it switched on. This command is best used in conjunction with `colorDepth`.

Example 1:

This statement checks whether the Macintosh is color capable and plays the movie "Color Movie" if it is:

```
if the colorQD = TRUE then play movie "Color Movie"
```

Example 2:

This statement checks whether the Macintosh is color capable and sets the color depth to 256 colors if it is:

```
if the colorQD = TRUE then set the colorDepth to 8
```

- **colorDepth** and **switchColorDepth** properties

- **commandDown**

Syntax: the commandDown

This function determines whether the Command key is being pressed on the Macintosh or the Control key is being pressed in Windows.

- The `commandDown` function is TRUE when the Command key is being pressed on the Macintosh or the Control key is being pressed in Windows.
- The `commandDown` function is FALSE when the Command key is not being pressed on the Macintosh or the Control key is not being pressed in Windows.

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

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

Example:

These statements have Lingo pause a projector whenever the user presses Command-P on the Macintosh or Control-P in Windows. By setting the `keyDownScript` property to `doCommandKey`, the `on startMovie` handler makes the `doCommandKey` handler the first event handler executed when a key is pressed. The `doCommandKey` handler checks whether the Command (or Ctrl) and P keys are pressed at the same time and pauses the movie if they are.

```
on startMovie
    set the keyDownScript to "doCommandKey"
end startMovie

on doCommandKey
    if (the commandDown) and (the key = "p") then pause
end
```

- **controlDown, key, keyCode, optionDown, and shiftDown** functions

- **constrainH**

Syntax: `constrainH (whichSprite , integerExpression)`

This function evaluates *integerExpression*, and then gives a value that depends on the horizontal coordinates of the left and right edges of *whichSprite*.

- When the value is between the left and right coordinates, the value doesn't change.
- When the value is less than the left horizontal coordinate, the value is changed to the value of the left coordinate.
- When the value is greater than the right horizontal coordinate, the value is changed to the value of the right coordinate.

The `constrainH` and `constrainV` functions constrain only one axis each; the `constraint of sprite` property limits both. Note that this function does not change the sprite's properties.

Example 1:

These statements check the `constrainH` for sprite 1 when it has left and right coordinates of 40 and 60:

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

```
put constrainH(1, 55)
-- 55
```

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

Example 2:

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

```
set the locH of sprite 1 to constrainH(2, the mouseH)
```

- **constrainV** function; **constraint of sprite**, **left**, and **right** sprite properties

- **constraint of sprite**

Syntax: the constraint of sprite *whichSprite*

This sprite property determines the constraints on the position of the sprite specified by *whichSprite*. When the `constraint of sprite` property is turned on, the sprite specified by *whichSprite* is constrained to the bounding rectangle of another sprite.

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

To remove a constraint of sprite property, set it to 0:

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

The `constraint of sprite` property can be tested and set. The default value is 0.

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

Example 1:

This statement constrains sprite (i + 1) to the boundary of sprite 14.

```
set the constraint of sprite (i + 1) to 14
```

Example 2:

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

```
if the constraint of sprite 3 <> 0 then -  
  showConstraint
```

- **constrainH** and **constrainV** functions; **locH** and **locV** sprite properties

◆ **constrainV**

Syntax: `constrainV (whichSprite , integerExpression)`

This function evaluates *integerExpression*, and then gives a value that depends on the vertical coordinates of the top and bottom edges of the sprite specified by *whichSprite*.

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

Note that this function does not change the sprite properties.

Example 1:

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

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

put constrainV(1, 55)
-- 55

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

Example 2:

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

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

- ◆ **bottom of sprite**, **constraint of sprite**, and **top of sprite** sprite properties; **constrainH** function

- **contains**

Syntax: *stringExpression1* contains *stringExpression2*

This operator compares two strings.

- When *stringExpression1* contains *stringExpression2*, the condition is TRUE (1).
- When *stringExpression1* does not contain *stringExpression2*, the condition is FALSE (0).

The `contains` comparison operator has a precedence level of 1.

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

Example:

This statement determines whether a string from a field contains only numeric input before converting it using the `value()` function. You can use this handler to test it:

```
on isNumber aLetter
  put "1234567890." into digits
  if digits contains aLetter then
    return TRUE
  else
    return FALSE
  end if
end isNumber
```

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

- **offset** function; **starts** comparison operator

- **continue**

Syntax: `continue`

The `continue` command resumes playing the movie after a pause.

Example:

This statement has the movie resume playing when it is paused and the Return key is pressed:

```
set the keydownScript to "if the key = RETURN ↵  
  then continue"
```

This handler checks whether the movie is currently paused when the user releases the mouse button and has it continue if it is:

```
on mouseUp  
  if the pauseState = #paused then continue  
end
```

- **delay** and **pause** commands; **pauseState** function

- **controlDown**

Syntax: the controlDown

This function determines whether the Control key on a Macintosh or the Ctrl key on a PC is being pressed.

- The `controlDown` function is TRUE when the Control key is being pressed.
- The `controlDown` function is FALSE when the Control key is not being pressed.

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

Example:

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

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

- **charToNum**, **commandDown**, **key**, **keyCode**, **optionDown**, and **shiftDown** functions

- **controller of member**

Syntax: the controller of member *castName*

A digital video movie cast member can be made to show or hide its controller with this cast member property. Setting this property to 1 shows the controller; setting it to 0 hides it.

The `controller of member` property applies to QuickTime and QuickTime for Windows movies only. Setting `controller of member` for a Video for Windows movie performs no operation and generates no error message. Checking the `controller of member` for a Video for Windows movie always returns FALSE.

The digital video movie must be in `directToStage` playback mode in order to display the controller.

Example:

This statement has the QuickTime cast member Demo show its controller:

```
set the controller of member "Demo" to 1
```

- **directToStage of member** cast member property

● **copyToClipboard**

Syntax: `copyToClipboard member whichCastmember`

This command copies the specified cast member to the Clipboard. You can use this command to copy cast members between movies or applications. The cast window does not need to be the active window when you use the `copyToClipboard` command.

Example 1:

This statement copies the cast member named chair to the Clipboard:

```
copyToClipboard member "chair"
```

Example 2:

This statement copies cast member number 5 to the Clipboard:

```
copyToClipboard member 5
```

- **cos**

Syntax: `cos (angle)`

This function calculates the cosine of the specified angle. The angle must be expressed in radians.

Example:

The following statement calculates the cosine of $\pi ()/2$ and displays it in the message window:

```
put cos (pi ()/2)
```

● **count**

Syntax: `count (list)`

This function returns the number of entries in the specified list.

The `count` command works with linear and property lists.

Example:

This statement displays the number 3, the number of entries:

```
put count ( [10, 20, 30] )
```

```
-- 3
```

- **crop of member**

Syntax: the crop of member *whichCastmember*

This cast member property affects how the digital video cast member is displayed inside a sprite when the digital movie is larger than the sprite that it appears in. It can be tested and set.

- When the `crop of member` is FALSE the cast member is scaled--either stretched or shrunk--to fit inside the sprite rectangle.
- When the `crop of member` is TRUE, the cast member is not scaled. It is cropped to fit inside the sprite rectangle.

Example:

This statement instructs Lingo to crop any sprite that refers to the digital video cast member Interview:

```
set the crop of member "Interview" to TRUE
```

- **center of member** digital video cast member property

- **cursor**

Syntax: `cursor [castNumber , maskCastNumber]`

or

`cursor whichCursor`

This command changes the cast member that is used for a cursor. The `cursor` command stays in effect until you turn it off by setting the cursor to zero.

- Use the syntax `cursor [castNumber , maskCastNumber]` to specify the number of a cast member to use as a cursor and its optional mask. The hot spot of the cursor is the registration point of the cast member.

The cast member that you specify must be a 1-bit cast member. If the cast member is larger than 16 x 16 pixels, Director crops it to a 16 x 16 square, starting in the upper left corner of the image. The bitmap's registration point is the cursor's hot spot.

- Use the syntax `cursor whichCursor` to use the default cursors that are supplied by the system. The term *whichCursor* must be an integer that specifies the appearance of the cursor. The following values specify cursors:

- 0** -- no cursor set
- 1** -- arrow (pointer) cursor
- 1** -- I-beam cursor
- 2** -- crosshair cursor
- 3** -- crossbar cursor
- 4** -- watch cursor
- 200** -- blank cursor

To hide the cursor, set the cursor to 200 (a blank cursor).

During system events such as loading a file, the operating system may put up the watch cursor, and then change to the pointer cursor when returning control to the application. This overrides the `cursor` command settings from the previous movie. Therefore, in a presentation using a custom cursor for multiple movies, store any special cursor resource number as a global variable. Global Lingo variables stay in memory between movies. This allows you to use the `cursor` command at the beginning of any new movie that is loaded.

Cursor commands can be interrupted by an Xtra or other external agent. If the cursor is set to a value in Director and something else takes control of the cursor, resetting the cursor to the original value has no effect because Director doesn't perceive that the cursor changed. You can work around this by explicitly setting the cursor to some third value and then resetting it to the original value.

Notes:

- In Windows, a cursor can't be a resource; it must be a cast member. If a cursor isn't available in Windows because it hasn't been converted from a resource to a cast member, Lingo uses the standard arrow cursor instead. It is recommended that you don't make custom cursors resources when you create movies that you intend to play on both the Macintosh and PC.

- Be sure not to confuse `cursor 1` with `cursor [1]`. The first selects the I-beam from the system cursor set; the second uses castmember 1 as the custom cursor.

Example:

On the Macintosh, this statement changes the cursor to a watch cursor whenever the value in the variable named status equals 1:

```
if status = 1 then cursor 4
```

This handler checks whether the cast member assigned to the variable is a 1-bit cast member and then uses it as the cursor if it is:

```
on myCursor someMember
  if the depth of member someMember = 1
    then cursor[ someMember ]
  else
    beep
  end if
end
```

- ◆ **cursor of sprite** property; **openResFile** command; **rollOver** function

• cursor of sprite

Syntax: the cursor of sprite *whichSprite* to [*castNumber*, *maskCastNumber*]
 the cursor of sprite *whichSprite* to *whichCursor*

This sprite property determines the cursor resource that is used when the pointer is over the sprite specified by the integer expression *whichSprite*. The cursor property stays in effect until you turn it off by setting the cursor to zero.

When you set the `cursor of sprite` in a given frame, Director keeps track of the sprite rectangle to know whether to alter the cursor. This rectangle persists when the movie enters another frame unless you set the cursor of sprite property for that channel to 0.

Note: In Windows, a cursor can't be a resource; it must be a cast member. If a cursor isn't available in Windows because it hasn't been converted from a resource to a cast member, Lingo uses the standard arrow cursor instead. It is recommended that you don't make custom cursors resources when you create movies that you intend to play on both the Macintosh and PC.

The `cursor of sprite` property is an integer that specifies the resource ID number of the cursor. The following cursors are always available:

0	--no cursor set; uses system default
-1	--arrow (pointer) cursor
1	--I-beam cursor
2	--crosshair cursor
3	--crossbar cursor
4	--watch cursor
200	--blank cursor

To hide the cursor, set the `cursor` to 200 (a blank cursor resource).

To use custom cursors, set the `cursor of sprite` property to an external resource that contains the custom cursor.

The `cursor of sprite` property is useful for changing the cursor when the mouse pointer is over specific regions of the screen. You can use this to indicate regions where certain actions are possible when the user clicks.

When the cursor is over the location of a sprite that has been removed, the rollover still occurs. Avoid this problem by not doing rollovers over these locations or by relocating the sprite up above the menu bar before deleting it.

On the Macintosh, you can use a numbered cursor resource in the current open movie file as the cursor by replacing *whichCursor* with the number of the cursor resource.

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

Example:

This statement changes the cursor to a watch cursor whenever the value in the variable named `status` equals 1:

```
if status = 1 then cursor 4
```

- ◆ **cursor** and **openResFile** commands

- **date**

Syntax: the abbr date
 the abbrev date
 the abbreviated date
 the date
 the long date
 the short date

This function gives the current date in the system clock in one of three formats: abbreviated, long, or short. If no format is specified, the default is short. The abbreviated format can also be referred to as `abbrev` and `abbr`.

The format that Director uses parts of the date vary depending on how the date is formatted on the computer. As a result, you have no reliable way to use the date in calculations after you distribute the movie.

- 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 & Time control panel.

Example 1:

This statement gives the abbreviated date:

```
put the abbreviated date
-- "Sat, Sep 7, 1991"
```

Example 2:

This statement gives the long date:

```
put the long date
-- "Saturday, September 7, 1991"
```

Example 3:

This statement gives the short date:

```
put the short date
-- "9/7/91"
```

Example 4:

This statement tests whether the current date is January 1 by checking whether the first four characters of the date are 1/1. If it is January 1, the alert "Happy New Year!"

appears:

```
if char 1 to 4 of the date = "1/1/" ↵  
    then alert "Happy New Year!"
```

Note: The three date formats vary, depending on the country for which your System file was designed. These examples are for the United States.):

- time function

● delay

Syntax: `delay numberOfTicks`

This command halts the movie for a given amount of time. The integer expression *numberOfTicks* specifies the number of ticks to wait. (There are 60 ticks per second.) The only mouse and keyboard interactivity possible during this time is to stop the movie by pressing Control+Alt+Period.

The `delay` command works only when the playback head is moving. Place scripts using the `delay` command in either an `on enterFrame` or `on exitFrame` handler.

To mimic the behavior of a halt in a handler when the playback head is not moving, use the `startTimer` command or assign the current value of the `timer` to a variable and wait for an amount of time to pass before exiting the frame. (An example is in the following set.)

Because it increases the time of individual frames, the `delay` command is useful for controlling the playback rate of a sequence of frames.

Example 1:

This handler delays the movie for 2 seconds when the playback head exits the current frame:

```
on exitFrame
    delay 2 * 60
end exitFrame
```

Example 2:

This handler, which could be placed in a frame script, delays the movie a random number of ticks:

```
on keyDown
    if the key = RETURN then delay random(180)
end keyDown
```

Example 3:

The first of these handlers sets 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 gTimer < ( 10 * 60 ) then
    go to the frame
  end if
end
```

Note: The `delay` command does not function when the playback head is not moving.

- **startTimer** command; **timer** property

- **delete**

Syntax: delete *chunkExpression*

This command deletes the specified chunk expression (character, word, item, or line) in any string container. Sources of strings include field cast members and variables that hold strings.

Example 1:

This statement deletes the first word of line 3 in the field cast member Address:

```
delete word 1 of line 3 of member "Address"
```

Example 2:

This statement deletes the first character of the string in the variable `bidAmount`:

```
if char 1 of bidAmount = "$" then delete char 1 ↵  
of bidAmount
```

- **char...of, field, item...of, line...of, word...of**, chunk expression keywords; **hilite** text property, **Understanding text and fields**

- **deleteAt**

Syntax: `deleteAt list , number`

This command deletes the item in the position specified by number from the list specified by *list*. The value *number* is the item's position in the order of the list.

If you try to delete an object that isn't in the list, Director gives an alert. You can avoid this by first checking whether the item is in the list.

The `deleteAt` command works with linear and property lists.

Example:

This statement deletes the second item from the list named designers, which contains ["gee", "kayne", "ohashi"]:

```
set designers = ["gee", "kayne", "ohashi"]
deleteAt designers, 2
```

The result is the list ["gee", "ohashi"].

This handler checks whether an object is in a list before attempting to delete it:

```
on myDeleteAt theList, theIndex
  if count( theList ) < theIndex then
    beep
  else
    deleteAt theList, theIndex
  end if
end
```

- **addAt command**

• deleteFrame

Syntax: deleteFrame

This command deletes the current frame. After the current frame is deleted, the next frame becomes the new current frame.

The `deleteFrame` command works during a score generation session only.

Example:

The following handler checks whether the sprite in channel10 of the current frame has gone past the right edge of a 640 x 480 stage and deletes the frame if it has:

```
on testSprite
  beginRecording
    if the locH of sprite 10 > 640 -
      then deleteFrame
    endRecording
end
```

• deleteOne

Syntax: `deleteOne list, value`

This command deletes a value from a linear or property list. If the value appears in the list more than once, `deleteOne` deletes the first occurrence only.

When the list is a property list, the property associated with the deleted value is also removed from the list.

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

Example:

The first statement creates a list consisting of the days Tuesday, Wednesday, and Friday. The second statement deletes the name Wednesday from the list.

```
set days = ["Tuesday", "Wednesday", "Friday"]
deleteOne days, "Wednesday"
put days
-- ["Tuesday", "Friday"]
```

The `put days` statement has the message window display the result, which is:
["Tuesday", "Friday"].

- **deleteProp**

Syntax: `deleteProp list, property`

This command deletes the item that has the specified property from the specified list. For linear lists, this is the same as the `deleteAt` command. When there are more than one of the same property, only the first property in the list is deleted.

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

Example:

This statement deletes the property `color` from the list `[#height:100, #width: 200, #color: 34, #ink: 15]`, which is called `spriteAttributes`:

```
deleteProp spriteAttributes, #color
```

The result is the list `[#height:100, #width: 200, #ink: 15]`.

- **deleteAt** command

- **depth of member**

Syntax: the depth of member *whichCastmember*

This cast member property gives the color depth of the bitmap cast member specified by *whichCastmember*. Black and white is 1-bit color depth; 256 colors is 8-bit color depth; thousands of colors is 16-bit color depth; and millions of colors is 32-bit color depth.

This property can be tested, but not set from Lingo.

Example:

This statement determines the color depth of the cast member Shrine:

```
put the depth of member "Shrine"
```

• deskTopRectList

Syntax: the deskTopRectList

This system property indicates the size of the computer's monitors and their position in the desktop. It is useful for checking whether objects such as windows, sprites, and pop-up menus appear entirely on one screen.

The result is a list of standard rect coordinates, where each rect is the boundary of a monitor. The coordinates for each monitor are relative to the upper left corner of monitor 1, which has the value (0,0). The first set of rect coordinates is the size of the first monitor. If a second monitor is present, there is a second set of coordinates that show 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 x 768 pixels and the second monitor is 640 x 480 pixels.

This handler tells how many monitors are in the current system:

```
on countMonitors
  return count( deskTopRectList )
end
```

- **digitalVideo**

- center of member
controller of member
crop of member
directToStage of member
duration of member
frameRate of member
loop of member
movieRate of sprite
movieTime of sprite
pausedAtStart
preload of member
sound of member
trackStartTime(member)
trackStopTime(member)
video of member
volume of member

● **digitalVideoTimeScale**

Syntax: `the digitalVideoTimeScale`

This system property determines the time units that the system uses to measure time for digital video cast members. The value is in units per second. When `the digitalVideoTimeScale` is set to 0, Director uses the time scale of the movie that is currently playing.

Setting the time unit that measures digital video lets you precisely access tracks by making sure that the system's time unit for video is a multiple of the digital video's time unit.

Example:

This statement sets the time units that the system uses to measure digital video to 600:

```
set the digitalVideoTimeScale to 600
```

● **digitalVideoType of member**

Syntax: the digitalVideoType of member *whichCastmember*

This digital video cast member property indicates the format of the specified digital video. This property can be tested but not set. Possible values are *#quickTime* or *#videoForWindows*.

Example:

The following statement tests whether the cast member Today's Events is a QuickTime or AVI digital video and displays the result in the message window:

```
put the digitalVideoType of member "Today's Events"
```

- **directToStage of member**

Syntax: the `directToStage` of member *castName*

This property determines whether a digital video cast member plays in front of all other layers on the stage.

- When this property is set to 1, a digital video movie plays in front of all other layers.
- When this property is set to 0, a digital video movie cast member can appear in any layer of the stage's animation planes. (In Windows, the `directToStage` of member property is always TRUE. Setting the `directToStage` of member to FALSE has no effect in Windows.)

No cast members appear in front of a `directToStage` digital video movie. Also, ink effects do not affect the appearance of a `directToStage` digital video movie. Using this property may improve the playback performance of a digital video movie cast member.

Example:

This statement makes the QuickTime movie "The Residents" always play in the top layer of the stage:

```
set the directToStage of member "The Residents" to 1
```

• do

Syntax: do *stringExpression*

This command evaluates *stringExpression* and executes the result as a Lingo statement. This command is useful for evaluating expressions that the user has typed, and for executing commands stored in string variables, fields, arrays, and files.

Using uninitialized local variables within a `do` command creates a compile error. Initialize any local variables in advance.

Note: This command does not allow global variables to be declared, they must be declared in advance.

Example:

This statement performs the statement contained within quotes:

```
do "beep 2"
```

```
do getAt(commandList, 3)
```


• **dontPassEvent**

Syntax: `dontPassEvent`

This command prevents Lingo from passing an event message to subsequent locations in the message hierarchy.

The `dontPassEvent` command applies only to the current event being handled. It does not affect future events.

The `dontPassEvent` command applies only within primary event handlers or handlers that they call. It has no effect elsewhere.

Example 1:

This handler has the computer beep when the Tab or Enter key is pressed and keeps the message from passing on to subsequent locations in the message hierarchy:

```
on myKey
    if the key = TAB or the key = ENTER then beep
    dontPassEvent
end myKey
```

Example 2:

This statement makes `myKey` the primary event handler:

```
set the keyDownScript to "myKey"
```

When these are in effect at the same time, pressing the Tab or Enter key any time the movie is playing has the computer beep but not pass the `keyDown` message on to anywhere else in the movie.

- **doubleClick**

Syntax: the doubleClick

This function determines whether the last two mouse clicks were considered a double-click.

- The doubleClick function is TRUE if the last two mouse clicks were a double-click.
- The doubleClick function is FALSE if the last two mouse clicks were not a double-click.

Example:

This statement sends the playback head to the frame Enter Bid when the user double-clicks the mouse button.

```
if the doubleClick then go to frame "Enter Bid"
```

- **clickOn, mouseDown, mouseUp** functions

● drawRect of window

Syntax: the drawRect of window *windowName*

This window property identifies the rectangular coordinates of the section of the movie that appears in the movie's window. The coordinates are given as a rect, with entries in the order left, top, right, and bottom.

This can be useful for scaling or panning movies.

The drawRect of window property can be tested or set.

Example 1:

This statement displays the current coordinates of the movie window called Control Panel.

```
put the drawRect of window "Control Panel"  
-- rect(10, 20, 200, 300).
```

Example 2:

This statement sets the rect of the movie to the values of the rect `movieRectangle`. The portion of the movie within the rect is the part of the movie that appears in the window:

```
set the drawRect of window "Control Panel" to  
movieRectangle
```

● **dropShadow of member**

Syntax: the dropShadow of member *whichCastMember*

This field cast member property determines the size of the drop shadow for the characters in a field cast member. Possible values are a range of pixels.

Example:

This statement sets the drop shadow of the field cast member Comment to five pixels:

```
set the dropShadow of member "Comment" to 5
```

• **duplicate(list)**

Syntax: `duplicate(oldList)`

or

`set <x> = duplicate(oldList)`

This function returns a copy of a list. It is useful for saving the current content of a list for later use. Nested lists (list items that are themselves lists) are copied as lists, with all their content duplicated.

When you assign a list to a variable, the variable contains a reference to the list, not the list itself. You can make an independent copy of the list by using the following structure:

```
set newList = value(string(oldList))
```

Example:

This statement makes a copy of the list `CustomersToday` and assigns it to the variable `CustomerRecord`:

```
put duplicate(CustomersToday) into CustomerRecord
```

• duplicate member

Syntax: `duplicate member original [, new]`

This command makes a copy of the cast member specified by *original*. The optional *new* parameter specifies a specific cast window location for the duplicate cast member. If the *new* parameter isn't included, the duplicate cast member is placed in the first open cast window position.

Example 1:

This statement makes a copy of cast member Desk and places it in the first empty cast window position.

```
duplicate member "Desk"
```

Example 2:

This statement makes a copy of cast member Desk and places it in cast window position 125.

```
duplicate member "Desk", member 125
```

• duplicateFrame

Syntax: duplicateFrame

This command duplicates the current frame and its content. The duplicate frame is inserted after the current frame and then becomes the current frame. This command can be used during score generation only.

The `duplicateFrame` command does the same thing 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
  set the memberNum of sprite 20 to -
    the number of member "Ball" of castLib "Toys"
  repeat with i = 0 to numberOfFrames
    duplicateFrame
  end repeat
endRecording
end
```

- **duration of member**

Syntax: the duration of member *whichCastmember*

This cast member property determines the duration of *whichCastmember*.

- When *whichCastmember* is a digital video cast member, the property indicates the digital video's duration. The value is in ticks (60ths of a second).
- When *whichCastmember* is a transition cast member, the property indicates the transition's duration. The value for a transition is in milliseconds. During playback, this setting has the same effect as the Duration setting in the **Frame: Transition** dialog box.

Example 1:

This statement sets the variable `howLong` to the duration of the QuickTime cast member Demo:

```
put the duration of member "Demo" into howLong
```

Example 2:

This statement sets the duration of the transition cast member Fog to 1 second:

```
set the duration of member "Fog" = 60
```


- **editable of member**

Syntax: the editable of member *whichCastmember*

This field cast member property determines whether the specified field cast member is editable on stage.

- When the `editable of member` is TRUE, the specified field cast member is editable.
- When the `editable of member` is FALSE, the specified field cast member isn't editable.

Example:

This statement makes the field cast member Answer editable:

```
set the editable of member "Answer" = TRUE
```

- **editable of sprite**

Syntax: the editable of sprite *whichSprite*

This sprite property indicates whether a field sprite is editable.

- When the field can be edited by the user, the `editable of sprite` is TRUE.
- When the field cannot be edited by the user, the `editable of sprite` is FALSE.

To use Lingo to make a field sprite editable, the sprite must first be a puppet.

The `editable of sprite` property lets you change whether a field can be edited as the movie plays. This lets you turn editable on and off depending on current conditions in the movie.

You can also make a field cast member editable by using the Editable option in the **Field Cast Member Properties** dialog box.

You can make a field sprite editable by using the Editable option in the score.

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

For more information about handling several editable fields in a movie, see *Learning Lingo*.

Example 1:

This handler first makes the field sprite a puppet and then makes it editable:

```
on myNotes
  puppetSprite 5, TRUE
  set the editable of sprite 5 to TRUE
end
```

Example 2:

This statement checks whether a field sprite is editable and displays a message if it is:

```
if the editable of sprite 13 = TRUE →
  then set the text of member "Notice" to "Please →
  enter your answer below."
```

● EMPTY

Syntax: EMPTY

This character constant represents the empty string, "", a string with no characters.

You can scroll to a specific line in a scrolling field by inserting `EMPTY` before the line.

Example 1:

This statement erases all characters in the field cast member Notice by setting the field to `EMPTY`:

```
set the text of member "Notice" to EMPTY
```

Example 2:

This statement does the same as the previous statement but uses a different form:

```
put EMPTY into field "Notice"
```

- **emulateMultiButtonMouse**

Syntax: the emulateMultiButtonMouse

This system property determines whether clicking the right mouse button on a Windows computer corresponds to clicking the mouse button with the Control key pressed on a Macintosh. Since the mouse buttons for Windows and Macintosh computers are different, this property is very useful for providing consistent mouse button responses for cross-platform movies.

- When the `emulateMultiButtonMouse` is TRUE, the movie treats clicking the right mouse button on a Windows computer the same as clicking the mouse button while the Control key is pressed on the Macintosh.
- When the `emulateMultiButtonMouse` is FALSE, the movie doesn't treat clicking the right mouse button on a Windows computer the same as clicking the mouse button while the Control key is pressed on the Macintosh.

Example:

The following statement checks whether the movie is playing on a PC and sets the `emulateMultiButtonMouse` to TRUE if it is:

```
if the machineType = 256 then set the emulateMultiButtonMouse to TRUE
```

- **the keyPressed, the rightMouseDown, the rightMouseUp**

- **enabled of menuItem**

Syntax: the enabled of menuItem *whichItem* of menu *whichMenu*

This menu item property determines whether the menu item specified by *whichItem* is displayed in plain text or dimmed, and whether it is selectable. The term *whichMenu* specifies the menu that contains the menu item.

- If the enabled of menuItem is TRUE, the menu item appears in plain text and is selectable.
- If the enabled of menuItem is FALSE, the menu item appears dimmed and is not selectable.

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

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

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
  put the number of menuItems of menu theMenu into n
  repeat with i = 1 to n
    set the enabled of menuItem i of menu theMenu -
      to vSetting
  end repeat
end ableMenu
```

- **name of menu, number of menus, checkMark of menuItem, script of menuItem,** and **number of menuItems** properties

- **end**

This keyword marks the end of handlers, methods, and multi-line control structures.

- if...then, method, on, repeat while, and repeat with keywords; on mouseDown, on mouseUp, on keyDown, on startMovie, on stepMovie, on stopMovie, and on idle event handlers.

- **end case**

Syntax: end case

This keyword ends a case statement.

Example:

This handler uses the `end case` keyword to end the case statement:

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

- **case**

- **end repeat**

See repeat while
repeat with
repeat with...in list
repeat with...down to

● **endRecording**

Syntax: endRecording

This keyword ends a score update session.

You can resume puppeting after the `endRecording` keyword is issued.

Example:

When used in the following handler, the `endRecording` keyword ends the score generation session:

```
on animBall numberOfFrames
  beginRecording
    set the memberNum of sprite 20 to -
    the number of member "Ball"
    set horizontal = 0
    set vertical = 300
    repeat with i = 0 to numberOfFrames
      set the locH of sprite 20 to horizontal
      set the locV of sprite 20 to vertical
      set horizontal = horizontal + 3
      set vertical = vertical + 2
    updateFrame
  end repeat
endRecording
end
```

● **ENTER**

Syntax: ENTER

This character constant represents the Enter key.

- This key is marked "enter" on the Macintosh keyboard.
- Although PC keyboards also label the key that enters a carriage return as Enter, the element `ENTER` only refers to the Enter key on the number pad.

Example:

This statement checks whether the Enter key is pressed and sends the playback head to the frame `addSum` if it is:

```
on keyDown
  if the key = ENTER then go to frame "addSum"
end
```

- **enterFrame**

See [on enterFrame](#) event handler.

● erase member

Syntax: erase member *whichCastmember*

This command deletes the specified cast member and leaves its slot in the cast window empty.

Example 1:

This statement deletes the cast member named *Gear* in the cast *Hardware*:

```
erase member "Gear" of castLib "Hardware"
```

Example 2:

This handler deletes cast members *start* through *finish*:

```
on deleteMember start, finish
  repeat with i = start to finish
    erase member i
  end repeat
end on deleteMember
```

- **exit**

Syntax: exit

This keyword has Lingo leave a handler, and return to the place from where the handler was called. When the handler is nested within another handler, Lingo returns to the main handler.

Example:

The first statement of this script checks whether the monitor is set to black and white, and exits if it is:

```
on setColors
  if the colorDepth = 1 then exit
  set the foreColor of sprite 1 to 35
end setColors
```

- **abort** command; **on** and **method** keywords

- **exit repeat**

Syntax: `exit repeat`

This keyword has Lingo leave a repeat loop and go to the statement following the `end repeat` statement, but remain within the current handler or method.

The `exit repeat` keyword is useful for breaking out of a repeat loop when a specified condition--such as two values being equal or a variable being a certain value--exists.

Example:

This handler looks 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 has Lingo leave the repeat loop and go to the statement `return i`:

```
on findVowel testString
  repeat with i = 1 to the number of chars -
    in testString
      if "aeiou" contains -
        char I of testString then exit repeat
    end repeat
  return i
end findVowel
```

- **repeat while** and **repeat with** keywords

● **exitLock**

Syntax: the exitLock

This property determines whether the user can quit to the Desktop from projectors.

- When the `exitLock` is `FALSE`, the user can quit to the desktop by pressing `Control+period`, `Control+Q` or `Control+W`.
- When the `exitLock` is `TRUE`, the user cannot quit to the desktop by pressing `Control+period` or `Control+w`.

The `exitLock` property can be tested and set. The default value is `FALSE`.

Example 1:

This statement sets the `exitLock` property to `TRUE`:

```
set the exitLock to TRUE
```

Example 2:

This handler checks whether `Control+period` or `Control+Q` was pressed and whether the `exitLock` is set so that the user cannot exit to the Desktop. When this is the case, the playback head goes to the frame "quit sequence," which could provide an alternative way to exit the movie:

```
on checkExit
  if the commandDown and ¬
    (the key = "." or the key = "q") and ¬
    the exitLock = TRUE then go to frame "quit sequence"
end checkExit
```

- **exp**

Syntax: `exp(integer)`

This function calculates e, the natural logarithm base, to the power specified by *integer*.

Example:

The following statement calculates the value of e to the exponent 5:

```
put exp(5)
-- 148.4132
```


- **FALSE**

Syntax: `FALSE`

This logical constant applies to an expression that is logically FALSE, such as `2 > 3`. When treated as a number value, FALSE has the numerical value of 0.

This is functionally equivalent to the `pauseState` being FALSE.

Example:

This statement turns off the `soundEnabled` property by setting it to FALSE:

```
set the soundEnabled to FALSE
```

- **TRUE** logical constant, **NOT** keyword

- **field**

Syntax: field *whichField*

This keyword refers to a field cast member.

The field cast member is specified by *whichField*.

- When *whichField* is a string, it is used as the cast member name.
- When *whichField* is an integer, it is used as the cast member number.

Character strings can be read from or put into the field. You can also use chunk expressions with fields.

Example:

This statement puts the characters 5 through 10 of the field name entry into the variable myKeyword:

```
put char 5 to 10 of member "entry" into myKeyword
```

This statement checks whether the user entered the word "desk" and goes to the frame "deskBid" if he or she did:

```
if field "bid" contains "desk" then go to "deskBid"
```

- **char...of, item...of, line...of, and word...of** chunk expression keywords

● **fileName of castLib**

Syntax: the fileName of castLib *whichCast*

This property gives the filename of the specified cast.

- When the cast is external, the fileName of castLib gives the cast's full pathname and filename.
- When the cast is internal, the fileName of castLib gives the name of the movie.

This property can be tested and set.

Example:

This statement displays the pathname and filename of the external cast Buttons in the message window:

```
put the fileName of castLib "Buttons"
```

This statement sets the filename of the external cast Buttons to Content.Cst:

```
set the fileName of castLib "Buttons" to →  
the pathName & "Content.Cst"
```

The movie would then use the external cast file Content.Cst as the cast Buttons.

• **fileName of member**

Syntax: the `fileName` of member *whichCastmember*

This cast member property refers to the name of the file assigned to the linked cast member specified by *cast member*. This is useful for switching which external linked file is assigned to a cast member while the movie plays, similar to the way you can switch cast members. When the linked file is in a different folder than the movie, you must include the file's pathname.

The `fileName of member` property can be tested and set. After the filename is set, Director uses that file the next time the cast member is used.

Example:

This statement makes the QuickTime movie "ChairAnimation" the linked file assigned to cast member 40:

```
set the fileName of member 40 = "ChairAnimation"
```

• **fileName of window**

Syntax: the `fileName` of window *whichWindow*

This window property refers to the filename of the movie assigned to the window specified by *whichWindow*. When the linked file is in a different folder than the movie, you must include the file's pathname.

You assign a movie to a window by setting the `fileName` of window for the window to the movie's filename. This is required before you can play the movie in the window.

The `fileName` of window property can be tested and set.

Example:

This statement assigns the file named Control Panel to the window named Tool Box:

```
set the fileName of window "Tool Box" = ↵  
"Control Panel"
```

This statement displays the filename of the file assigned to the window named Navigator:

```
put the fileName of window "Navigator"
```

- **filled of member**

Syntax: the filled of member *whichCastmember*

This shape cast member property indicates whether the specified cast member is filled with a pattern.

- When the filled of member is TRUE, the shape is filled with a pattern.
- When the filled of member is FALSE, the shape isn't filled with a pattern.

Example:

The following statements make the shape cast member Target Area a filled shape and assigns it the pattern numbered 0, which is a solid color:

```
set the filled of member "Target Area" = TRUE  
set the pattern of member "Target Area" = 0
```

- **filled of member**

• findEmpty

Syntax: `findEmpty(member castmemberNumber)`

This function gives the next empty cast member position on or after the cast member specified by *whichCastmember*. This function works on the current cast.

Example:

This statement finds the first empty cast member on or after cast member 100.

```
put findEmpty(member 100)
```

- **findPos**

Syntax: `findPos(list, prop)`

This command identifies which position the property specified by *property* holds in the property list specified by *list*.

The `findPos` command works with sorted property lists only. Using `findPos` with a linear list or unsorted property list produces a script error.

The `findPos` command does the same thing as the `findPosNear` command, except that the result of the `findPos` command is void when the specified property is not in the list.

Example:

This statement identifies the position of the property `c` in the list `Answers`, which consists of `[#a:10, #b:12, #c:15, #d:22]`:

```
findPos(Answers, #c)
```

The result is 3, because `c` is the third property in the list.

- **findPosNear** command

- **findPosNear**

Syntax: `findPosNear(list , prop)`

This command identifies which position the property specified by *property* holds in the property list specified by *list*.

The `findPosNear` command works with sorted property lists only. Using `findPosNear` with a linear list or unsorted property list produces an error.

The `findPosNear` command does the same thing as the `findPos` command, except that when the specified property is not in the list, the `findPosNear` command identifies the position of the closest property in the list, based on the sort order. This would be useful in finding the closest name in a sorted directory of names.

Example:

This statement identifies the position of a property in the sorted list `Answers`, which consists of `[#Nile:2, #Pharaoh:4, #Raja:0]`:

```
findPosNear(Answers, #Ni)
```

The result is 1, because Ni is closest to Nile, the first property in the list.

- **findPos** command

● **finishIdleLoad**

Syntax: `finishIdleLoad loadTag`

This command completes loading for all the cast members that have the specified load tag.

Example:

This statement completes loading of all cast members that have the load tag 20:

```
finishIdleLoad 20
```

- **fixStageSize**

Syntax: the fixStageSize

This property determines whether the stage size remains the same when you load a new movie, regardless of the stage size saved with that movie. When the `fixStageSize` property is TRUE, the stage size remains the same when you load a new movie.

The `fixStageSize` property cannot change the stage size for a movie that is currently playing. This property is primarily used for movies played back with the player.

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

Example:

This statement determines if the `fixStageSize` property is turned on, and sends the playback head to a specified frame if it is.

```
if the fixStageSize = FALSE then ↵  
    go to frame "proper size"
```

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

```
set the fixStageSize to (not the fixStageSize)
```

- **centerStage** property

- **float**

Syntax: `float (expression)`

Converts an expression to a floating point number. The number of digits that follow the decimal point is set using the `floatPrecision` property.

Example:

This statement converts the integer 1 to floating-point 1.

```
put float(1)
-- 1.0
```

- **floatPrecision** property

- **floatP**

Syntax: `floatP(expression)`

This function indicates whether the value specified by *expression* is a floating point number.

- The `floatP` function is TRUE (1) if *expression* is a floating point number.
- The `floatP` function is FALSE (0) if *expression* is not a floating point number.

The "P" in `floatP` stands for "predicate."

Example:

This statement tests whether 3.0 is a floating point number. The message window displays the number 1, indicating that it is TRUE:

```
put floatP(3.0)
-- 1
```

This statement tests whether 3 is a floating point number. The message window displays the number 0, indicating that it is FALSE:

```
put floatP(3)
-- 0
```

- **float, integerP, objectP, stringP, and symbolP** functions

• floatPrecision

Syntax: the floatPrecision to *integer*

This system property rounds off the display of floating point numbers to the number of decimal places specified by *integer*. The maximum is 19 significant digits.

The `floatPrecision` property determines only the number of digits used to display floating point numbers. The number of digits used to perform calculations doesn't change.

The `floatPrecision` property can be tested and set. The default value is 4.

Example:

This statement rounds off the square root of 3.0 to three decimal places:

```
set the floatPrecision to 3
put sqrt(3.0) into x
put x

-- 1.732
```

This statement rounds off the square root of 3.0 to eight decimal places:

```
set the floatPrecision to 8
put x

-- 1.73205081
```

- **font of member**

Syntax: the font of member *whichCastmember*

This field property determines the typeface of the font used to display the specified field cast member. The parameter *whichCastmember* can be either a cast member name or number.

The `font of member` field property can be set, affecting every line in the field. When tested, it returns the height of the first line of the field.

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

Example:

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

```
put the font of member "Rokujo Speaks" into oldFont
```

- **text of member** property; **alignment of member**, **lineHeight of member**, **fontSize of member**, and **fontStyle of member** field properties

- **fontSize of member**

Syntax: the `fontSize` of member *whichCastmember*

This field property determines the size of the font used to display the specified field cast member. The parameter *whichCastmember* can be either a cast member name or number.

The `fontSize` field property can be tested and set.

Example:

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

```
put the fontSize of member "Rokujo Speaks" into ↵
   oldSize
```

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

- **text of member** property; **alignment of member**, **font of member**, and **lineHeight of member** field properties

- **fontStyle of member**

Syntax: the fontStyle of member *whichCastmember*

This field property determines the styles applied to the font used to display the specified field cast member.

The value of the property is a string of styles delimited by commas. Lingo uses a font that is a combination of the styles in the string. The available styles are plain, bold, italic, underline, shadow, outline, condense, and extend. In addition, you can use the word *normal* to remove all of the styles that are currently applied. The parameter *whichCastmember* can be either a cast member name or number.

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

The `fontStyle of member` field property can be tested and set.

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

Example:

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

```
put the fontStyle of member "Rokujo" into oldStyle
```

This statement sets the `fontStyle of member` setting for the field cast member Rokujo Speaks to bold italic:

```
set the fontStyle of member "Poem" to "bold, italic"
```

- **text of member** property; **alignment of member**, **font of member**, **lineHeight of member**, and **fontSize of member** field properties

- **foreColor of member**

Syntax: set the foreColor of member *castName* to *colorNumber*

This cast member property sets the foreground color of a field cast member.

Example:

This statement changes the color of the field in cast member 1 to the color in palette entry 250:

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

● **foreColor of sprite**

Syntax: the foreColor of sprite *whichSprite*

This sprite property determines the foreground color of the sprite specified by *whichSprite*. Setting the `foreColor` sprite property in a Lingo script is equivalent to choosing the foreground color from the tools window when the sprite is selected on the stage.

The foreground color applies only to 1-bit bitmap and shape cast members. It does not affect the display of a field or button cast member. An 8-bit, 16-bit, or 24-bit bitmap is affected, but generally not in a useful way.

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

Changing a sprite's foreground color during a `mouseDown` is a useful way to indicate when a sprite is clicked.

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. If you are changing several sprite properties--or several sprites--you only have to use one `updateStage` command at the end of all the changes.

The `foreColor of sprite` property can be tested and set, although in order to set it with Lingo the sprite must be a puppet.

Example:

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

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

The following statement makes 36 the number for the foreground color of a random sprite from sprite 11 to sprite 13.

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

• **forget window**

Syntax: `forget window whichWindow`

This command tells Lingo to close and delete the window specified by *window* when the window is no longer in use and no other variables refer to it.

Example:

This statement has Lingo delete the window Control Panel when the movie no longer uses the window:

```
forget window "Control Panel"
```

- **frame**

Syntax: the frame

This function gives the number of the current frame of the current movie.

Example:

This statement sends the playback head to the frame before the current frame:

```
go to (the frame - 1)
```

- **label** and **marker** functions, **go** command

● frameLabel

Syntax: `the frameLabel`

This frame property identifies the label assigned to the current frame. When the current frame has no label, the value of the `frameLabel` property is 0.

The `frameLabel` property can be tested at any time. It can also be set during a score generation session.

Example:

This statement checks the label of the current frame. In this case, the current `frameLabel` is `Start`:

```
put the frameLabel  
-- "Start"
```

- **framePalette**

Syntax: `the framePalette`

This frame property identifies the cast member number of the palette used in the current frame.

The `framePalette` 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
```

This statement makes cast member 45, which is a palette cast member, the palette for the current frame.

```
set the framePalette to 45
```

- **puppetPalette** command

- **frameRate of member**

Syntax: the frameRate of member *QTcastmember*

This digital video cast member property specifies the frame rate that the digital video movie specified by *QTcastmember* plays at. The possible values for the `frameRate of member` correspond to the radio buttons for selecting digital video playback options.

- When the `frameRate of member` is between 0 and 255, the digital video movie plays every frame at that frame rate. The `frameRate of member` property cannot be greater than 255.
- When the `frameRate of member` is set to -1, the digital video movie plays every frame at its normal rate.
- When the `frameRate of member` is set to -2, the digital video movie plays every frame as fast as possible.

Example:

This statement sets the frame rate of the QuickTime cast member Rotating Chair to 30 frames per second:

```
set the frameRate of member "Rotating Chair" to 30
```

This statement has the QuickTime cast member Rotating Chair play every frame as fast as possible:

```
set the frameRate of member "Rotating Chair" to -2
```

- **movieRate of sprite, movieTime of sprite** sprite properties

● frameScript

Syntax: the frameScript

This frame property identifies the cast member number of the frame script assigned to the current frame.

The `frameScript` property can be tested. During a score recording session, you can also assign a frame script to the current frame by setting the `frameScript` property. (This property could only be read in earlier versions of Director.)

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
```

This statement makes the script cast member "Button responses" the frame script for the current frame:

```
set the frameScript to member "Button responses"
```

● **frameSound1**

Syntax: the frameSound1

This frame property determines the number of the cast member assigned to the first sound channel in the current frame. This property can also be set during a score recording session.

Example:

As part of a score recording session, this statement assigns the sound cast member Jazz to the first sound channel:

```
set the frameSound1 to member "Jazz"
```

• **frameSound2**

Syntax: the frameSound2

This frame property determines the number of the cast member assigned to the second sound channel for the current frame. This property can also be set during a score recording session.

Example:

As part of a score recording session, this statement assigns the sound cast member Jazz to the second sound channel:

```
set the frameSound2 to member "Jazz"
```

• **framesToHMS**

Syntax: `framesToHMS (frames , tempo , dropFrame , fractionalSeconds)`

This function converts the specified number of frames to their equivalent length in hours, minutes, and seconds. This is useful for predicting the actual playtime of a movie or controlling a video playback device.

- The integer expression *frames* specifies the number of frames.
- The integer expression *tempo* specifies the tempo in frames per second.
- The *dropFrame* argument is a logical expression. Normally, this is FALSE. This argument is meaningful only if FPS is set to 30 frames per second. (Drop frame is a method of compensating for the color NTSC frame rate which is not exactly 30 frames per second.)
- The *fractionalSeconds* argument determines what happens to residual frames. When TRUE replaces *fractionalSeconds*, the residual frames are converted to the nearest hundredth of a second. When FALSE replaces *fractionalSeconds*, the residual frames are returned as an integer number of frames.

The resulting string uses the form: "sHH:MM:SS.FFD", where:

s--"-" if the time is less than zero, or space if the time is greater than or equal to zero

HH--hours

MM--minutes

SS--seconds

FF--fraction of a second if fractionalSeconds is TRUE, or frames if fractionalSeconds is FALSE

D--"d" if dropFrame is TRUE, or space if dropFrame is FALSE

Example:

This statement converts a 2710-frame, 30 frame-per-second movie. The dropFrame and fractionalSeconds arguments are both turned off:

```
put framesToHMS(2710, 30, FALSE, FALSE)
-- " 00:01:30.10 "
```

- **HMStoFrames** function

- ◆ **frameTempo**

Syntax: `the frameTempo`

This frame property indicates the tempo assigned to the current frame.

The `frameTempo` property can be tested. It can also be set during a score recording session. This property could only be tested in earlier versions of Director.

Example:

This statement checks the tempo used in the current frame. In this case, the tempo is 15 frames per second:

```
put the frameTempo
-- 15
```

- ◆ **puppetTempo** command

● **frameTransition**

Syntax: the frameTransition

This frame property gives the number of the transition cast member assigned to the current frame. During a score recording session, you can also set this property as a way to specify transitions.

Example:

When used in a score recording session, this statement makes the transition cast member Fog the transition for the frame that Lingo is currently recording:

```
set the frameTransition to member "Fog"
```

- **freeBlock**

Syntax: the freeBlock

This function indicates the size of the largest free contiguous block of memory, in bytes. A kilobyte (K) is 1024 bytes. A megabyte (MB) is 1024 kilobytes. In order to load a cast member, you need a free block at least as large as the cast member.

Example:

This statement determines whether the largest contiguous free block is smaller than 10K, and displays an alert if it is:

```
if the freeBlock < 10 * 1024 then ↵  
    alert "Not enough memory!"
```

- **freeBytes**, **memorySize**, and **ramNeeded** functions; **size of member** cast member property

- **freeBytes**

Syntax: the freeBytes

This function indicates the total number of bytes of free memory, which may not be contiguous. A kilobyte (K) is 1024 bytes. A megabyte (MB) is 1024 kilobytes.

This function differs from `freeBlock`, because it reports all free memory, not just contiguous memory.

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"
```

- **freeBlock**, **memorySize**, and **ramNeeded** functions; **size of member** cast member property

● frontWindow

Syntax: the frontWindow

This system property indicates which movie in a window is currently frontmost on the stage. When the stage is frontmost, the `frontWindow` is the `stage`. When a media editor or floating palette is frontmost, the `frontWindow` is `<void>`.

This property can be tested but not set.

Example:

This statement determines whether the stage is currently the frontmost window and, if it is, brings the window `Try This` to the front:

```
if the frontWindow = the stage then ↵  
  moveToFront window "Try This"
```

- **getaProp**

Syntax: `getaProp(list, positionOrProperty)`

This command identifies the value associated with the position or value specified by *positionOrProperty* in the list specified by *list*.

- When the list is a linear list, the result is the value at the position specified by *positionOrProperty*.
- When the list is a property list, the result is the value associated with the property specified by *positionOrProperty*.

The `getaProp` command gives the result `void` when the specified value is not in the list.

When used with linear lists, the `getaProp` command does the same as the `getAt` command.

Example:

This statement identifies the value in the third position of the linear list `Answers`, which consists of `[10, 12, 15, 22]`:

```
getaProp(Answers, 3)
```

The result is 15, because 15 is the third value in the list.

This statement identifies the property associated with the value 15 in the property list `Answers`, which consists of `[#a: 10, #b:12, #c:15, #d:22]`:

```
getaProp(Answers, #c)
```

The result is 15, which is the value associated with property `c`.

- **getOne, getProp** commands

- **getAt**

Syntax: `getAt (list, position)`

This command identifies the item in the position specified by *position* in the list specified by *list* . If the list contains fewer elements than the specified position, an alert appears.

The `getAt` command works with linear and property lists. This command does the same as the `getaProp` command for linear lists.

Example:

This statement has the message window display the third item in the list Answers, which consists of [10, 12, 15, 22]:

```
getAt (Answers, 3)
```

The result is 15.

- **getaProp** command

● getLast

Syntax: `getLast (list)`

This command identifies the last value in the list specified by *list*. The `getLast` command works with linear and property lists.

Example:

This statement identifies the last item in the list `Answers`, which consists of `[10, 12, 15, 22]`:

```
put getLast(Answers)
```

The result is 22.

This statement identifies the last item in the list `Bids`, which consists of `[#Gee:750, #Kayne:600, #Ohashi:850]`:

```
put getLast(Bids)
```

The result is 850.

● **getNthFileNameInFolder**

Syntax: `getNthFileNameInFolder (folderPath, fileNumber)`

This function returns a fileName from the directory folder at the specified path and number within the folder. To be found by the `getNthFileNameInFolder` function, Director movies must be set to be visible in the folder structure. However, the `getNthFileNameInFolder` function finds other types of files whether they are visible or invisible. If the function returns an EMPTY string, you have specified a number greater than the number of files in the folder.

Note: To specify other folder names, use the full path defined in the format for the specific platform the movie is running on.

● For example, in the Macintosh, use a pathname such as "HardDisk:Director:Movies:" To look for files on the Macintosh desktop, you would use the path "HardDisk:Desktop Folder:"

● To specify a pathname in Windows, use a directory path such as "C:\Director\Movies"

Example:

The following handler returns a list of filenames in the folder at the current path. To call the function, use parentheses, as in "put currentFolder()".

```
on currentFolder
  put [] into fileList
  repeat with i = 1 to the maxInteger
    put getNthFileNameInFolder(the pathName, i) ↵
      into n
    if n = EMPTY then exit repeat
    append(fileList, n)
  end repeat
  return fileList
end currentFolder
```

- **getOne**

Syntax: `getOne(list, value)`

This command identifies the position or property associated with the value specified by *value* in the list specified by *list*.

- When the list is a linear list, the result is the value's position in the list.
- When the list is a property list, the result is the property associated with the value in the list.

For values in the list more than once, only the first occurrence is displayed. The `getOne` command gives the result 0 when the specified value is not in the list.

When used with linear lists, the `getOne` command does the same 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]`:

```
getOne(Answers, 12)
```

The result is 2, because 12 is the second value in the list.

This statement identifies the property associated with the value 12 in the property list `Answers`, which consists of `[#a:10, #b:12, #c:15, #d:22]`:

```
getOne(Answers, 12)
```

The result is `b`, which is the property associated with the value 12.

- **getPos** command

- **getPos**

Syntax: `getPos(list , value)`

This command identifies the position of the value specified by *value* in the list specified by *list*. When the specified value is not in the list, the `getPos` command gives the value 0.

The `getPos` command works with linear and property lists.

For values in the list more than once, only the first occurrence is displayed. This command does the same as the `getOne` command when used for linear lists.

Example:

This statement identifies the position of the value 12 in the list `Answers`, which consists of `[#a:10, #b:12, #c:15, #d:22]`:

```
getPos(Answers, 12)
```

The result is 2, because 12 is the second value in the list.

- **getOne** command

- **getProp**

Syntax: `getProp(list , property)`

This command identifies the value associated with the property specified by *property* in the property list specified by *list*.

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

The `getProp` command is identical to the `getaProp` command, except that the `getProp` command displays an error message when the specified property is not in the list.

Example:

This statement identifies the value associated with the property `#c` of the property list `Answers`, which consists of `[#a:10, #b:12, #c:15, #d:22]`:

```
getProp(Answers, #c)
```

The result is 15, because 15 is the value associated with `#c`.

- **getOne** command

● **getPropAt**

Syntax: `getPropAt(list , index)`

This command identifies the property name associated with the position specified by *index* in the property list specified by *list*. If the specified item isn't in the list, Director displays an error message.

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

Example:

This statement displays the second property in the given list.

```
put getPropAt([#a:10, #b:20],2)
-- #b
```

- **global**

Syntax: `global variable1 [, variable2] [, variable3]...`

This keyword identifies a variable as a global variable so that it can be shared by other handlers or movies.

Every handler that examines or changes the contents of a global variable must use the global keyword to identify the variables as global. Otherwise, the handler treats the variable as a local variable even if it is declared to be global in another handler.

A global variable can be declared by a script, a handler, or a method, and its value can be used by other scripts, handlers, and methods.

Example:

```
global startingPoint
set startingPoint = whichMenu
```

- **showGlobals** command and **property** keyword

- **go**

Syntax: `go [to] [frame] whichFrame`
 `go [to] movie whichMovie`
 `go [to] [frame] whichFrame of movie whichMovie`

This command causes the playback head to jump to the frame specified by *whichFrame* of the movie specified by *whichMovie*. The expression *whichFrame* can be a marker label or an integer frame number. The expression *whichMovie* must specify a movie file. (If the movie is in another folder, *whichMovie* must specify the pathname.)

The phrase `go to the frame` has the playback head loop in the current frame. This is a convenient way to keep the playback head in the same frame, but keep Lingo active. Avoid using `go to the frame` in a frame that has a transition. This slows down the movie and overwhelms the processor as it constantly tries to perform the transition.

It's better to refer to marker labels instead of frame numbers, because editing a movie can cause frame numbers to change. Thus a command like `go to frame 35` can become incorrect. It's also easier to read your script if you use marker labels.

The `go to movie` command loads frame 1 of the movie. If the command is called from within a handler, the handler in which it is placed continues executing. If you want to suspend the handler while playing the movie, use the `play` command.

When you specify a movie to play, you must also specify its path if the movie is in a different folder. There's no need to include the movie's `.DIR` or `.DXR` file extension in the `go to movie` command. In fact, it's often better to omit the suffix during development. If you use `.DIR` files during the main development cycle and then later protect those movies, Lingo will fail when it encounters hard-coded filenames with the incorrect suffix. If you omit the file suffix, Lingo looks for a file with either suffix in the specified path.

The following are reset when loading a movie: the `beepOn`, the `constraint` properties, the `keyDownScript`, the `mouseDownScript`, the `mouseUpScript`; the `cursor` of `sprite` and `immediate` of `sprite` properties; the `cursor` and `puppetSprite` commands; and custom menus. However, the `timeoutScript` is not reset when loading a movie.

Example:

This statement sends the playback head to the marker named `start`:

```
go to "start"
```

This statement sends the playback head to the marker named `Memory` in the movie named `Noh Tale to Tell`:

```
go to frame "Memory" of movie "Noh Tale to Tell"
```

- **label**, **marker**, and **pathName** functions; **play** command

- **go loop**

Syntax: `go loop`

This command has the playback head continuously return to the first marker to the left and then loop back. If no markers are to the left of the playback head, the playback head continues to the right.

The `go loop` command is equivalent to the statement `go to the marker(0)` that was used in earlier versions of Lingo.

Example:

This statement has the movie loop between the current frame and the previous marker:

```
go loop
```

- **go, go next, go previous** commands

- **go next**

Syntax: `go next`

This command sends the playback head to the next marker in the movie. If no markers are to the right of the playback head, the playback head goes to the first marker to the left. If there are no markers to the left, the playback head goes to frame 1.

It 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
```

- **go, go loop, go previous** commands

- **go previous**

Syntax: `go previous`

This command sends the playback head to the previous marker in the movie. It 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 previous marker in the movie:

```
go previous
```

- **go, go loop, go next** commands

- **halt**

Syntax: halt

This command has Lingo exit the current handler and any handler that called it. After exiting all handlers, the `halt` command then stops the movie.

Example:

This statement checks whether the amount of free memory is less than 50K, and if it is, exits all handlers that called it, and then stops the movie:

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

- **abort** and **pass** commands; **exit** keyword

- **height of member**

Syntax: the height of member *whichCastmember*

This cast member property determines the height in pixels of the cast member specified by *whichCastmember*. The `height of member` property applies only to bitmap and shape cast members. It does not affect field or button cast members.

The `height of member` property can be tested but not set.

Example:

This statement assigns the height of cast member 50 to the variable `vHeight`:

```
put the height of member 50 into vHeight
```

- **spriteBox** command; the **width of member** property; **height of sprite** and **width of sprite** sprite properties

- **height of sprite**

Syntax: the height of sprite *whichSprite*

This sprite property determines the vertical size in pixels of the sprite specified by *whichSprite*. The height applies only to bitmap and shape cast members. It does not affect field or button cast members.

Setting this property does not have any effect on bitmap sprites unless the sprite's stretch property is set to TRUE. In order to set this property with Lingo, the sprite must be a puppet.

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. When you are changing several sprite properties--or several sprites--you only have to use the `updateStage` command once at the end of all the changes.

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

Example:

This statement sets the height of sprite 10 to 26 pixels:

```
set the height of sprite 10 to 26
```

This statement assigns the height of sprite (i + 1) to the variable `vHeight`:

```
put the height of sprite (i + 1) into vHeight
```

- **height of member, stretch of sprite, width of member, and width of sprite** sprite properties; **spriteBox** command

- **hilite**

Syntax: `hilite chunkExpression`

This command highlights (selects) the specified chunk in a field sprite. You can select any chunk that Lingo lets you define, such as a character, word, or line. In Windows, the color that highlights characters is the color set in Selected Items in the Display Properties dialog box.

Example:

This statement highlights the fourth word in the field cast member Comments, which contains the string "Thought for the Day":

```
hilite word 4 of member "Comments"
```

- **char...of**, **item...of**, **line...of**, and **word...of** chunk expression keywords; **delete** command; **mouseChar**, **mouseLine**, and **mouseWord** integer functions; **field** keyword; **selEnd** and **selStart** field property

- **hilite of member**

Syntax: the hilite of member *whichCastmember*

This button property determines whether a checkbox or radio button sprite is selected.

- When the `hilite of member` is TRUE, the checkbox or radio button is selected.
- When the `hilite of member` is FALSE, the checkbox or radio button is not selected.

When *whichCastmember* is a string, it is used as the cast member name. When *whichCastmember* is an integer, it is used as the cast member number.

The `hilite of member` button property can be tested and set. The default value is FALSE.

Example:

This statement checks whether the button named 2400 baud is selected and sets the baud rate to 2400 if it is:

```
if the hilite of member "2400 baud" = TRUE then ↵
    setBaudRate(2400)
```

This statement uses Lingo to select the button cast member powerSwitch by setting the `hilite of member` for the cast member to TRUE:

```
set the hilite of member powerSwitch to TRUE
```

- **checkBoxAccess** and **checkBoxType** properties

● **HMStoFrames**

Syntax: `HMStoFrames(hms , tempo , dropFrame , fractionalSeconds)`

This function converts movies measured in hours-minutes-seconds to the equivalent number of frames.

- The string expression *hms* specifies the time in the form "sHH:MM:SS.FFD", where:
 - s** is a dash (-) if the time is less than zero, or a space if the time is greater than or equal to zero.
 - HH** represents number of hours.
 - MM** represents number of minutes.
 - SS** represents number of seconds.
 - FF** represents fraction of a second if *fractionalSeconds* is TRUE. FF represents frames if *fractionalSeconds* is FALSE.
 - D** is the letter "d" if *dropFrame* is TRUE. D is a space if *dropFrame* is FALSE.
- The expression *tempo* specifies the tempo in frames per second.
- The *dropFrame* argument is a logical expression.. When TRUE replaces *dropFrame*, it is a drop-frame. When FALSE replaces *dropFrame*, it is not. When the string *hms* ends in a "d", the time is treated as a drop-frame, regardless of the value of *dropFrame*.
- The *fractionalSeconds* argument determines the meaning of the fractional seconds. When it is set to TRUE, the numbers after the seconds specify a fraction of a second, to the nearest hundredth of a second. When it is set to FALSE, the numbers after the seconds specify the number of residual frames.

Example:

This statement determines the number of frames in a 1-minute, 30.1-second movie when the tempo is 30 frames per second. The *dropFrame* and *fractionalSeconds* arguments are both turned off:

```
put HMStoFrames(" 00:01:30.10 ", 30, FALSE, FALSE)
-- 2710
```

- **framesToHMS** function

- **idle**

- on idle event handler

- **idleHandlerPeriod**

Syntax: `the idleHandlerPeriod`

This movie property determines the maximum number of ticks that passes until the movie sends an `idle` message. The default value is 0, which has the movie send idle handler messages as frequently as possible.

When the playback head enters a frame, Director starts a timer, repaints the appropriate sprites on the stage, and then issues an `enterFrame` event. At this point, if the amount of time set for the tempo setting has elapsed, Director generates an `exitFrame` event and goes to the next specified frame.

However, if the amount of time set for this frame hasn't elapsed, Director waits until the time runs out. During this time, Director periodically generates an `idle` message. The amount of time between `idle` events is determined by the `idleHandlerPeriod`.

Example:

The following statement has the movie send an `idle` message at most once per second:

```
set the idleHandlerPeriod = 60
```

- **managing memory**

- **idleLoadDone**

Syntax: `idleLoadDone (loadTag)`

This function reports whether Director has loaded all cast members that have the specified load tag.

- The result is TRUE when all cast members with the given tag have been loaded.
- The result is FALSE when some cast members with the given load tag are still waiting to be loaded.

Example:

This statement checks whether all cast members whose load tag is 20 have been loaded, and then plays the movie Kiosk if they are:

```
if idleLoadDone(20) = TRUE then play "Kiosk"
```

- **managing memory**

- **idleLoadMode**

Syntax: the idleLoadMode

This system property determines when the `preLoad` and `preLoadMember` commands attempt to load cast members during idle periods. The following values are possible values for the `idleLoadMode` and their effect:

- 0**--Does not perform idle loading
- 1**--Performs idle loading when there is free time between frames
- 2**--Performs idle loading during idle events
- 3**--Performs idle loading as frequently as possible.

Example:

This statement has the movie try as frequently as possible to load cast members designated for preloading by the `preLoad` and `preLoadMember` commands:

```
set the idleLoadMode = 2
```

- **managing memory**

- **idleLoadPeriod**

Syntax: `the idleLoadPeriod`

This property determines the number of ticks (1/60th of a second) that Director waits before returning to attempt to load cast members waiting to be loaded. The default value for `the idleLoadPeriod` is 0, which has Director service the load queue as frequently as possible.

Example:

This statement has Director service the set of cast members waiting to be loaded every 1/2 second (30 ticks):

```
set the idleLoadPeriod = 30
```

- **managing memory**

- ◆ **idleLoadTag**

Syntax: the idleLoadTag

This system property is a number that identifies, or "tags", the cast members that have been queued for loading when the computer is idle. The property can be tested and set. When you set the property, it can be any number that you choose.

Example:

This statement makes the number 10 the idle load tag.

```
set the idleLoadTag = 10
```

- ◆ **managing memory**

- **idleReadChunkSize**

Syntax: the `idleReadChunkSize` of member *whichCastmember*

This movie property determines the maximum number of bytes from a cast member that Director can load when it loads cast members from the load queue. The property can be tested and set.

Example:

This statement specifies that 500K is the maximum number of bytes from cast member number 50 that Director can load in one attempt at loading cast members in the load queue:

```
set the idleReadChunkSize to 500000
```

- **managing memory**

• if

Syntax: if *logicalExpression* then *then-statement*

or

```
if logicalExpression then then-statement  
else else-statement  
end if
```

or

```
if logicalExpression then  
statement(s)  
end if
```

or

```
if logicalExpression then  
statement(s)  
else  
statement(s)  
end if
```

or

```
if logicalExpression1 then  
statement(s)  
else if logicalExpression2 then  
statement(s)  
else if logicalExpression3 then  
statement(s)  
end if
```

The *if-then* structure evaluates the *logicalExpression* specified by *logicalExpression*.

- When the condition is TRUE, Lingo executes the *statement(s)* that follow *then*.
- When it is FALSE, Lingo executes the *statement(s)* following *else*. If no statements follow *else*, Lingo exits the *if-then* structure.

When the condition is a property, Lingo automatically checks whether the property is TRUE. You don't need to explicitly add the phrase "`= TRUE`" after the property, but it is common practice to do so.

The *else* portion of the statement is optional. If you need to have more than one *then-statement* or *else-statement*, you must end with the form *end if*.

When you use *else*, it always corresponds to the previous *if* statement. This means that sometimes you need to include an *else nothing* statement to associate an *else* keyword with the proper *if* keyword.

Examples:

This statement checks whether the Return key was pressed, and then continues if it was:

```
if the key = RETURN then continue
```

This statement checks whether the color QuickDraw software is available on a Macintosh. If it is available, Lingo plays the movie Color Movie. If the color QuickDraw software isn't available, Lingo plays the movie Black & White Movie:

```
if the colorQD = TRUE then play "Color Movie"  
else play "Black & White Movie"
```

This handler checks whether the Command and "q" keys were pressed simultaneously, and then executes the subsequent statements if it was:

```
on keyDown  
if (the commandDown) and (the key = "q") then  
  cleanUp  
  quit  
end if  
end keyDown
```

- **case** keyword

• ilk

Syntax: `ilk(list)`
 `ilk(item, type)`

This function indicates the type of a list, rect, or point.

- The syntax `ilk(list)` returns whether *list* is a linear list or property list. For linear lists, `ilk(list)` returns `#linearList`; for property lists, `ilk(list)` returns `#propList`.
- The syntax `ilk(item, type)` compares the object represented by *item* and indicates whether the object is of the specified *type*. When the object is of the specified type, the `ilk` function returns TRUE (1). When the object isn't of the specified type, the `ilk` function returns FALSE (0). The following are the values that would be returned for each combination of linear list, property list, point, or rect items and types:

item	possible type=returned value
linear list:	<code>#list=1, #linearlist=1, #proplist=0, #point=0, #rect=0</code>
property list:	<code>#list=1, #linearlist=0, #proplist=1, #point=0, #rect=0</code>
point:	<code>#list=1, #linearlist=1, #proplist=0, #point=1, #rect=0</code>
rect:	<code>#list=1, #linearlist=1, #proplist=0, #point=0, #rect=1</code>

Examples:

This statement identifies whether the list named `bids` is a property list and displays the result in the message window:

```
put ilk(bids, #proplist)
```

Because the list is a property list the message window displays 1, the numeric equivalent of TRUE.

This statement identifies whether the variable `vTotal` is a list and displays the result in the message window:

```
put ilk(vTotal, #list)
```

Because the variable is not a list, the message window displays 0, which is the numeric equivalent of FALSE.

• importFileInto

Syntax: `importFileInto member whichCastmember , fileName`

```
importFileInto member whichCastmember of ↵  
castLib whichCast, fileName
```

This command replaces the content of the cast member specified by *whichCastmember* with the file specified by *fileName*.

The `importFileInto` command is useful when you are finishing developing a movie. Use it at that time to embed media that you have kept linked and external so that it could be edited during the project. However, using this command in projectors can be a problem, because imported files can quickly consume memory.

Example:

This statement replaces the content of the sound cast member Memory with the sound file Wind:

```
importFileInto member "Memory", "Wind"
```

•
in

number of chars in, number of items in, number of lines in, and number of words in functions.

● inflate rect

Syntax: `inflate (rectangle , widthChange , heightChange)`

This command changes the dimensions of the rectangle specified by *rectangle*. The change is relative to the center of the rectangle.

- The *widthChange* parameter specifies how much the rectangle changes horizontally.
- The *heightChange* parameter specifies how much the rectangle changes vertically.

The total change in each direction is twice the number you specify. For example, replacing *widthChange* with 15 increases the rectangle's width by 30 pixels.

Values less than 0 for horizontal or vertical reduce the rectangle's size.

Examples:

This statement increases the width of the rectangle by 4 pixels and the height by 2 pixels:

```
inflate (Rect(10, 10, 20, 20), 2, 1)
-- Rect (8, 9, 22, 21)
```

This statement increases both the height and width of the rectangle by 20 pixels:

```
inflate (Rect(0, 0, 100, 100), -10, -10)
-- Rect (-10, -10, 110, 110)
```

• ink of sprite

Syntax: the ink of sprite *whichSprite*

This sprite property determines the ink effect applied to the sprite specified by *whichSprite*.

The following ink effects are available:

- 0--Copy
- 1--Transparent
- 2--Reverse
- 3--Ghost
- 4--Not copy
- 5--Not transparent
- 6--Not reverse
- 7--Not ghost
- 8--Matte
- 9--Mask
- 32--Blend
- 33--Add pin
- 34--Add
- 35--Subtract pin
- 36--Background transparent
- 37--Lightest
- 38--Subtract
- 39--Darkest

In the case of background transparent (ink effect 36), you set the color that becomes transparent by selecting the color from the background color chip in the tools window while the sprite is selected in the score. You can do the same thing by using Lingo to set the `backColor` property, but this is unpredictable when the sprite has more than 1-bit color.

If you set this property within a script while the playback head is not moving, be sure to use the `updateStage` command to redraw the stage. If you change several sprite properties--or several sprites--you need to use only one `updateStage` command at the end of all the changes.

For further information about ink effects, see *Using Director*.

The ink sprite property can be tested and set. To change any sprite property using Lingo, the sprite must be a puppet.

Example:

This statement changes the variable `currentInk` to the value for the ink effect of sprite (`i + 1`):

```
put the ink of sprite 3 into currentInk
```

This statement gives sprite (`i + 1`) a matte ink effect by setting the ink effect of sprite property to 8, which specifies matte ink:

set the ink of sprite (i + 1) to 8

- backColor of sprite and foreColor of sprite sprite property, Ink pop-up menu (score)

● insertFrame

Syntax: insertFrame

This command duplicates the current frame and its content. The duplicate frame is inserted after the current frame and then becomes the current frame. It can be used only during a score recording session.

This command does the same as the `duplicateFrame` command.

Example:

The following handler generates a frame that has the transition cast member Fog assigned in transition channel followed by a set of empty frames. The number of frames is determined by the argument `numberOfFrames`.

```
on animBall numberOfFrames
  beginRecording
    set the frameTransition to -
    the number of member "Fog"
    repeat with i = 0 to numberOfFrames
      insertFrame
    end repeat
  endRecording
end
```

- ◆ **inside**

Syntax: `inside(point, rectangle)`

This function indicates whether the point specified by *point* is within the rectangle specified by *rectangle*.

- ◆ When the point is within the rectangle, the `inside` function is TRUE.
- ◆ When the point is outside the rectangle, the `inside` function is FALSE.

Example:

This statement indicates whether the point `Center` is within the rectangle `Zone` and displays the result in the message window:

```
put inside(Center, Zone)
```

- ◆ **map**, **mouseH**, **mouseV**, and **point** functions

- **installMenu**

Syntax: `installMenu whichCastmember`

This command installs the menu defined in the field cast member specified by *whichCast member*. These custom menus appear only while the movie is playing. To remove the custom menus, use the `installMenu` command with no argument, or with 0 as the argument.

For an explanation of how menu items are defined in a field cast member, refer to the `menu: keyword`.

Examples:

This statement installs the menu defined in field cast member 37:

```
installMenu 37
```

This statement installs the menu defined in the field cast member named Menubar by using the `number of member` property to refer to the field cast member:

```
installMenu member "Menubar"
```

This statement disables menus that were installed by the `installMenu` command:

```
installMenu 0
```

- **menu** keyword

- **integer**

Syntax: `integer(numericExpression)`

This function rounds the value of *numericExpression* to the nearest whole integer.

You can force an integer to be a string by using the `string()` function.

Example:

This statement rounds off the number 3.75 to the nearest whole integer:

```
put integer(3.75)
-- 4
```

This statement rounds off the value in parentheses. This provides a usable value for the `locH` of `sprite`, which requires an integer:

```
set the locH of sprite 1 to
to integer(0.333 * stageWidth)
```

- **float** and **string** functions

- **integerP**

Syntax: `integerP(expression)`

This function indicates whether the expression specified by *expression* is an integer:

- When *expression* can be evaluated to an integer, `integerP` is TRUE (1).
- When *expression* cannot be evaluated to an integer, `integerP` is FALSE (0).

The "P" in `integerP` stands for "predicate."

Examples:

This statement checks whether 3 can be evaluated to an integer. Because it is an integer, the message window displays the number 1, which is the numeric equivalent of TRUE:

```
put integerP(3)
-- 1
```

This statement checks whether 3 can be evaluated to an integer. Because 3 is surrounded by quotes, it cannot be evaluated to an integer, so the message window displays the number 0, which is the numeric equivalent of FALSE:

```
put integerP("3")
-- 0
```

This statement checks whether the numerical value of the string in field cast member Entry is an integer, and displays an alert if it isn't.

```
if integerP(value(field "Entry")) = FALSE then ↵
  alert "Please enter an integer."
```

- **floatP, objectP, stringP, and symbolP** functions

- **intersect**

Syntax: `intersect(rectangle1 , rectangle2)`

This function determines the rectangle formed where *rectangle1* and *rectangle2* intersect.

Example:

This statement assigns the variable `newRectangle` the rectangle formed where `rectangle toolkit` intersects `rectangle Ramp`:

```
set newRectangle = intersect(toolkit, Ramp)
```

- **map** and **rect** functions

- **into**

This code fragment occurs in a number of Lingo constructs, such as `put...into`.

• item...of

Syntax: item *whichItem* of *chunkExpression*

or

 item *firstItem* to *lastItem* of *chunkExpression*

This chunk expression keyword specifies an item or a range of items in a chunk expression. An item in this case is any sequence of characters delimited by commas.

The terms *whichItem* , *firstItem* , and *lastItem* must be integers or integer expressions that refer to the position of items in the chunk.

Chunk expressions refer to any character, word, item, or line in any source of strings. Sources of strings include field cast members and variables that hold strings.

When the number that specifies the last item is greater than the item's position in the chunk expression, the actual last item is specified instead.

Examples:

This statement determines the third item in the chunk expression that consists of names of colors and displays the result in the message window:

```
put item 3 of "red, yellow, blue green, orange"  
-- "blue green"
```

The result is the entire chunk blue green because this is the entire chunk within the commas.

This statement determines the third through fifth item in the chunk expression and displays the result in the message window:

```
put item 3 to 5 of "red, yellow, blue green, orange"  
-- "blue green, orange"
```

This statement attempts to determine the third through fifth items in the chunk expression. Because there are only four items in the chunk expression, the fourth item is used instead of the fifth item. The result appears in the message window:

```
put item 3 to 5 of "red, yellow, blue green, orange"  
-- " blue green, orange"  
  
put item 5 of "red, yellow, blue green, orange"  
-- ""
```

This statement inserts the item Desk as the fourth item in the second line of the field cast member All Bids:

```
put "Desk" into item 4 of line 2 -  
of member "All Bids"
```

- char...of, line...of, and word...of chunk expression keywords; number of items in chunk function

● itemDelimiter

Syntax: the itemDelimiter

This property indicates the special character used to separate items.

You can use the `itemDelimiter` function to parse filenames by setting `itemDelimiter` to a colon (:) on the Macintosh or a backslash (\) in Windows. Restore the `itemDelimiter` to a comma (,) for normal operation. Be sure to restore it to "," for normal operation.

The `itemDelimiter` function can be tested and set.

Example:

This handler determines the last component in a Macintosh pathname. The handler first records what the current delimiter is, and then changes the delimiter to a colon (:). When a colon is the delimiter, Lingo can use the `last item of` to determine the last item in the chunk that makes up a Macintosh pathname. Before exiting, the delimiter is reset to its original value.

```
on getLastComponent pathName
    set save = the itemDelimiter
    set the itemDelimiter = ":"
    set f = the last item of pathName
    set the itemDelimiter = save
    return f
end
```

- **key**

Syntax: the key

This function indicates the last key that was pressed. (This value is the ANSI value assigned to the key, not the numerical value.)

This function can be used for testing keys within event script and for navigation/keyboard shortcuts.

You can use the `key` to write handlers that perform certain actions when the user presses specific keys. This is a way to provide keyboard shortcuts and other forms of interactivity for the user. When used in a primary event handler, the actions you specify are the first to be executed.

Examples:

These statements have the movie pause when the user presses Return. By setting the `keyDownScript` property to `checkKey`, the `on startMovie` handler makes the `checkKey` handler the first event handler executed when a key is pressed. The `checkKey` handler checks whether the Return key is pressed and pauses the movie if it is:

```
on startMovie
  set the keyDownScript to "checkKey"
end startMovie

on checkKey
  if the key = RETURN then pause
end
```

This `keyDown` handler checks whether the last key pressed is the Enter key, and then calls the handler `addNumbers` if it is:

```
on keyDown
  if the key = ENTER then addNumbers
end keyDown
```

- **commandDown, controlDown, keyCode, and optionDown** functions

- **keyCode**

Syntax: the keyCode

This function gives the numerical code for the last key pressed. (This keyboard code is the key's numerical value, not the ANSI value.)

You can use the `keyCode` function to detect when the user has pressed the arrow or function keys, which cannot be specified by the `key` function. The value of `keyCode` varies on international keyboards.

The `keyCode` function can be tested but not set.

Examples:

This handler uses the message window to display the appropriate key code each time a key is pressed:

```
on enterFrame
  set the keydownScript = "put the keyCode"
end
```

This statement checks whether the up arrow (whose key code is 126) is pressed, and goes to the previous marker if it is:

```
if the keyCode = 126 then go to marker(-1)
```

- **commandDown, controlDown, key, and optionDown** functions

- **keyDown**

- on keyDown handler

- **keyDownScript**

Syntax: `the keyDownScript`

This property specifies the Lingo that is executed when a key is pressed. The Lingo can be a simple statement or a calling script for a handler.

When a key is pressed and the `keyDownScript` is defined, Lingo executes the instructions specified for the `keyDownScript` first. Unless the instructions include the `pass` command so that the `keyDown` message can pass on to other objects in the movie, no other `keyDown` handlers are executed.

Setting the `keyDownScript` property does the same 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 the `keyDownScript` to `if the key = RETURN then continue`. When this is in effect and the movie is paused, the movie always continues whenever the user presses the Return key.

```
set the keyDownScript to "if the key = RETURN then continue"
```

- **keyUpScript**, **mouseDownScript**, and **mouseUpScript** properties

● keyPressed

Syntax: the keyPressed

This system property gives the character assigned to the key that was last pressed. The result is in the form of a string. When no key has been pressed, the `keyPressed` is an empty string.

The `keyPressed` property works inside repeat loops. This property can be tested but not set.

Example:

The following statement checks whether the user pressed Enter in Windows or Return on a Macintosh and runs the handler `updateData` if he or she did:

```
if the keyPressed = RETURN then updateData
```

● keyUpScript

Syntax: the keyUpScript

This property specifies the Lingo that is executed when a key is released. The Lingo can be a simple statement or a calling script for a handler.

When a key is released and the `keyUpScript` is defined, Lingo executes the instructions specified for the `keyUpScript` first. Unless the instructions include the `pass` command so that the `keyUp` message can pass on to other objects in the movie, no other `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 the `keyUpScript` to `if the key = RETURN then continue`. When this is in effect and the movie is paused, the movie always continues whenever the user presses the Return key.

```
set the keyUpScript -  
  to "if the key = RETURN then continue"
```

- **label**

Syntax: `label (expression)`

This function indicates the frame associated with the marker label specified by *expression*. The term *expression* should be a label in the current movie; if it's not, this function returns 0.

Examples:

This statement sends the playback head to the tenth frame after the frame labeled Start:

```
go to label("Start") + 10
```

This statement assigns the frame number of the fourth item in the label list to the variable `whichFrame`:

```
put label(line 4 of the labelList) into whichFrame
```

- **go** and **play** commands; **labelList** and **marker** functions

- **labelList**

Syntax: the labelList

This function gives a listing of the frame labels in the current movie, one label per line.

Example:

This statement makes a listing of frame labels the content of the field cast member Key Frames:

```
put the labelList into field "Key Frames"
```

- **label** and **marker** functions

- **last**

Syntax: the last *chunk* in (*chunkExpression*)

This function identifies the last chunk specified by *chunk* of the chunk expression specified by *chunkExpression*.

Chunk expressions refer to any character, word, item, or line in any container of characters. Containers include the contents of field cast members; variables that hold strings; and specified characters, words, items, lines, and ranges within containers.

Examples:

This statement identifies the last word of the string "Macromedia, the multimedia company" and displays the result in the message window:

```
put the last word of "Macromedia,↵  
the multimedia company"
```

The result is the word `company`.

This statement identifies the last character of the string "Macromedia, the multimedia company" and displays the result in the message window:

```
put the last char of "Macromedia,↵  
the multimedia company"
```

The result is the letter `y`.

- **char...of** and **word...of** chunk expression keywords

- **lastClick**

Syntax: the lastClick

This function gives the time in ticks (60ths of a second) since the mouse button was last pressed.

The lastClick can be tested, but not set.

Example:

This statement checks whether it has been 10 seconds since the last mouse click, and sends the playback head to the marker No Click if it has:

```
if the lastClick > 10 * 60 then go to "No Click"
```

- **lastEvent**, **lastKey**, and **lastRoll** functions; **startTimer** command

- **lastEvent**

Syntax: the lastEvent

This function gives the time in ticks (60ths of a second) since the last mouse click, rollover, or key press occurred.

Example:

This statement checks whether it has been 10 seconds since the last mouse click, rollover, or key press, and sends the playback head to the marker Help if it has:

```
if the lastEvent > 10 * 60 then go to "Help"
```

- **lastClick**, **lastKey**, and **lastRoll** functions; **startTimer** command

● **lastFrame**

Syntax: `the lastFrame`

This property is the number of the last frame in the movie.

The `lastFrame` property can be tested but not set.

Example:

This statement displays the number of the last frame of the movie in the message window:

```
put the lastFrame
```

- **lastKey**

Syntax: the lastKey

This function gives the time in ticks (60ths of a second) since the last key was pressed.

Example:

This statement checks whether it has been 10 seconds since the last key was pressed, and sends the playback head to the marker "No Key" if it has:

```
if the lastKey > 10 * 60 then go to "No Key"
```

- **lastClick**, **lastEvent**, and **lastRoll** functions; **startTimer** command

- **lastRoll**

Syntax: the lastRoll

This function gives the time in ticks (60ths of a second) since the mouse was last moved.

Example:

This statement checks whether it has been 45 seconds since the mouse was last moved, and sends the playback head to the marker No Roll if it has:

```
if the lastRoll > 45 * 60 then go to "Attract Loop"
```

- **lastClick**, **lastEvent**, and **lastKey** functions; **startTimer** command

- **left of sprite**

Syntax: the left of sprite *whichSprite*

This sprite property is the left horizontal coordinate of the bounding rectangle of the sprite specified by *whichSprite*.

Sprite coordinates are measured in pixels, starting with (0,0) at the upper left corner of the stage.

The `left of sprite` property can be tested, but not set. Use the `spriteBox` command to set the left horizontal coordinate of a sprite.

Examples:

The following statement determines whether the sprite's left edge is to the left of the stage's left edge. If the sprite's left edge is to the stage's left edge, the script runs the handler `offLeftEdge`:

```
if the left of sprite 3 < 0 then offLeftEdge
```

This statement measures the left horizontal coordinate of the sprite numbered (i + 1) and assigns the value to the variable named `vLowest`:

```
put the left of sprite (i + 1) into vLowest
```

- **bottom of sprite, height of sprite, locH of sprite, locV of sprite, right of sprite, top of sprite, and width of sprite properties; spriteBox command**

- **length**

Syntax: `length(string)`

This function gives the number of characters in the string specified by *string*. Spaces and control characters like Tab and Return count as characters.

Examples:

This statement displays the number of characters in the string "Macro"&"media":

```
put length("Macro" & "media")  
-- 10
```

This statement checks whether the content of the field cast member File Name has more than 31 characters and displays an alert if it does:

```
if length(field "File Name") > 31 then ↵  
    alert "That file name is too long."
```

- **chars** and **offset** functions

- **line...of**

Syntax: line *whichLine* of *chunkExpression*

or

line *firstLine* to *lastLine* of *chunkExpression*

This chunk expression keyword specifies a line or a range of lines in a chunk expression. A line chunk is any sequence of characters delimited by Returns.

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.

Examples:

This statement assigns the first four lines of the variable `Action` to the field cast member `To Do`:

```
set the text of member "To Do" = line 1 to 4 ↵
of Action
```

This statement inserts the word "and" after the second word of the third line of the string assigned to the variable `Notes`:

```
put "and" after word 2 of line 3 of Notes
```

- **char...of, item...of, word...of**, and chunk expression keywords; **number of words**
in chunk function

• the lineCount of member

Syntax: the lineCount of member *whichCastMember*

This field cast member property indicates the number of lines that appear in the field cast member on the stage. The number of lines depends on how the string wraps, not the number of carriage returns in the string.

Example:

This statement determines how many lines the field cast member Today's News has when it appears on the stage and assigns the value to the variable `numberOfLines`:

```
put the lineCount of member "Today's News" -  
into numberOfLines
```

● **lineHeight**

Syntax: `lineHeight(member whichCastMember, lineNumber)`

This function gives the height, in pixels, of a specific line in the specified field cast member.

Example:

This statement determines the height, in pixels, of the first line in the field cast member Today's News and assigns the result to the variable headline:

```
put lineHeight(member "Today's News",1) into headline
```


- **lineHeight of member**

Syntax: the lineHeight of member *whichCastmember*

This field property determines the line spacing used to display the specified field cast member. The parameter *whichCastmember* can be either a cast member name or number.

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

Example:

This statement sets the variable `oldHeight` to the current `lineHeight` of member setting for the field cast member Rokujo Speaks:

```
put the lineHeight of member "Rokujo Speaks" into ↵
  oldHeight
```

- **text of member** property; **alignment of member**, **font of member**, **fontSize of member**, and **fontStyle of member** field properties

● **linePosToLocV**

Syntax: `linePosToLocV(member whichCastMember, lineNumber)`

This function gives a specific line's distance, in pixels, from the top edge of the field cast member.

Example:

This statement measures the distance, in pixels, that the second line of the field cast member Today's News is from the top of the field cast member and assigns the result to the variable `startOfString`:

```
put linePosToLocV(member "Today's News",2) →  
into startOfString
```

- **the lineSize of member**

Syntax: the lineSize of member *whichCastmember*

This shape cast member property determines the thickness, in pixels, of the border of the specified shape cast member. It can be tested and set.

Example:

This statement sets the thickness of the shape cast member Answer Box to 5 pixels:

```
set the lineSize of member "Answer Box" = 5
```

● **lineSize of sprite**

Syntax: the lineSize of sprite *whichSprite*

This sprite property determines the thickness, in pixels, of the border of the sprite specified by *whichSprite*. The `lineSize of sprite` property applies only to shape sprites. For non-rectangular shapes the border is the edge of the shape, not its bounding rectangle.

The `lineSize of sprite` property can be tested and set. For a sprite property to be set using Lingo, the sprite must be a puppet.

Examples:

This statement displays the thickness of the border of sprite 4:

```
put the lineSize of sprite 4 into thickness
```

This statement sets the thickness of the border of sprite 4 to 3 pixels:

```
set the lineSize of sprite 4 to 3
```

- **list**

Syntax: `list(value1 , value2 , value3...)`

This function defines a linear list made up of the values specified by *value1*, *value2*, *value3*.... This is an alternative to using square brackets ([]) to create a list.

Example:

This statement sets the variable named `designers` equal to a linear list that contains the names `Gee`, `Kayne`, and `Ohashi`:

```
set designers = list("Gee", "Kayne", "Ohashi")
```

The result is the list ["Gee", "Kayne", "Ohashi"].

- **listP**

Syntax: `listP(item)`

This function indicates whether the item specified by *item* is a list, rect, or point.

- When `listP` is TRUE (1), the item specified by *item* is a list, rect, or point.
- When `listP` is FALSE (0), the item specified by *item* is not a list, rect, or point.

Example:

This statement checks whether the list in the variable `designers` is a list, rect, or point and displays the result in the message window:

```
put listP(designers)
```

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

This statement checks whether the point in the variable `Spot` is a list, rect, or point and displays the result in the message window:

```
put listP(Spot)
```

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

- **ilk** function

- **loaded of member**

Syntax: the loaded of member *whichCastMember*

This cast member property specifies whether the cast member specified by *whichCastMember* is loaded into memory.

- When the loaded of member is TRUE, the cast member is loaded into memory.
- When the loaded of member is FALSE, the cast member is not loaded into memory.

Different cast member types have slightly different behavior for loading.

- Shape and script cast members are always loaded in memory.
- Movie cast members are never unloaded.
- Digital video cast members can be preloaded and unloaded independently of whether they are being used. (A digital video cast member plays faster from memory than from disk.)

The loaded of member property can be tested but not set.

Example:

This statement checks whether cast member Demo Movie is loaded in memory, and goes to an alternate movie if it isn't:

```
if the loaded of member "Demo Movie" = FALSE then ↵  
go to "Waiting"
```

- **size of member** cast property; **preLoad** and **unLoad** commands; **ramNeeded** function

- **loc of sprite**

Syntax: the loc of sprite *whichSprite*

This property determines the stage coordinates of the specified sprite. The value is given as a point. The `loc of sprite` can be tested and set.

Example:

This statement checks the stage coordinates of sprite 6. The result is the point (50, 100):

```
put the loc of sprite 6
-- point(50, 100)
```

- **bottom of sprite, height of sprite, left of sprite, locV of sprite, right of sprite, top of sprite, and width of sprite** sprite properties

- **locH of sprite**

Syntax: the locH of sprite *whichSprite*

This sprite property is the horizontal position of the specified sprite's registration point. Sprite coordinates are relative to the upper left corner of the stage. See "Registration points" in Chapter 4 of *Using Director* for information about registration points.

The `locH of sprite` property can be tested and set. For a sprite property to be set using Lingo, the sprite should be a puppet. Otherwise, the sprite reverts to the cast member set in the score when the playback head exits the frame.

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 are changing several sprite properties--or several sprites--you need only one `updateStage` command at the end of all the changes.

Examples:

This statement checks whether the horizontal position of sprite 9's registration point is to the right of the right edge of the monitor, and moves the sprite's right edge to the edge of the stage if it is:

```
if the locH of sprite 9 > the stageRight then -  
    set the locH of sprite 9 to the stageRight
```

This statement puts sprite 15 at the same horizontal location as the mouse click:

```
set the locH of sprite (15) to the mouseH
```

- **bottom of sprite, height of sprite, left of sprite, locV of sprite, right of sprite, top of sprite, and width of sprite** sprite properties; **spriteBox** and **updateStage** commands

• **locToCharPos**

Syntax: `locToCharPos (member whichCastMember, location)`

This function returns a number that identifies which character in the specified field cast member is closest to the point specified by *location*. 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 statements determines which character is closest to the point (100, 100) in the field cast member Today's News and assigns the result to the variable `PageDesign`:

```
put locToCharPos(member "Today's News", point(100,100))→  
into PageDesign
```

• locVToLinePos

Syntax: locVToLinePos (member *whichCastMember*, *locV*)

This function returns the number of the line of characters that appears at the vertical position specified by *locV*. The *locV* value is the number of pixels from the top of the field cast member, not the part of the field cast member that currently appears on the stage.

Example:

This statement determines which line of characters appears 150 pixels from the top of the field cast member Today's News and assigns the result to the variable pageBreak:

```
put locVToLinePos(member "Today's News", 150) into pageBreak
```

- **locV of sprite**

Syntax: the locV of sprite *whichSprite*

This sprite property is the vertical position of the specified sprite's registration point. Sprite coordinates are relative to the upper left corner of the stage. See "Registration points" in Chapter 4 of *Using Director* for information about registration points.

The `locV of sprite` property can be tested and set. For a sprite property to be set using Lingo, the sprite should be a puppet. Otherwise, the sprite reverts to the cast member set in the score when the playback head exits the frame.

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 are changing several sprite properties--or several sprites--you need only one `updateStage` command at the end of all the changes.

Examples:

This statement checks whether the vertical position of sprite 9's registration point is to the below the bottom of the stage, and moves the sprite's bottom edge to the bottom of the stage if it is:

```
if the locV of sprite 9 > the stageBottom then -  
    set the locV of sprite 9 to the stageBottom
```

This statement puts sprite 15 at the same vertical location as the mouse click:

```
set the locV of sprite (15) to the mouseV
```

- **bottom of sprite, height of sprite, left of sprite, locH of sprite, right of sprite, top of sprite, and width of sprite** sprite properties; **spriteBox** and **updateStage** commands

• log

Syntax: `log(number)`

This function calculates the natural logarithm of the number specified by *number* , which must be a decimal number greater than zero.

Examples:

This statement assigns the natural logarithm of 10.5 to the variable `Answer`. The result is calculated to two decimal places:

```
set Answer = log(10.5)
```

This statement calculates the natural logarithm of the square root of the value `Number`, and then assigns the result to the variable `Answer`:

```
set Answer = log(the sqrt of Number)
```

- **long**

- the **date** and **time** functions

• loop

Syntax: loop

This keyword refers to the marker. The `loop` keyword with the `go to` command is equivalent to the statement `go to marker`.

Example:

This handler loops the movie between the previous marker and the current frame:

```
on exitFrame
  go loop
end exitFrame
```

- **loop of member**

Syntax: the loop of member *whichCastmember*

This digital video cast member property determines whether the specified digital video movie cast member is set to loop.

- When the loop of member is TRUE(1), the digital video movie cast member loops.
- When the loop of member is set to FALSE(0), the digital video movie cast member doesn't loop.

Example:

This statement sets the QuickTime movie cast member Demo to loop:

```
set the loop of member "Demo" to 1
```


● machineType

Syntax: the machineType

This function indicates the kind of computer that is currently being used. These codes indicate the type of computer:

1	Macintosh 512Ke
2	Macintosh Plus
3	Macintosh SE
4	Macintosh II
5	Macintosh IIx
6	Macintosh IIcx
7	Macintosh SE/30
8	Macintosh Portable
9	Macintosh IIfx
11	Macintosh IIx
15	Macintosh Classic
16	Macintosh IIsi
17	Macintosh LC
18	Macintosh Quadra 900
19	PowerBook 170
20	Macintosh Quadra 700
21	Classic II
22	PowerBook 100
23	PowerBook 140
24	Macintosh Quadra 950
25	Macintosh LCIII
27	PowerBook Duo 210
28	Macintosh Centris 650
30	PowerBook Duo 230
31	PowerBook 180
32	PowerBook 160
33	Macintosh Quadra 800
35	Macintosh LC II
42	Macintosh IIfx
45	PowerMac 7100/70
46	Macintosh IIvx
47	Macintosh Color Classic
48	PowerBook 165c
50	Macintosh Centris 610
52	PowerBook 145
53	PowerComputing 8100/100
73	PowerMac 6100/60
76	Macintosh Quadra 840av
256	IBM PC-type machine

Note: These codes are for general classification purposes only. It is unwise to use them to make assumptions about the performance or screen size of the computer your movie is running on.

Example:

This statement checks whether the computer is a Macintosh Classic and plays the movie Classic Movie if it is:

```
if the machineType = 15 then play "Classic Movie"
```

These statements check whether the current operating system is Windows or Macintosh, and runs a handler intended for that platform:

```
if the machineType = 256 then
  WindowsActions
else
  MacintoshActions
end if
```

- **colorDepth** property; **colorQD** function

● map

Syntax: `map(targetRect, sourceRect, destinationRect)`

or

`map(targetPoint, sourceRect, destinationRect)`

This function is used to position and size a rectangle or point, based on the relationship of a source rectangle to a target rectangle.

Example:

This handler modifies the rectangle of sprite n so that the sprite's new rect has the same relationship to its old rect as the dimensions of the stage have to the rect of sprite 2:

```
on scaleMySprite n
  set the stretch of sprite to TRUE
  set the rect of sprite n = ↵
    map(the rect of sprite n, ↵
      the rect of sprite 2, ↵
      the rect of the stage)
  updateStage
end scaleMySprite
```

● **margin of member**

Syntax: the margin of member *whichCastmember*

This field cast member property determines the size, in pixels, of the margin inside the field box.

Example:

The following sets the margin inside the box for the field cast member Today's News to 15 pixels:

```
set the margin of member "Today's News" to 15
```

- **marker**

Syntax: `marker (integerExpression)`

This function returns the frame number of markers before or after the current frame. This can be useful for implementing a "next" or "previous" button, or for setting up an animation loop.

The *integerExpression* can evaluate to any positive or negative integer or zero. For example:

- `marker(2)` returns the frame number of the second marker after the current frame.
- `marker(1)` returns the frame number of the first marker after the current frame.
- `marker(0)` returns the frame number of the current frame, if the current frame is marked, or the frame number of the previous marker if the current frame is not marked.
- `marker(-1)` returns the frame number of the first marker before the current frame.
- `marker(-2)` returns the frame number of the second marker before the current frame.

Examples:

This statement sends the playback head to the beginning of the current frame:

```
go to marker(0)
```

This statement sets the variable `nextMarker` equal to the next marker in the score:

```
put marker(1) into nextMarker
```

- **go** command; **frame**, **labelList** and **label** functions

- **mAtFrame**

Syntax: method mAtFrame *frameNumber, subFrameNumber*
 [*statements*]
 end mAtFrame

This special message is used by Lingo in conjunction with any XObject or that has been assigned to the `perFrameHook` property, as follows:

```
set the perFrameHook to objectName
```

Subsequently, the `mAtFrame` message is automatically sent to the object every time the playback head reaches a new frame, or every time an internal subframe is reached within a visual transition.

The functionality within `mAtFrame` must be supplied by the XObject definition (as opposed to predefined methods). That is why `mAtFrame` is technically called a message, instead of a predefined method.

The `perFrameHook` property is primarily designed for use with XObjects that need to be called at every subframe, such as frame-by-frame video recorders. Objects should generally use an `on stepMovie` handler if they need to be called at every frame.

- **method** keywords; **perFrameHook** property; or **on stepMovie** movie handler

- **max**

Syntax: `max(list)`

or

`max (value1, value2, value3, ...)`

This function returns the highest value in the specified list, or the highest of a given series of values.

Example:

This handler assigns the variable `Winner` the maximum value in the list `Bids`, which consists of [`#Castle:600`, `#Schmitz:750`, `#Wang:230`]. The result is then inserted in the content of the field cast member `Congratulations`:

```
on findWinner Bids
  set Winner = max(Bids)
  set the text of member "Congratulations" =
    "You have won, with a bid of $" & Winner &"!"
end
```

● maxInteger

Syntax: the maxInteger

This property returns the largest whole number that is supported by the system. On most personal computers, this is 2,147,483,647 (2 to the 31st power, minus 1.)

This can be useful for initializing boundary variables before a loop or for limit testing.

Example:

This example 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
  put 31 into b
  put the maxInteger into v
  repeat while v > 0
    put b && "-" && v
    put b-1 into b
    put v/2 into v
  end repeat
end showMaxValues
```


• mci

Syntax: mci "*string* "

The multimedia extensions for Windows respond to commands sent to the media control interface, or mci. You can use the `mci` command to pass the strings specified by *strings* to the Windows media control interface.

Strings passed by the `mci` command play only in Windows; they are not executed on the Macintosh. Because the Macintosh does not support the mci interface, the `mci` command gives you a way to include commands intended for the Windows environment within a movie that you create and can play on the Macintosh.

Example:

This statement makes the command `play cdaudio from 200 to 600 track 7 play` only when the movie plays back in Windows:

```
mci "play cdaudio from 200 to 600 track 7"
```

- ◆ **mDescribe**

Syntax: *XObjectName* (mDescribe)

This predefined method is used only with. The purpose of `mDescribe` is to create a list of methods in the message window. This list contains the names of other methods of the XObject, plus any comments by the programmer of the XObject that document the functionality or syntax of these methods.

You only use this method for authoring. Do not include it in scripts within a movie.

Before using `mDescribe` to display an XObject's method, first open the appropriate library using the `openXlib` command. To display information about the XObject, enter the `showXlib` command followed by *XObjectName* (mDescribe) in the message window. A display of all open Xlibrary resource files and all XObjects is contained in those Xlibraries.

Example:

This statement displays methods and comments assigned to the `fileIO` XObject:

```
fileIO (mDescribe)
```

- ◆ **mMessageList** predefined method; **showXLib** command

- **mDispose**

Syntax: *object*(mDispose)

This predefined method supports XObjects in earlier versions of Lingo. It is recommended that you use lists and parent lists. They are a simpler way of achieving the same result.

This predefined method is used to destroy the object specified by *object*, which was created earlier with the `mNew` method. It is used to dispose of instances of XObjects. Use it to free up memory when an XObjects is no longer needed.

You do not need to explicitly dispose of child objects created from parent scripts. Lingo disposes of these objects when they are no longer referenced by a variable within the movie.

It is best that you check for previous instances of an object with the same name, and dispose of it before creating new instances of an object using `mNew`. The initialize handler in the following example illustrates this. In this way, if the movie is aborted before the normal `mDispose`, you won't fill up memory by repeatedly creating new objects. This can happen during the development of a project, when you repeatedly stop it before the end, and play it again from the beginning.

If you define an `mDispose` method in an XObject, it will be executed instead of the predefined method. The result--which is seldom what you want--is that the object will not really get disposed. If you need to perform various housekeeping actions before disposing, put the routines in a method with another name, like `mRelease`.

Example:

This handler determines whether the item assigned to the variable `myObject` is an XObject and disposes of it if it is:

```
on cleanUp
  global myObject
  if objectP(myObject) then myObject(mDispose)
end cleanUp
```

- **mNew** predefined method

- **me**

Syntax: me

This keyword can be used within parent scripts as a shorthand means of referring to the script itself.

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:

```
set myBird1 to new(script "Bird")
```

This is the `new` handler of the `Bird` script:

```
on new me
    return me
end
```

- **new** function; **ancestor** property. **parent scripts-child objects**

- **media of member**

Syntax: the media of member *whichCastmember*

This function returns data that describes the specified cast member. The result is a set of numbers that identifies the cast member.

You can use the `the media of member` to copy the content of one cast member into another cast member by setting the second cast member's `media of member` value to the `media of member` value for the first.

Example :

This statement copies the content of the cast member Sunrise into the cast member Dawn by setting the `media of member` value for Dawn to the `media of member` value for Sunrise:

```
set the media of member "Dawn" to the →  
media of member "Sunrise"
```

- **type of member**

• member

Syntax: `member whichCastmember`
 `member whichCastmember of castLib whichCast`

This keyword indicates that the object specified by *whichCastmember* is a cast member. If *whichCastmember* is a string, it is used as the cast member name. If *whichCastmember* is an integer, it is used as the cast member number.

Example 1:

The following statement sets the `hilite` of the button cast member named Enter Bid to TRUE:

```
set the hilite of member "Enter Bid" to TRUE
```

Example 2:

This statement puts the name of sound cast member 132 into the variable `soundName`:

```
put the name of member 132 into soundName
```

Example 3:

This statement determines whether cast member 9 has a name assigned.

```
if stringP(the name of member 9) then exit
```

- **memberNum of sprite**

Syntax: the memberNum of sprite *whichSprite*

This sprite property determines the number of the cast member associated with the sprite specified by *whichSprite*.

Setting this property lets you switch the cast member assigned to a sprite. The sprite should be a puppet before you do this. If it isn't puppeted, the sprite reverts to the cast member set in the score when the playback head exits the frame.

The value for the `memberNum of sprite` counts all cast member locations in all casts without separating out their cast numbers. It adds 1000 for each lower numbered cast. For example, the `memberNum` for cast member 1 in cast 1 is 1. However, the `memberNum` for cast member 1 of cast 2 is 1001. You can avoid confusion by specifying the `member` for the `sprite` and specifying a cast member and its cast. For example, this statement assigns cast member Tree in cast Landscape to sprite 5:

```
set the member of sprite 5 to member "Tree" of castLib "Landscape"
```

You could also use the same approach but use cast member numbers, as in the following example:

```
set the member of sprite 5 to member 10 of castLib 3
```

A typical use of this is 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. If you are changing several sprite properties--or several puppet sprites--you only have to use one `updateStage` command after making all of the changes.

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

Example 1:

The following statement switches the cast member assigned to sprite 3 to cast member number 35:

```
set the memberNum of sprite 3 to 35
```

Example 2:

This statement assigns the cast member Narrator to sprite 10 by setting the `memberNum of sprite` to Narrator's cast number:

```
set the memberNum of sprite 10 = the number of member "Narrator"
```

- **number of member property**

- **members**

- the number of members property

- **memberType of member**
- type of member cast property

- **memorySize**

Syntax: the memorySize

This function returns the total amount of memory (in bytes) allocated to the program, whether in use or free. It is useful for checking minimum memory requirements. A kilobyte (K) is 1024 bytes. A megabyte (MB) is 1024K.

Example:

```
if the memorySize < 500 * 1024 then alert -  
    "There is not enough memory to run this movie."
```

- **freeBlock** and **freeBytes**, and **ramNeeded** functions; **size of member** cast property

● menu

Syntax: *menu: menuName*
 itemName | script
 itemName | script
 ...

 or

menu: menuName
 itemName | script
 itemName | script
 ...
 [*more menus*]

This keyword specifies the actual content of custom menus, in conjunction with the `installMenu` command. Menu definitions are typed in field cast members. You refer to a particular menu definition by its cast member name or number.

The `menu:` keyword specifies the name of the menu. In the subsequent lines you can specify the menu items for that menu. You can have a script execute when the user chooses that item by putting the script after the "or" ("|") symbol. A new menu is defined by the subsequent occurrence of the `menu:` keyword.

You can create hierarchical menus by using XCMDs or simulate them by writing scripts that display a graphic cast member that mimics a submenu.

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.)

Symbol	Example--Description (key combination)
	Open/O go to frame "Open"--Associates a script with the menu item
@	menu: @--Creates the Apple symbol and enables Macintosh menu bar items when you define an Apple menu on the Macintosh
(Save (--Disables the menu item
(-	(- --Creates a disabled line in the menu
!☐	!☐Easy Select--Checks the menu with a checkmark (Option-v)
<B	Bold<B--Sets the menu item's style to Bold
<I	Italic<I--Sets the style to Italic
<U	Underline<U--Sets the style to Underline
<O	Outline<O--Sets the style to Outline
<S	Shadow<S--Sets the style to Shadow

/ Quit/Q--Defines a command-key equivalent

Special symbols should follow the item name, and precede the "¼" symbol. You can also use more than one special character to define a menu item. Using <B<U, for example, sets the style to Bold and Underline.

Example:

This set of statements specifies the content of a custom File menu:

```
menu: File
Open/O | go to frame "Open"
Close/W | go to frame "Close"
(-
Quit/Q | go to frame "Quit"
```

- **installMenu** command; **name of menu** menu property; **name of menutem,** **number of menutems,** **checkMark of menutem,** **enabled of menutem,** and **script of menutem** menu item properties

- **menuItem**

See name of menuItem, number of menuItems, checkMark of menuItem, enabled of menuItem, and script of menuItem menu item properties.

- **method**

Syntax: `method methodName [argument1] [, argument2] ...`

This keyword is now obsolete. It was used to define a method, which was used in earlier versions of Director and is now used only for compatibility with scripts written in earlier movies. You should use lists and parent scripts in Director 5.0; they are a simpler way to achieve the same result.

A method is a special kind of handler that exists inside an XObject and that has its own special syntax. It uses Lingo to create expressions that are commands or functions. A method is a script, or series of scripts, that handle different messages (or processes) for objects created by an XObject.

There are two kinds of objects: internal (created by factories) and external (created by XObjects). Factories and XObjects use methods. The difference is that you define an XObject's methods in the movie script or a cast member script, but an XObject's methods are predefined in the XObject itself. To see an XObject's methods, type `XObjectName(mDescribe)` in the message window.

Each object has its own set of messages created by its methods. Messages are the way objects communicate with each other and with the rest of Lingo. Messages are sent by an object's methods and provide all of the necessary functionality for each particular object's task. Methods are associated with the objects created by their XObject. Each object can use all the methods in its XObject.

A method is defined using the `method` keyword:

```
method messagename
```

For ease of reference, it is a good convention to begin the value you substitute for *messagename* with a lowercase m.

- **exit** and **return** keywords; **mNew** predefined method

- **mGet**

Syntax: *object*(mGet, *whichElement*)

Methods were used for managing arrays in earlier versions of Director. It is now recommended that you use lists and parent lists instead. They are a simpler way of achieving the same result.

The integer expression *whichElement* specifies which array element the mGet method returns. If you retrieve an element that has not been assigned a value with the mPut method, the element has the numerical value 0.

Different types of data can be stored in various elements of the same array. You can use the functions floatP, integerP, objectP, stringP, and symbolP to determine the data type of a particular element.

Example:

These first three statements use mPut to put data into an internal array. Using 3, 7, and 12 assigns these values to the third, seventh, and twelfth elements of the array:

```
put objectName (mNew) into myObject
myObject(mPut, 3, 2 + 2)
myObject(mPut, 7, sqrt(2.0))
myObject(mPut, 12, "hello" && "there")
```

This statement displays the value associated with the third element of the array:

```
put myObject(mGet, 3)
```

The result is 4, which is the equivalent of 2 + 2.

This statement displays the value associated with the seventh element of the array:

```
put myObject(mGet, 7)
```

The result is 1.4142, which is the equivalent of the square root of 2.

This statement displays the value associated with the twelfth element of the array:

```
put myObject(mGet, 12)
```

The result is "hello there", which is the value that was assigned in the first example.

Note: Different types of data can be stored in various elements of the same array. You can use the functions integerP, floatP, stringP, symbolP, and objectP to determine the data type of a particular element.

- **mPut** predefined method

- **min**

Syntax: `min(list)`

or

`min(a1, a2, a3...)`

This function specifies the minimum value in the list specified by *list*.

Example:

This handler assigns the variable `vLowest` the minimum value in the list `bids`, which consists of [`#Castle:600`, `#Shields:750`, `#Wang:230`]. The result is then inserted in the content of the field cast member `Sorry`:

```
on findLowest bids
  set vLowest = min(bids)
  set the text of member "Sorry" = -
    "We're sorry, your bid of $" & vLowest && "is not a
    winner!"
end
```

- **max** function

- **mInstanceRespondsTo**

Syntax: *XObject*(mInstanceRespondsTo, *message*)

This predefined method can only be used with XObjects. It returns a positive integer if an instance of the XObject responds to the specified message, which must be a string or symbol expression. In this case the integer returned is the number of arguments required by the message, plus 1. The method returns 0 if XObject does not respond to the specified message.

Example:

This statement checks whether the SerialPort XObject responds to the message string mWrite.

```
put SerialPort(mInstanceRespondsTo, "mWrite")
```

The result is 2; one for the first parameter, plus one.

- **mRespondsTo** predefined method

- ◆ **mMessageList**

Syntax: *XObject* (mMessageList)

This predefined method can only be used with XObjects. It returns a string that describes the XObject and its methods. The string is the same string that the `mDescribe` method displays in the message window; however, it may be put into fields or variables.

Example:

This statement displays methods and comments assigned to the fileIO XObject:

```
put fileIO(mMessageList)
```

- ◆ **mDescribe, mInstanceRespondsTo, mRespondsTo** predefined methods

- **mName**

Syntax: `XObject(mName)`

or

`XObjectInstance(mName)`

This predefined method (which can be used only with XObjects and their instances) gives a string that contains the name of the XObject that created the instance.

Example:

These statements create an instance of the serial port XObject and places it in the variable `modemPort`. It then displays the name of the XObject instance:

```
put SerialPort(mNew, 0) into modemPort
put modemPort(mName)
```

The result is `SerialPort`, which is the name of the XObject.

● mNew

Syntax: XObject(mNew [,argument1] [,argument2] ...)

This predefined method is used to create objects or instances of an external XObject in RAM. To create the instance of a particular class of objects, you assign an object variable to the particular XObject name using the `mNew` method.

Arguments to the `mNew` method are optional. Of course, a particular XObject may have been written to require a certain number of arguments of a certain type. See its documentation, its example movie, or its `mDescribe` in the message window for this information.

There is no requirement for any particular number of arguments to the `mNew` method of objects. Typically you use the `mNew` method to assign instance variables used throughout the methods of an object.

In order to clear the object you create using `mNew` from RAM at the end of the movie, it is a good idea to use the predefined `mDispose` method for XObjects.

Before creating new instances of an object using `mNew`, it is also a good idea to check for previous instances of an object that has the same name, and `mDispose` it before you create a new one. In this way, if the movie is aborted before the normal `mDispose`, you won't fill up RAM by repeatedly creating new ones. This can happen during the development of a project, when you repeatedly stop the movie before the end, and play it again from the beginning.

Examples:

This statement creates a new instance of `myArrayObject`. The new instance is named `myArray`:

```
put myArrayObject(mNew) into myArray
```

This statement creates a new instance of `birdObj`. The new instance is named `bird` and has initial instance variables `wingCastNum` and `legCastNum`:

```
put birdObj(mNew, wingCastNum, legCastNum) into bird
```

This statement creates a new instance of the XObject `PioneerLaserDisc` `myArrayObject`. The new instance is named `vDisc`:

```
put PioneerLaserDisc(mNew, 1, 9600, 0) into vDisc
```

This handler checks for existing instances of factories and XObjects and disposes of any it finds. It then creates a new instance of the `myArrayObject`:

```
on startMovie
  global myObject
  -- check for previous instances:
  if objectP(myObject) then myObject(mDispose)
  -- create a new instance of the object in RAM:
  put myArrayObject(mNew) into myObject
```

end startMovie

- **method** keyword; **mDescribe** and **mDispose** predefined methods

• mod

Syntax: *integerExpression1* mod *integerExpression2*

This arithmetic operator performs the arithmetic modulus operation on two integer expressions. In this operation, *integerExpression1* is divided by *integerExpression2*. The resulting value of the entire expression is the integer remainder of the division.

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

Examples:

This statement divides 7 by 4 and then displays the remainder in the message window:

```
put 7 mod 4
```

The result is 3.

This handler sets the ink effect of all odd-numbered sprites to copy, which is the ink effect specified by the number 0. First, the handler checks whether the sprite that has the number in the variable `mySprite` is an odd-numbered sprite by dividing the sprite number by 2 and then checking whether the remainder is 1. When the remainder is 1, which is the result for an odd-numbered number, the ink effect is set to copy:

```
on setInk
  if (mySprite mod 2) = 1 then
    set the ink of sprite mySprite to 0
  else
    set the ink of sprite mySprite to 8
  end if
end setInk
```

- **modal of window**

Syntax: `the modal of window "window "`

This window property specifies whether movies can respond to events that occur outside the window specified by *window*.

- When the `modal of window` property is TRUE, movies cannot respond to events outside the window.
- When the `modal of window` property is FALSE, movies can respond to events outside the window.

Setting the `modal of window` to TRUE lets you make a specific movie in a window the only movie that the user can interact with.

Example:

This statement lets movies respond to events outside of the window Tool Panel:

```
set the modal of window "Tool Panel" to FALSE
```


- **modified of member**

Syntax: `the modified of member whichCastmember`

This function indicates whether the cast member specified by *whichCastmember* has been modified since it was read from the movie file.

- When `the modified of member` is TRUE (1), the cast member has been modified since it was read from the movie file.
- When `the modified of member` is FALSE (0), the cast member has not been modified since it was read from the movie file.

Example:

This statement tests whether the cast member Introduction has been modified since it was read from the movie file:

```
put the modified of member "Introduction"
```

The result is 0, which is the numerical equivalent of FALSE.

- **mouseCast**

Syntax: the mouseCast

This integer function gives the cast number of the sprite that is under the cursor when the function is called. When the cursor is not over a cast member, it gives the result -1.

This is useful for having the movie perform specific actions when the cursor rolls over a sprite and the sprite uses a certain cast member.

Examples:

This statement checks whether the cast member Off Limits is the cast member assigned to the sprite under the cursor and displays an alert if it is. This is one example of how you can specify an action depending on which cast member is assigned to the sprite:

```
if the mouseCast = the number of member "Off Limits"→  
then alert "Stay away from there!"
```

This statement assigns the number of the sprite under the cursor to the variable lastCast:

```
put the mouseCast into lastCast
```

- **memberNum of sprite** sprite property; **mouseChar**, **mouseItem**, **mouseLine**, **mouseWord**, and **rollOver** functions; the **number of member** cast property

- **mouseChar**

Syntax: the mouseChar

This integer function, used for field sprites, gives the number of the character that is under the cursor when the function is called. The count is from the beginning of the field. If the mouse is not over a field or is in the gutter of a field, the result is -1.

Example:

This statement determines whether the cursor is not over a field sprite and changes the content of the field cast member Instructions to "Please point to a character." when it is:

```
if the mouseChar = -1 then ↵
  put "Please point to a character." ↵
  into field "Instructions"
```

This statement assigns the character under the cursor in the specified field to the variable `currentChar`:

```
put char (the mouseChar) of member (the mouseCast) ↵
  into currentChar
```

- **mouseItem**, **mouseLine** and **mouseWord** functions; **char...of** chunk expression keyword; the **number of chars in** chunk function

- **mouseDown**

Syntax: the mouseDown

This function indicates whether the mouse button is currently being pressed.

- When the mouseDown is TRUE, the button is being pressed.
- When the mouseDown is FALSE, the button is not being pressed.

Examples:

This handler has the movie beep until the user clicks the mouse:

```
on enterFrame
  repeat while the mouseDown = FALSE
    beep
  end repeat
end
```

This statement has Lingo exit the repeat loop or handler it is in when the user clicks the mouse:

```
if the mouseDown then exit
```

- mouseH, mouseUp, and mouseV functions; on mouseDown and on mouseUp event handlers

- **mouseDownScript**

Syntax: the mouseDownScript

This property specifies the Lingo that is executed when the mouse button is pressed. The Lingo can be a simple statement or a calling script for a handler.

When the mouse button is pressed and the `mouseDownScript` is defined, Lingo executes the instructions specified for the `mouseDownScript` first. Unless the instructions include the `pass` command so that the `mouseDown` message can pass on to other objects in the movie, no other `on mouseDown` handlers are executed.

Setting the `mouseDownScript` property does the same thing as using the `when keyDown then` command that appeared in earlier versions of Director.

When the instructions you've specified for the `mouseDownScript` property are no longer appropriate, turn them off by using the statement `set the mouseDownScript to empty`.

The `mouseDownScript` property can be tested and set, and the default value is `EMPTY`, which means that the `mouseDownScript` has no Lingo at all assigned to it.

Example:

This statement sets the `mouseDownScript` to `if the mouseDown then go to next`. When this is in effect and the user clicks the mouse button, the playback head always jumps to the next marker in the movie.

```
set the mouseDownScript -  
    to "if the mouseDown then go to next"
```

This statement sets the `mouseDownScript` to `if the clickOn = 0 then beep`. When this is in effect and the user clicks anywhere on the stage, the computer beeps.

```
set the mouseDownScript -  
    to "if the clickOn = 0 then beep"
```

- **dontPassEvent** command; **mouseUpScript** property; **on mouseDown** and **on mouseUp** event handlers

- **mouseH**

Syntax: the mouseH

This function indicates the horizontal position of the mouse cursor. The value of `mouseH` is the number of pixels the cursor is from the left edge of the stage.

The `mouseH` function is useful for moving sprites to the horizontal position of the mouse cursor and checking whether the cursor is within a region of the stage. Using `mouseH` and `mouseV` functions together, you can determine the cursor's exact location.

The `mouseH` function can be tested but not set.

Examples:

This handler moves sprite 10 to the mouse cursor location and updates the stage when the user clicks the mouse button:

```
on mouseDown
  set the locH of sprite 1 to the mouseH
  set the locV of sprite 1 to the mouseV
  updateStage
end
```

This statement tests whether the cursor is more than 10 pixels to the right or left of a starting point and sets the variable `Far` to `TRUE` if it is:

```
if abs(the mouseH - startH) > 10 then ¬
  put TRUE into draggedEnough
```

- **locH of sprite** and **locV of sprite** sprite properties; **mouseV** function

- **mouseItem**

Syntax: the mouseItem

This integer function gives the number of the item that is under the pointer when the function is called and the cursor is over a field sprite. (An item is any sequence of characters delimited by commas.) Counting starts at the beginning of the field. If the mouse is not over a field, the result is -1.

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 ↵  
  put "Please point to an item." ↵  
  into field "Instructions"
```

This statement assigns the item under the cursor in the specified field to the variable currentItem:

```
put item (the mouseItem) of member (the mouseCast) ↵  
  into currentItem
```

- **item...of** chunk expression keyword; **mouseChar**, **mouseLine**, and **mouseWord** functions; **number of items in** chunk function

- **mouseLine**

Syntax: the mouseLine

This integer function gives the number of the line under the pointer when the function is called and the cursor is over a field sprite. Counting starts at the beginning of the field. When the mouse is not over a field sprite, the result is -1.

Examples:

This statement determines whether the cursor is over a field sprite and changes the content of the field cast member Instructions to "Please point to a line." when it is not:

```
if the mouseLine = -1 then →  
  put "Please point to a line." →  
  into field "Instructions"
```

This statement assigns the number of the item under the cursor in the specified field to the variable `currentLine`:

```
put line (the mouseLine) of member (the mouseCast) →  
  into currentLine
```

- **mouseChar**, **mouseItem**, and **mouseWord** functions; **line...of** chunk expression keyword; **number of lines in** chunk function

- **mouseUp**

Syntax: the mouseUp

This function indicates whether the mouse button is released.

- The mouseUp function is TRUE when the mouse button is released.
- The mouseUp function is FALSE when the mouse button is being pressed.

Examples:

This handler has the movie beep as long as the mouse button is being pressed. The beep stops when the user clicks the mouse button:

```
on enterFrame
  repeat while the mouseUp = FALSE
    beep
  end repeat
end enterFrame
```

This statement has Lingo exit the repeat loop or handler it is in when the user releases the mouse button:

```
if the mouseUp then exit
```

- **mouseDown**, **mouseH**, and **mouseV** functions; **on mouseDown** and **on mouseUp** event handlers

- **mouseUpScript**

Syntax: the mouseUpScript

This property determines the Lingo that is executed when the mouse button is released. The Lingo can be a simple statement or a calling script for a handler.

When the mouse button is released and the `mouseUpScript` is defined, Lingo executes the instructions specified for the `mouseUpScript` first. Unless the instructions include the `pass` command so that the `mouseUp` message can pass on to other objects in the movie, no other `mouseUp` handlers are executed.

When the instructions you've specified for the `mouseUpScript` property are no longer appropriate, turn them off by using the statement `set the mouseUpScript to empty`.

Setting the `mouseUpScript` property does the same as using the `when mouseUp then` command that appeared in earlier versions of Director.

The `mouseUpScript` property can be tested and set. The default value is `EMPTY`.

Examples:

This statement sets the `mouseUpScript` to `continue`. When this is in effect and the movie is paused, the movie always continues whenever the user releases the mouse button.

```
set the mouseUpScript to "continue"
```

This statement has the movie beep when the user releases the mouse button after clicking anywhere on the stage:

```
set the mouseUpScript -  
to "if the clickOn = 0 then beep"
```

- **dontPassEvent** command; **mouseDownScript** property; **on mouseDown** and **on mouseUp** event handlers

- **mouseV**

Syntax: the mouseV

This function indicates the vertical position of the mouse cursor. The value of `mouseV` is the number of pixels the cursor is from the top of the stage.

The `mouseV` function is useful for moving sprites to the vertical position of the mouse cursor and checking whether the cursor is within a region of the stage. Using `mouseH` and `mouseV` functions together, you can identify the cursor's exact location.

Examples:

This handler moves sprite 1 to the mouse cursor location and updates the stage when the user clicks the mouse button:

```
on mouseDown
  set the locH of sprite 1 to the mouseH
  set the locV of sprite 1 to the mouseV
  updateStage
end
```

This statement tests whether the cursor is more than 10 pixels above or below a starting point and sets the variable `vFar` to TRUE if it is:

```
if abs(the mouseV - startV) > 10 then →
  put TRUE into draggedEnough
```

- **mouseH** function; **locH of sprite** and **locV of sprite** sprite properties

- **mouseWord**

Syntax: the mouseWord

This integer function gives the number of the word under the cursor when the function is called and when the cursor is over a field sprite. Counting starts from the beginning of the field. When the mouse is not over a field, the result is -1.

Examples:

This statement determines whether the cursor is over a field sprite and changes the content of the field cast member Instructions to "Please point to a word." when it is not:

```
if the mouseWord = -1 then ↵
  put "Please point to a word." ↵
  into member "Instructions"
```

This statement assigns the number of the word under the cursor in the specified field to the variable `currentWord`:

```
put word (the mouseWord) of member (the mouseCast) ↵
  into currentWord
```

- **mouseChar**, **mouseItem**, and **mouseLine** functions; **number of words in** chunk function; **word...of** chunk expression keyword

- **move member**

Syntax: `move member whichCastmember [, member whichLocation]`

This command moves the cast member specified by *whichCastmember* to a different location in the cast window.

- Using the `move member` command without the optional parameter, the cast member moves to the first empty location in the cast window.
- Including the `member whichLocation` parameter in the `move member` command moves the cast member to the location specified by *whichLocation*.

Example:

This statement moves cast member Shrine to the first empty location in the cast window:

```
move member "Shrine"
```

This statement moves cast member Shrine to location 20 in the Bitmaps cast window:

```
move member "Shrine", member 20 of castLib "Bitmaps"
```

- **moveableSprite of sprite**

Syntax: the moveableSprite of sprite *whichSprite*

This sprite property indicates whether a sprite is moveable.

- When the sprite can be moved by the user, the moveableSprite of sprite is TRUE (1).
- When the sprite cannot be moved by the user, the moveableSprite of sprite is FALSE (0).

To use Lingo to make a field sprite moveable, the sprite must first be a puppet sprite.

You can also make a sprite moveable by using the Moveable option in the score. However, controlling whether a sprite is moveable by using Lingo lets you turn this condition on and off as situations in the movie require. For example, referring to the "Mechanical Simulation" sample movie, you could let the user drag parts from the toolkit but make them unmoveable after they are on the pegboard by turning moveableSprite of sprite on and off at the appropriate times.

Setting the moveableSprite of sprite property lets you control whether sprites are moveable from other scripts.

The moveableSprite of sprite property can be tested and set.

Examples:

This handler first makes the sprite a puppet and then makes it moveable:

```
on spriteMove
  puppetSprite 5, TRUE
  set the moveableSprite of sprite 5 to TRUE
end
```

This statement checks whether a sprite is moveable and displays a message if it isn't:

```
if the moveableSprite of sprite 13 = FALSE →
  then set the text of member "Notice" to →
  "You can't drag this item by using the mouse."
```

- **puppetSprite** command

● **moveToBack**

Syntax: `moveToBack window "whichWindow "`

This command moves the window specified by *whichWindow* behind all other windows.

Example:

These statements move the first window in the `windowList` behind all other windows:

```
set myWind=getat(the windowList, 1)
moveToBack myWind
```

If the first record of the `windowList` was "Demo Window", the long version of the `moveToBack` would be:

```
moveToBack window "Demo Window"
```

• **moveToFront**

Syntax: `moveToFront window "whichWindow"`

This command moves the window specified by *whichWindow* in front of all other windows.

Example:

This statement moves the first window in the `windowList` in front of all other windows:

```
set myWind=getat(the windowList, 1)
moveToFront myWind
```

If the first record of the `windowList` was "Demo Window", the long version of the `moveToFront` would be:

```
moveToFront window "Demo Window"
```


- **movie**

Syntax: the movie

This string function returns the name of the currently open movie.

Example:

This statement assigns the name of the current movie to the field cast member Current Movie:

```
put the movie into member "Movie Name"
```

- **go** and **play** commands; **pathName** function

● **movieFileFreeSize**

Syntax: `the movieFileFreeSize`

This function returns the number of unused bytes in the current movie.

When the movie has no unused space, the `movieFileFreeSize` function returns 0.

Example:

This statement displays the number of unused bytes that are in the current movie:

```
put the movieFileFreeSize
```

- **movieFileSize**

Syntax: `the movieFileSize`

This function returns the number of bytes in the current movie.

Example:

This statement displays the number of bytes in the current movie:

```
put the movieFileSize
```

- **movieName**

Syntax: the movieName

This function indicates the simple name of the current movie. The `movieName` function is equivalent to the `movie` function.

Example:

This statement displays the name of the current movie in the message window:

```
put the movieName
```

- **movie**, **moviePath**, and **pathName** functions

- **moviePath**

Syntax: the moviePath

This function indicates the pathname of the folder that the current movie is located in. The `moviePath` function is equivalent to the `pathName` function.

Example 1:

This statement displays the pathname of the current movie's folder:

```
put the moviePath
```

Example 2:

This statement plays a sound file "crash.aif" stored in the "sounds" subfolder of the current movie's folder:

```
sound playFile 1, the moviePath&"sounds/crash.aif"
```

Note: If you choose to specify a subfolder location, as in this example, using "/" will insure that the path is understood on both Macintosh and Windows computers. Only on a Macintosh can you use ":" to separate subfolders.

- **movie, movieName, and pathName** functions

● **movieRate of sprite**

Syntax: `the movieRate of sprite channelNumber`

This sprite property controls the rate at which a digital video in a specific channel plays. The movie rate is a value specifying the playback of the digital video. A value of 1 is normal forward play, -1 is reverse, 0 is stop. Higher and lower values are possible. For example a value of 0.5 has the digital video play slower than normal. However, frames may be dropped when the `movieRate of sprite` exceeds 1. The severity of dropping frames depends on factors such as the performance of the computer the movie is playing on, whether the digital video sprite is stretched, and so on.

This property can be tested and set.

Example:

This statement sets the rate for a digital video in sprite channel 9 to normal playback speed:

```
set the movieRate of sprite 9 to 1
```

This statement has the digital video in sprite channel 9 play in reverse:

```
set the movieRate of sprite 9 to -1
```

● **movieTime of sprite**

Syntax: the movieTime of sprite *channelNumber*

This sprite property determines the current time of a digital video movie playing in the channel specified by *channelNumber*. The value of the `movieTime` is measured in ticks.

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

Example:

This statement displays the current time of the QuickTime movie in channel 9 in the message window:

```
put the movieTime of sprite 9
```

This statement sets the current time of the QuickTime movie in channel 9 to the value in the variable `Poster`:

```
set the movieTime of sprite 9 to Poster
```

- **mPerform**

Syntax: `object (mPerform, message [, argument1] [, argument2] ...)`

This predefined method is used to send an arbitrary message to any Lingo object. It is similar to the Lingo `do` command, which executes a Lingo statement stored as a string. However, `mPerform` invokes a particular method of the specified object by sending that message to the object indirectly.

This is accomplished as follows: The first argument to `mPerform` is a required argument called a "message expression." This expression can be either in the form of either a string or symbol. This message specifies the name of the method to be invoked by the `mPerform` message.

Optional additional arguments, which can be any data type, constant, or property used in the method to be invoked, follow this required first argument.

Typically, the object name is specified by use of the `me` keyword, since the typical use of `mPerform` is within a method that invokes one of several other methods.

A powerful use for `mPerform` is to eliminate a lot of `if...then` conditional tests within methods that call other methods.

Examples:

This statement creates an instance named `modemPort` of the `SerialPort` XObject:

```
put SerialPort(mNew, 0) into modemPort
```

These statements invoke the `mWriteChar` method with the argument `charNum`:

```
modemPort(mPerform, "mWriteChar", charNum)
modemPort(mWriteChar, charNum)
```

- **me and method keywords**

- **mPut**

Syntax: `object(mPut, whichElement, expression)`

This predefined method, which can only be used with objects, puts data into an object's internal array. Every object produced has an associated array capable of storing an arbitrary number of integers, floating point numbers, strings, objects, or symbols. The elements of the array are numbered 1, 2, 3, The `mGet` predefined method is used to retrieve values from a particular element.

The integer expression *whichElement* specifies which array element the `mPut` method assigns. The value of *expression* is assigned to the specified element.

Example:

These three statements use `mPut` to put data into an internal array. Using 3, 7, and 12 assigns these values to the third, seventh, twelfth elements of the array:

```
myObject(mPut, 3, 2 + 2)
myObject(mPut, 7, sqrt(2.0))
myObject(mPut, 12, "hello" && "there")
```

This statement displays the value associated with the third element of the array:

```
put myObject(mGet, 3)
```

The result is 4.

This statement displays the value associated with the seventh element of the array:

```
put myObject(mGet, 7)
```

The result is 1.4142, which is the square root of 2.

This statement displays the value associated with the twelfth element of the array:

```
put myObject(mGet, 12)
```

The result is the string "hello there".

- **mGet** predefined method

- **mRespondsTo**

Syntax: *XObjectInstance* (mRespondsTo, *message*)

This predefined method, which can only be used with instances of XObjects, returns a positive integer when *XObjectInstance* responds to the specified message, which must be a string or symbol expression. In this case the integer returned is the number of arguments required by the message, plus 1. The method returns 0 if *XObjectInstance* does not respond to the specified message.

Example:

These statements create an instance of the XObject SerialPort and checks whether it responds to the message string `mWrite`.

```
put SerialPort(mNew, 0) into modemPort
put modemPort(mRespondsTo, "mWrite")
```

- **mInstanceRespondsTo** predefined method

• multiSound

Syntax: the multiSound

This system property is TRUE when the system supports more than one sound channel. (A PC must have a multichannel sound card for the multiSound property to be TRUE.)

Example:

This statement plays the sound file Music in sound channel 2 if the computer supports more than one sound channel:

```
if the multiSound then sound playFile 2, "Music"
```

- **name of member**

Syntax: the name of member *whichCastmember*

This cast member property determines the name of the specified cast member.

- When *whichCastmember* evaluates to a string, it is used as the cast name.
- When *whichCastmember* evaluates to an integer, it is used as the cast number.

The name is a descriptive string assigned by the user. Setting this property is equivalent to entering a name in the **Cast Member Properties** dialog box.

The `name` cast member property can be tested and set.

Examples:

This statement changes the name of cast member named On to Off:

```
set the name of member "On" to "Off"
```

This statement sets the name of cast member 15 to Background Sound:

```
set the name of member 15 to "Background Sound"
```

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`:

```
put the name of member (i + 1) into itsName
```

- **number of member** cast property, **Cast Member Properties** dialog box

- **name of CastLib**

Syntax: the name of castLib *whichCast*

This cast member property returns the name of the specified cast. This property can be tested and set.

Example:

This statement changes the name of the cast Buttons to Interface:

```
set the name of castLib "Buttons" to "Interface"
```

- **name of menu**

Syntax: the name of menu *whichMenu*

This menu property returns a string containing the name of the specified menu. The expression *whichMenu* can evaluate to either a menu number or a menu name.

The `name of menu` property can be tested but cannot be set directly. Use the `installMenu` command to set up a custom menu bar.

Example:

This statement assigns the name of menu number 1 to the variable `firstMenu`:

```
put the name of menu 1 into firstMenu
```

The following handler returns a list of menu names, one per line:

```
on menuList
  put EMPTY into list
  repeat with i = 1 to the number of menus
    put the name of menu i & RETURN after list
  end repeat
  return list
end menuList
```

- **number of menus** property; **name of menuitem** menu item property

- **name of menuItem**

Syntax: the name of menuItem *whichItem* of menu *whichMenu*

This menu item property determines the text that appears in the menu item specified by *whichItem* in the menu specified by *whichMenu*. The *whichItem* expression can be either a menu item name or a menu item number; *whichMenu* can be either a menu name or a menu number.

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

Examples:

This statement sets the variable `itemName` to the name of the eighth item in the Edit menu:

```
put the name of menuItem 8 of menu "Edit" -  
into itemName
```

This statement has a specific filename follow the term Open in the File menu:

```
set the name of menuItem "Open" of menu fileMenu -  
to "Open" & fileName
```

- **name of menu** and **number of menuItems** properties

- **name of window**

Syntax: the name of window "*whichWindow*"

This property determines the name of the specified window. It can be tested and set.

Example:

This statement changes the name of window Yesterday to Today

```
set the name of window "Yesterday to "Today"
```


- **name of xtra**

Syntax: the name of xtra *whichXtra*.

This property indicates the name of the specified Xtra. It can be tested and set.

Example:

The following statement changes the name of the Xtra Editor to Text Whiz:

set the name of xtra "Editor" to "Text Whiz"

- **Using Xtras**

● new

Syntax: `new (type)`

or

`new (type, castLib whichCast)`

or

`new (type, member whichCastMember of castLib whichCast)`

or

`set x = new (parentScript arg1, arg2, ...)`

or

`new (script parentScriptName , value1, value2 , ...)`

This function creates a new cast member or child object.

For cast members, the parameter `type` sets the cast member's type. Possible predefined values correspond to the existing cast member types: `#bitmap`, `#field`, etc. The `new` function can also create Xtra cast member types, which can be identified by any name that the author chooses.

The optional `whichCastMember` and `whichCast` parameters specify the cast member slot and cast window where the new cast member is stored. When no cast member slot is specified, the first empty slot is used. The `new` function returns the cast member slot.

When the argument for the `new` function is a parent script, the `new` function creates a child object. The parent script should include an `on new` handler that sets the child object's initial conditions.

The child object has all the handlers of the parent script. The child object has the same property variable names that are declared in the parent script, but each child object has its own values for these properties.

Because the child object is a value, it can be assigned to variables, placed in lists, and passed as a parameter.

Being able to assign individual property values to child objects is the primary advantage of using `on new` handlers.

You display information about a child object by using the `put` command to display information about it in the message window.

For more information about creating child objects from parent scripts, see Chapter 10, "Parent Scripts & Child Objects," in *Learning Lingo*.

Example 1:

This handler creates a new bitmap cast member and assigns it to the variable Background:

```
on makeBitmap
  set Background = new(#bitmap)
end makeBitmap
```

Example 2:

These statements use a `new` handler to create a child object of a parent script. The parent script is a script cast member named Bird, which contains these handlers:

```
on new me
  return me
end

on fly me
  put "I am flying"
end fly
```

Example 3:

These statements create a child object called `myBird`, and make it fly by calling the fly handler in the Bird parent script:

```
set myBird to new(script "Bird")
fly myBird
```

Example 4:

This example uses a new Bird parent script, which contains the property variable speed:

```
property speed

on new me, initSpeed
  set speed to initSpeed
  return me
end

on fly me
  put "I am flying at " & speed & "mph"
end
```

Example 5:

The following statements create two child objects called `myBird1` and `myBird2`. When the fly handler is called from the child object, the speed of the object is displayed in the message window:

```
set myBird1 to new (script "Bird", 15)
set myBird2 to new(script "Bird", 25)
fly myBird1
```

```
fly myBird2
```

This message would appear in the message window:

```
-- "I am flying at 15 mph"
```

```
-- "I am flying at 25 mph"
```

- **type of member cast member property, ancestor, me, parent-child scripts**

- **next**

Syntax: next

This keyword refers to the next marker in the movie. The `next` keyword is equivalent to the phrase `the marker (+ 1)`.

Example:

This statement sends the playback head to the next marker in the movie:

```
go next
```

- **loop** and **previous** keywords

- **next repeat**

Syntax: next repeat

This keyword causes Lingo to go to the next step in a repeat loop in a script. This is different from the `exit repeat` keyword.

Example:

This repeat loop displays only odd numbers in the message window:

```
repeat with i = 1 to 10
    if (i mod 2) = 0 then next repeat
    put i
end repeat
```

- **not**

Syntax: `not logicalExpression`

This logical operator performs a logical negation on a logical expression.

- When the expression specified by *logicalExpression* is TRUE, the result is FALSE (0).
- When the expression specified by *logicalExpression* is FALSE, the result is TRUE (1).

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

Examples:

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.

This statement determines whether 1 is not greater than 2:

```
put not (1 > 2)
```

Because 1 is not greater than 2, the result is 1, which indicates that the expression is TRUE.

This handler sets the `checkMark` of `menuItem` for the item Bold in the Style menu to the opposite of its current setting:

```
on resetMenuItem
  set the checkMark of menuItem "Bold" ¬
    of menu "Style" to not (the checkMark ¬
      of menuItem "Bold" of menu "Style")
end resetMenuItem
```

- **and** and **or** logical operators

- **nothing**

Syntax: nothing

This command does nothing at all. It is useful for making the logic of an if-then statement more obvious. Also, a nested `if...then...else` statement that contains no explicit command for the `else` clause may require `else nothing`. Otherwise, Lingo interprets the `else` clause as part of the preceding `if`.

Examples:

The nested `if-then-else` statement in this handler uses the `nothing` command to satisfy the statement's `else` clause:

```
on mouseDown
  if the clickOn = 1 then
    if the moveable of sprite 1 = TRUE ↵
      then set the text of member "Notice" = ↵
        "Drag the ball"
    else nothing
    else set the text of member "Notice" = ↵
      "Click again"
    end if
  end mouseDown
```

This handler has the movie do nothing as long as the mouse button is being pressed:

```
on mouseDown
  repeat while the stillDown
    nothing
  end repeat
end mouseDown
```

- **if...then** keywords

- **number of member**

Syntax: the number of member *whichCastmember*

This cast member property indicates the cast number of the cast member specified by *whichCastmember*.

- When *whichCastmember* is a string, the string is used as the cast member name.
- When *whichCastmember* is an integer, the integer is used as the cast member number.

The `number of member` property can be tested, but not set.

Examples:

This statement assigns the cast number of the cast member Power Switch to the variable `whichCastmember`:

```
put the number of member "Power Switch" into -  
    whichCastmember
```

This statement assigns the cast member Red Balloon to sprite 1:

```
set the memberNum of sprite 1 -  
    to the number of member "Red Balloon"
```

- **memberNum of sprite** sprite property; **number of** property

● number of castLib

Syntax: the number of castLib *whichCast*

This cast member property indicates the number of the specified cast. For example, 2 is the number of castLib for Cast 2. The property can be tested but not set.

Example:

This repeat loop uses the message window to display the number of cast members that are in each of the movie's casts:

```
repeat with n = 1 to the number of castLibs
  put the name of castLib n &&"contains"&&the →
    number of members of castLib n&&"cast members."
end repeat
```

● number of castLibs

Syntax: the number of castLibs

This cast member property returns the number of casts that are in the current movie. This property can be tested but not set.

Example:

This repeat loop uses the message window to display the number of cast members that are in each of the movie's casts:

```
repeat with n = 1 to the number of castLibs
  put the name of castLib n &&"contains"&&the →
    number of members of castLib n&&"cast members."
end repeat
```

- **number of members**

Syntax: the number of members

This property indicates the number of the last cast member in the current movie. Any empty cast slots are also counted, so the actual number of cast members may be fewer than the number of members value.

The number of members property can be tested, but not set.

Example:

The following handler returns a string containing a list of all the cast member names, one per line:

```
on castList whichCast
  put EMPTY into list
  repeat with i = 1 to the number of members-
of castLib whichCast
    put the name of member I of castLib whichCast-
    & RETURN after list
  end repeat
  return list
end castList
```

- **number of member** cast property

- **number of chars in**

Syntax: the number of chars in *chunkExpression*

This chunk function returns a count of the characters in a chunk expression.

Chunk expressions refer to any character, word, item, or line in any container of characters. Containers include field cast members and variables that hold strings, and specified characters, words, items, lines, and ranges in containers.

Spaces and control characters such as Tab and Return count as characters.

Examples:

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 33.

This statement sets the variable `charCounter` to the number of characters in the `ith` word in the string `Names`:

```
put the number of chars in word i of member "Names" into
charCounter
```

- **length** function; **number of items in**, **number of lines in**, **number of words in** in chunk functions; **char...of** keyword

- **number of items in**

Syntax: the number of items in *chunkExpression*

This chunk function returns a count of the items in a chunk expression. An item chunk is any sequence of characters delimited by commas.

Chunk expressions refer to any character, word, item, or line in any container of characters. Containers include fields (field cast members) and variables that hold strings, and specified characters, words, items, lines, and ranges in containers.

Example:

This statement 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.

This statement sets the variable `itemCounter` to the number of items in the field Names:

```
put the number of items in field "Names" into
   itemCounter
```

- **item...of** chunk expression keyword, **number of chars in**, **number of lines in**, **number of words in** chunk functions

- **number of lines in**

Syntax: the number of lines in *chunkExpression*

This chunk function returns a count of the lines in a chunk expression.

Chunk expressions are used to refer to any character, word, item, or line in any container of characters. Containers include field cast members and variables that hold strings, and specified characters, words, items, lines, and ranges in containers.

Examples:

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.

This statement sets the variable `lineCounter` to the number of lines in the field Names:

```
put the number of lines in field "Names" into lineCounter
```

- **line...of** chunk expression keyword; **number of chars in**, **number of items in**, **number of words in** in chunk functions

- **number of members of castLib**

Syntax: the number of members of castLib *whichCast*

This cast member property indicates the number of the last cast member in the specified cast. This property can be tested but not set.

Example:

These statements use the message window to display the type of each cast member in the cast Central Casting. The `number of members of castLib` property is used to determine how many times the loop repeats.

```
i = 0
repeat i to the number of members of
of castLib "Central Casting"
  put "Cast member"&&i&&"is a"&&
    (the type of member I of
    castLib "Central Casting" &&"cast member.")
end repeat
```


- **number of menuItems**

Syntax: the number of menuItems of menu *whichMenu*

This menu property indicates the number of menu items in the custom menu specified by *whichMenu*. The *whichMenu* parameter can be a menu name or a menu number.

The `number of menuItems` menu property can be tested but not set directly. Use the `installMenu` command to set up a custom menu bar.

Examples:

This statement sets the variable `fileItems` to the number of menu items in the custom File menu:

```
put the number of menuItems of menu "File" ↵
into fileItems
```

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`:

```
put the number of menuItems of menu i into itemCount
```

- **installMenu** command; **number of menus** property

- **number of menus**

Syntax: `the number of menus`

This menu property indicates the number of menus installed in the current movie.

The `number of menus` menu property can be tested, but not set. Use the `installMenu` command to set up a custom menu bar.

Example:

This statement determines whether there are any custom menus installed in the movie and installs the menu `Menubar` if no menus are already installed:

```
if the number of menus = 0 then -  
    installMenu (the number of member "Menubar")
```

This statement has the message window display the number of menus that are in the current movie:

```
put the number of menus
```

- **installMenu** command; **number of menuitems** property

- **number of words in**

the number of words in *chunkExpression*

This function tells how many words are in the chunk expression specified by *chunkExpression*.

Chunk expressions refer to any character, word, item, or line in any container of characters. Containers include field cast members and variables that hold strings, and specified characters, words, items, lines, and ranges in containers.

Examples:

This statement has the message window display the number of words in the string "Macromedia, the multimedia company":

```
put the number of words ↵
  in "Macromedia, the multimedia company"
```

The result is 4.

This handler reverses the order of words in the string specified by the argument `wordList`:

```
on reverse wordList
  put EMPTY into list
  repeat with i = 1 to the number of words ↵
    in wordList
      put word i of wordList & " " before list
  end repeat
  delete char (the number of chars in list) of list
  return list
end reverse wordList
```

- **number of chars in, number of items in, number of lines in** chunk functions;
word...of chunk expression keyword

- **number of xtras**

Syntax: `the number of xtras`

This property gives the number of Lingo Xtras available to the movie. They could have been opened by the `openxlib` command or are present in the standard Xtras folder. It can be tested but not set.

Example:

This statement has the message window display the number of Lingo Xtras that are available to the movie:

```
put the number of xtras
```

- **Using Xtras**

- **numToChar**

Syntax: `numToChar (integerExpression)`

This function gives a string containing the single character whose ASCII sequence number is the value of *integerExpression*. It is useful for interpreting data from outside sources that are presented as numbers rather than as characters.

Example:

This statement has the message window display the character whose ASCII number is 65:

```
put numToChar (65)
```

The result is the letter "A."

- **charToNum** function

● **objectP**

Syntax: `objectP (expression)`

This function indicates whether the expression specified by *expression* is an object produced by a parent script, Xtra, or XObject.

- When `objectP` is TRUE, the expression is such an object.
- When `objectP` is FALSE, the expression is not such an object.

The "P" in `objectP` stands for "predicate."

It is good practice to use `objectP` to determine which items are XObjects when you create XObjects by using `mNew` or disposing of XObjects by using `mDispose`.

Examples:

This statement checks whether `modemPort` is an XObject and displays the result in the message window:

```
put objectP(modemPort)
```

This handler checks whether `externalFile` is an XObject and disposes of it if it is:

```
on stopMovie
  if objectP(externalFile) then ↵
    externalFile(mDispose)
  end stopMovie
```

- **floatP, integerP, stringP**, and **symbolP** functions; **mDispose** and **mNew** predefined methods; **Using Xtras**

• of

The word `of` is part of many Lingo properties, such as the `foreColor` of `sprite`, the `number of member`, the `name of menu`, and so on.

- **offset**

Syntax: `offset(stringExpression1, stringExpression2)`

This function gives the first place that first character of *stringExpression1* occurs in *stringExpression2*.

- When *stringExpression1* is found in *stringExpression2*, the result is the number that indicates the position of the first occurrence.
- When *stringExpression1* is not found in *stringExpression2*, the result is 0.

Lingo counts spaces as characters in both strings. On the Macintosh, the string comparison is not sensitive to case or diacritical marks. For example, Lingo considers "a" and "Å" the same character on the Macintosh.

Example:

This statement has the message window display the beginning position of the string "media" within the string "Macromedia":

```
put offset("media", "Macromedia")
```

The result is 6.

This statement has the message window display the beginning position of the string "Micro" within the string "Macromedia"

```
put offset("Micro", "Macromedia")
```

The result is 0, because "Macromedia" doesn't contain the string "Micro".

- **chars** and **length** functions; **contains** and **starts** comparison operators

- **offset rect**

Syntax: `offset (rectangle, horizontalChange, verticalChange)`

This function yields a rectangle that is offset from the rectangle specified by *rectangle*. The horizontal offset is the value specified by *horizontalChange*; the vertical offset is the value specified by *verticalChange*.

- When *horizontalChange* is greater than zero, the offset is toward the right of the stage; when *horizontalChange* is less than zero, the offset is toward the left of the stage.
- When *verticalChange* is greater than zero, the offset is toward the top of the stage; when *verticalChange* is less than zero, the offset is toward the bottom of the stage.

The values for *verticalChange* and *horizontalChange* are in pixels.

- **on**

Syntax: on *handlerName* [*argument1*] [, *arg2*] [, *arg3*] ...
 [*statements*]
 end *handlerName*

This keyword indicates the beginning of a handler. Handlers are collections of Lingo statements that you can execute by simply using the handler name. A handler can accept arguments as input values and return a value as a function result.

Handlers can be defined in score scripts, movie scripts, and scripts of cast members. A handler in a script of a cast member can only be called by other handlers in the same script. A handler in a score script or movie script can be called from anywhere.

You can use the same handler in more than one movie by putting the handler's script in the shared cast.

- **on activateWindow**, **on closeWindow**, **on enterFrame**, **on exitFrame**, **on idle**, **on keyDown**, **on keyUp**, **on mouseDown**, **on mouseUp**, **on resizeWindow**, **on startMovie**, **on stepMovie**, **on stopMovie**, and **on zoomWindow** event handlers

● on activateWindow

Syntax:

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

This event handler contains statements that are executed when the movie is running as a movie in a window and the window becomes active, such as when the user clicks the inactive window.

The `on activateWindow` handler is a good place for Lingo that you want executed every time the movie becomes active.

Example:

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

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

● on closeWindow

Syntax:

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

This event handler contains statements that are executed when the movie is running as a movie in a window and the user closes the window by clicking the window's close box.

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

Example:

This handler plays the sound file Sigh when the window that the movie is playing in closes:

```
on closeWindow
  puppetSound 2, "Sigh"
end
```

● **on deactivateWindow**

Syntax: on deactivateWindow
 statement(s)
 end

This message is sent when the window 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 file Snore when the window that the movie is playing in is deactivated:

```
on deactivateWindow
    puppetSound 2, "Snore"
end
```

- **on enterFrame**

Syntax: on enterFrame
 statement(s)
 end enterFrame

This event handler contains statements that are executed each time the playback head enters the frame that the `on enterFrame` handler is attached to. The `on enterFrame` handler is equivalent to the `on stepMovie` handler used in earlier versions of Director.

The `on enterFrame` event handler is a good place for Lingo that you want executed once at every new frame.

Place `on enterFrame` handlers in frame scripts or movie scripts.

- When you want to assign the handler to an individual frame, put the handler in the frame script.
- When you want to assign the handler to every frame unless you explicitly instruct the movie otherwise, put the `on enterFrame` handler in a movie script. The handler then executes every time the playback head enters a frame unless the frame script has its own `on enterFrame` handler. When the frame script has its own `on enterFrame` handler, the `on enterFrame` handler in the frame script overrides the one in the movie script.

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
```

- **on exitFrame, on idle, on keyDown, on keyUp, on mouseDown, on mouseUp, on startMovie, on stepMovie, and on stopMovie** event handlers

- **on exitFrame**

Syntax: on exitFrame
 statement(s)
 end exitFrame

This event handler contains statements that are activated 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 frame scripts or movie scripts.

- When you want to assign the handler to an individual frame, put the handler in the frame script.
- When you want to assign the handler to every frame unless explicitly instructed otherwise, put the handler in a movie script. The `on exitFrame` handler then executes every time the playback head exits the frame unless the frame script has its own `on exitFrame` handler. When the frame script has its own `on exitFrame` handler, the `on exitFrame` handler in the frame script overrides the one in the movie script.

Example:

This handler turns off all puppetSprite conditions when the playback head exits the frame:

```
on exitFrame
  repeat with i = 48 down to 1
    set the puppet of sprite i = FALSE
  end repeat
end exitFrame
```

This handler sends the playback head to a specified frame if the value in the variable `vTotal` exceeds 1000 when the playback head exits the frame:

```
on exitFrame
  if vTotal > 1000 then go to frame "Finished"
end
```

- **on enterFrame, on idle, on keyDown, on keyUp, on mouseDown, on mouseUp, on startMovie, on stepMovie, and on stopMovie** event handlers

- **on idle**

Syntax: on idle
 statement(s)
 end idle

This event handler contains statements that are executed whenever the movie has no other events to handle.

This is a useful location for Lingo statement that you want to execute as frequently as possible. Some common cases are updating values in global variables and displays that tell current movie conditions.

Because statements in `on idle` handlers run frequently, it is good practice to avoid placing Lingo that takes a long time to process in an `on idle` handler.

It is often preferable to put `on idle` handlers in frame scripts instead of movie scripts. This makes it easier to turn off the `on idle` handler when appropriate.

Example:

This handler updates the time being displayed in the movie whenever there are no other events to handle:

```
on idle
  put the short time into field "Time"
end idle
```

- **on enterFrame, on exitFrame, on keyDown, on keyUp, on mouseDown, on mouseUp, on startMovie, on stepMovie, and on stopMovie** event handlers

● on keyDown

Syntax: on keyDown
 statement(s)
 end

This event handler contains statements that are activated when a key is pressed.

When a key is pressed, Lingo searches these locations, in order, for an on keyDown handler: primary event handler, editable field sprite script, script of a field cast member, frame script, and movie script. (For sprites and cast members, on keyDown handlers work only for editable strings. A keyDown on a different type of cast member, such as a bitmap, has no effect.)

Lingo stops searching when it reaches the first location that has an on keyDown handler, unless the handler includes the pass command to explicitly pass the keyDown message on to the next location.

The on keyDown event handler is a good place to put Lingo that implements keyboard shortcuts or other interface features that you want to have occur when the user presses keys.

Where you place an on keyDown handler can affect when it runs.

- When you want the handler to apply to a specific editable field sprite, put the handler in a sprite script.
- When you want the handler to apply to an editable field cast member in general, put the handler in a script of the cast member.
- When you want the handler to apply to an entire frame, put the handler in a frame script.
- When you want the handler to apply throughout the entire movie, put the handler in a movie script.

You can override an on keyDown handler by placing an alternate on keyDown handler in a location that Lingo checks before it gets to the handler you want to override. For example, you can override an on keyDown handler assigned to a cast member by placing an on keyDown handler in a sprite script.

Example:

This handler checks whether the Return key was pressed and sends the playback head to another frame if it was:

```
on keyDown
  if the key = RETURN then go to frame "AddSum"
end keyDown
```

- on enterFrame, on exitFrame, on idle, on keyUp, on mouseDown, on mouseUp, on startMovie, on stepMovie, and on stopMovie event handlers

• **on keyUp**

Syntax: on keyUp
 statement(s)
 end

This event handler contains statements that are activated when a key is released. The on keyUp handler is similar to the on keyDown handler.

When a key is released, Lingo searches these locations, in order, for an on keyUp handler: primary event handler, editable field sprite script, script of a field cast member, frame script, and movie script. (For sprites and cast members, on keyUp handlers work only for editable strings. A keyUp on a different type of cast member, such as a bitmap, has no effect.)

Lingo stops searching when it reaches the first location that has an on keyUp handler, unless the handler includes the pass command to explicitly pass the keyUp message on to the next location.

The on keyUp event handler is a good place to put Lingo that implements keyboard shortcuts or other interface features that you want to have occur when the user releases keys.

Where you place an on keyUp handler can affect when it runs:

- When you want the handler to apply to a specific editable field sprite, put the handler in a sprite script.
- When you want the handler to apply to an editable field cast member in general, put the handler in a script of the cast member.
- When you want the handler to apply to an entire frame, put the handler in a frame script.
- When you want the handler to apply throughout the entire movie, put the handler in a movie script.

You can override an on keyUp handler by placing an alternate on keyUp handler in a location that Lingo checks before it gets to the handler you want to override. For example, you can override an on keyUp handler assigned to a cast member by placing an on keyUp handler in a sprite script.

Example:

This handler checks whether the Return key was released and sends the playback head to another frame if it was:

```
on keyUp
  if the key = RETURN then go to frame "AddSum"
end keyUp
```

- **on enterFrame, on exitFrame, on idle, on keyDown, on mouseDown, on mouseUp, on startMovie, on stepMovie, and on stopMovie** event handlers

● **on mouseDown**

Syntax: on mouseDown
 statement(s)
 end

This event handler contains statements that are activated when the mouse button is pressed.

When the mouse button is pressed, Lingo searches these locations, in order, for an `on mouseDown` handler: primary event handler, sprite script, script of a cast member, frame script, and movie script. Lingo stops searching when it reaches the first location that has an `on mouseDown` handler, unless the handler includes the `pass` command to explicitly pass the `mouseDown` message on to the next location.

The `on mouseDown` event handler is a good place to put Lingo that flashes images, triggers sound effects, or makes sprites move when the user presses the mouse button.

Where you place an `on mouseDown` handler can affect when it runs.

- When you want the handler to apply to a specific sprite, put the handler in a sprite script.
- When you want the handler to apply to a cast member in general, put the handler in a script of the cast member.
- When you want the handler to apply to an entire frame, put the handler in a frame script.
- When you want the handler to apply throughout the entire movie, put the handler in a movie script.

You can override an `on mouseDown` handler by placing an alternate `on mouseDown` handler in a location that Lingo checks before it gets to the handler you want to override. For example, you can override an `on mouseDown` handler assigned to a cast member by placing an `on mouseDown` handler in a sprite script.

Example:

This handler checks whether the user clicks anywhere on the stage and sends the playback head to another frame if he or she does:

```
on mouseDown
  if the clickOn = 0 then go to frame "AddSum"
end mouseDown
```

This handler, assigned to a sprite script, plays a sound when the sprite is clicked:

```
on mouseDown
  puppetSound "Crickets"
end mouseDown
```

- **on enterFrame, on exitFrame, on idle, on keyDown, on keyUp, on mouseUp, on startMovie, on stepMovie, and on stopMovie** event handlers

● **on mouseUp**

Syntax: on mouseUp
 statement(s)
 end mouseUp

This event handler contains statements that are activated when the mouse button is released.

When the mouse button is released, Lingo searches these locations, in order, for an `on mouseUp` handler: primary event handler, sprite script, script of a cast member, frame script, and movie script. Lingo stops searching when it reaches the first location that has an `on mouseUp` handler, unless the handler includes the `pass` command to explicitly pass the `mouseUp` message on to the next location.

An `on mouseUp` event handler is a good place to put Lingo that changes the appearance of objects--such as buttons--after they are clicked. You can do this by switching the cast member assigned to the sprite after the sprite is clicked and the mouse button is released. The sprite's different appearance indicates that the sprite has already been clicked.

Where you place an `on mouseUp` handler can affect when it runs.

- When you want the handler to apply to a specific sprite, put the handler in a sprite script.
- When you want the handler to apply to a cast member in general, put the handler in a script of the cast member.
- When you want the handler to apply to an entire frame, put the handler in a frame script.
- When you want the handler to apply throughout the entire movie, put the handler in a movie script.

You can override an `on mouseUp` handler by placing an alternate `on mouseUp` handler in a location that Lingo checks before it gets to the handler you want to override. For example, you can override an `on mouseUp` handler assigned to a cast member by placing an `on mouseUp` handler in a sprite script.

Example:

This handler, assigned to sprite 10, switches the cast member assigned to sprite 10 when the user releases the mouse button after clicking the sprite:

```
on mouseUp
  puppetSprite 10, TRUE
  set the memberNum of sprite 10 to
  the number of member "Dimmed"
end mouseUp
```

- **on enterFrame, on exitFrame, on idle, on keyDown, on keyUp, on mouseDown, on startMovie, on stepMovie, and on stopMovie** event handlers

● **on moveWindow**

Syntax: on moveWindow
 statement(s)
 end

This event handler contains statements that are executed when a window is moved, such as when the user drags the movie to a new location on the stage. The `on moveWindow` handler is a good place to put Lingo that you want executed every time the movie's window moves.

Example:

This handler plays the sound file Honk when the window that the movie is playing in moves:

```
on moveWindow
  puppetSound 2, "Honk"
end
```

- **on openWindow**

Syntax: on openWindow
 statement(s)
 end

This event handler contains statements that are executed when Director opens a window. The `on openWindow` handler is a good place to put Lingo that you want executed every time the movie's window opens.

Example:

This handler plays the sound file Hurray when the window that the movie is playing in opens:

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

● on resizeWindow

Syntax: on resizeWindow
 statement(s)
 end resizeWindow

This event handler contains statements that are activated when a movie is running as a movie in a window and the user resizes the window by dragging the window's grow box or one of its edges.

An `on resizeWindow` event handler is a good place to put Lingo related to the window's dimensions, such as positioning sprites and cropping digital video.

Example:

This handler moves sprite 3 to the coordinates stored in the variable `centerPlace` when the window that the movie is playing in is resized:

```
on resizeWindow centerPlace
    set the loc of sprite 3 to centerPlace
end
```

● on rightMouseDown

Syntax:

```
on rightMouseDown
  statements
end rightMouseDown
```

This event handler contains statements that are activated when the right mouse button on a Windows computer is pressed. For Macintosh computers, the statements are activated when the mouse button and Control key are pressed at the same time, provided that the `emulateMultiButtonMouse` property is set to `TRUE`. If the `emulateMultiButtonMouse` property isn't set to `TRUE`, this event handler has no effect on the Macintosh.

Example:

This handler opens the window Help when the user clicks the right mouse button in Windows:

```
on rightMouseDown
  open window "Help"
end
```


● on rightMouseUp

Syntax:

```
on rightMouseUp
    statements
end rightMouseUp
```

This event handler contains statements that are activated when the right mouse button on a Windows computer is released.

For Macintosh computers, the statements are activated if the mouse button is released while the Control key is pressed, provided that the `emulateMultiButtonMouse` property is set to `TRUE`. If the `emulateMultiButtonMouse` property isn't set to `TRUE`, this event handler has no effect on the Macintosh.

Example:

This handler opens the window Help when the user releases the right mouse button in Windows:

```
on rightMouseUp
    open window "Help"
end
```

- **on startMovie**

Syntax: on startMovie
 statement(s)
 end startMovie

This event handler contains statements that are activated after the movie preloads cast members but before the movie starts playing, regardless of where the playback head is.

An `on startMovie` handler is a good place to put Lingo that opens resource files, creates global variables, initializes variables, plays a sound while the rest of the movie is loading into memory, and checks and adjusts to computer conditions such as color depth.

Example:

This handler creates a global variable when the movie starts:

```
on startMovie
  global currentScore
  set currentScore = 0
end startMovie
```

- **on enterFrame, on exitFrame, on idle, on keyDown, on keyUp, on mouseDown, on mouseUp, on stepMovie**, and **on stopMovie** event handlers

- **on stepMovie**

Syntax: on stepMovie
 statement(s)
 end stepMovie

This handler contains statements that are executed each time the playback head enters a new frame. This handler, which was used in earlier versions of Director, has the same result as the `on enterFrame` handler.

The `on stepMovie` handler has no effect in Windows. For the best results, use the `on enterFrame` handler for Lingo that should run each time the playback head enters a new frame.

- **on enterFrame, on exitFrame, on idle, on keyDown, on keyUp, on mouseDown, on mouseUp, on startMovie, and on stopMovie** event handlers ; **perFrameHook** property

- **on stopMovie**

Syntax: on stopMovie
 statement(s)
 end stopMovie

This event handler contains statements that are activated 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.

Example:

This handler clears global variables and closes two resource files when the movie stops:

```
on stopMovie
  set gCurrentScore = 0
  closeResFile "Special Fonts"
  closeResFile "Special Cursors"
end stopMovie
```

- **on enterFrame, on exitFrame, on idle, on keyDown, on keyUp, on mouseDown, on mouseUp, on startMovie, and on stepMovie** event handlers

- **on timeOut**

Syntax: on timeOut
 statement (s)
 end timeOut

This event handler contains statements that are executed when the no one uses the keyboard or mouse for the length of time set in the `timeOutLength`. This is a useful location for Lingo that you want to execute when the user does nothing for a specified length of time.

An `on timeOut` handler must be placed in a movie script.

Example:

This handler plays the movie Attract Loop after users do nothing for the time set in the `timeOutLength` property. This would be a way to respond when users have gone away from the computer:

```
on timeOut
  play movie "Attract Loop"
end timeOut
```

- **on enterFrame, on exitFrame, on idle, on keyDown, on keyUp, on mouseDown, on mouseUp, on startMovie**, and **on stepMovie** event handlers

● **on zoomWindow**

Syntax: on zoomWindow
 statement(s)
 end zoomWindow

This event handler contains statements that are activated when a movie that is running as a movie in a window is resized when the user clicks the minimize/maximize button (Windows) or the zoom button (Macintosh). The operating system determines the dimensions after resizing the window.

An `on zoomWindow` event handler is a good place to put Lingo intended to rearrange sprites when window dimensions change.

Example:

This handler moves sprite 3 to the coordinates stored in the variable `centerPlace` when the window that the movie is playing in is resized:

```
on zoomWindow centerPlace
    set the loc of sprite 3 to centerPlace
end
```

- **open**

Syntax: open [*whichDocument* with] *whichApplication*

This command launches the application specified by the string *whichApplication*. By specifying *whichDocument*, you can specify a document that the application opens at the same time. When either is in a different folder than the current movie, you must specify the pathname.

If you are running MultiFinder, there must be enough memory to run both Macromedia Director and the other application at the same time.

Example:

This statement checks whether the computer the computer is a Macintosh (by checking whether it isn't a PC) and then opens the application MacWrite if it is:

```
if the machineType <> 256 then open "MacWrite"
```

This statement opens the MacWrite application, which is in the folder Applications on the drive myDrive, and the document named Storyboards:

```
open "Storyboards" with myDrive & "Applications:" -  
  & "MacWrite"
```

- **openResFile**, and **openXlib** commands

- **open window**

Syntax: `open window "whichWindow"`

This command opens a window that can play a Director movie and brings it to the front of the stage. The window is specified by *whichWindow* and must have a movie already assigned to it before you can use the `open window` command.

Example:

This statement opens the window Control Panel and brings it to the front:

```
open window "Control Panel"
```

- **close window** command

- **openResFile**

Syntax: `openResFile` *whichFile*

On the Macintosh, this command opens the resource file specified by the string *whichFile*. When the file is in a different folder than the current movie, *whichFile* must specify a pathname.

In earlier versions of Director, this command was necessary to make additional fonts and cursors available in your movies. However, you can now provide custom cursors by importing the cursor as a cast member and using the `cursor` property.

When the file is already open, `openResFile` has no effect. It is good practice to close any open file as soon as you are finished using it.

Do not use `openResFile` to open another application. (Its code resources will interfere with those of Macromedia Director.) Use a resource mover like ResEdit to move the resources you need to a separate resource file.

Example:

This statement opens the resource file Special Fonts:

```
openResFile "Special Fonts"
```

This statement opens the resource file Special Icons, which is in another folder:

```
openResFile the pathName&"Special Icons"
```

- **closeResFile** and **showResFile** commands; **cursor** property

- **openXlib**

Syntax: `openXlib whichFile`

This command opens the Xlibrary file specified by the string expression *whichFile*. If the file is in a different folder than the current movie, *whichFile* must include the pathname.

It is good practice to close any file you have opened as soon as you are finished using it. When the file is already open, `openXlib` has no effect.

Xlibrary files contain Xtras and XObjects as XCOD resources. Unlike `openResFile`, `openXlib` makes these Xtras and XObjects known to Director.

In Windows, the .DLL extension is optional.

The `openXlib` command also opens HyperCard XCMDs and XFCNs so that you can use them with Director on the Macintosh. When you need to use an XCMD from more than one application in a movie, use this command to open a link to the HyperCard stack, rather than install the XCMD in both places with ResEdit. When you do that, a resource conflict that results in a system beep occurs.

Example:

This statement opens the Xlibrary file Video Disc Xlibrary:

```
openXlib "Video Disc Xlibrary"
```

This statement opens the Xlibrary file Xtras, which is in a different folder than the current movie:

```
openXlib "My Drive:New Stuff:Transporter Xtras"
```

- **closeXlib** and **showXlib** commands

- **optionDown**

Syntax: the optionDown

This function determines whether the Option key on the Macintosh or the Alt key on the PC is being pressed.

- When the Option or Alt key is being pressed, the optionDown is TRUE.
- When the Option or Alt key is not being pressed, the optionDown is FALSE.

Example:

This handler checks whether the Option key or Alt key is being pressed and calls the handler named `doOptionKey` if it is:

```
on keyDown
  if the optionDown then doOptionKey(the key)
end keyDown
```

- **controlDown**, **commandDown**, **key**, and **shiftDown** functions

- **or**

Syntax: *logicalExpression1 or logicalExpression2*

This operator performs a logical OR operation on two logical expressions.

- When either expression or both expressions are TRUE, the result is TRUE (1).
- When both expressions are FALSE, the result is FALSE (0).

This is a logical operator with a precedence level of 4.

Examples:

This statement has the message window display whether at least one of the expressions `1 < 2` and `1 > 2` is TRUE:

```
put 1 < 2 or 1 > 2
```

Because the first expression is TRUE, the result is 1, which is the numerical equivalent of TRUE.

This statement checks whether the contents of the field cast member named `field` are either AK or HI, and displays an alert if they are:

```
if field "State" = "AK" or field "State" = "HI" ↵  
  then alert "You're off the map!"
```

- **and** and **not** logical operators

- **otherwise**

Syntax: otherwise *statement*

This optional keyword precedes instructions that Lingo carries out when none of the earlier conditions in a case statement are met.

Example:

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

- If the user pressed A, B, or C, the movie performs the corresponding action following the `of` keyword.
- If the user pressed any other key, the movie executes the statement that follows the `otherwise` keyword. In this case, the statement is a simple beep.

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

- **pageHeight of member**

Syntax: the pageHeight of member *whichCastmember*

This field cast member property gives 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 gets the height of the visible portion of the field cast member Today's News:

```
put the pageHeight of member "Today's News"
```

- **palette of member**

Syntax: the palette of member *whichCastMember*

This cast member property determines which palette is associated with the cast member specified by *whichCastMember*. This property applies to bitmap cast members only.

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

Example:

This statement displays the palette assigned to the cast member Leaves in the message window:

```
put the palette of member "Leaves"
```

- **paletteMapping**

Syntax: the paletteMapping

This movie property determines whether Director remaps the movie's palette. Its effect is similar to the Remap Palettes When Needed checkbox in the **Movie Properties** dialog box.

- When the paletteMapping is TRUE, the movie remaps palettes for cast members whose palette is different than the current movie palette.
- When the paletteMapping is FALSE, the movie doesn't remap palettes for cast members.

Example:

This statement has the movie always remap the movie's palette when needed:

```
set the paletteMapping = TRUE
```


● paletteRef

Syntax: the paletteRef

This property determines the palette associated with a bitmap cast member. Built-in Director palettes are indicated by symbols (`#systemMac`, `#rainbow`, etc...). Palettes that are cast members are treated as cast member references. This differs from `the palette of member`, which returns a positive number for cast palettes and negative numbers for built-in Director palettes.

The `paletteRef` property can be tested and set.

Example:

This statement assigns the Mac system palette to the bitmap cast member Shell:

```
set the paletteRef of member "Shell" to #systemMac
```

- **param**

Syntax: param(*parameter*)

This function gives the value of a parameter in a list. The variable *parameter* represents the parameter's position in the list.

Example:

This handler calculates the average value of a list of parameters:

```
on avg first, second, third
  set n = paramCount()
  set sum = 0.0
  repeat with i = 1 to n
    set sum = param(i) + sum
  end repeat
  return sum/n
end avg
```

This statement passes the handler three values and displays the result in the message window:

```
put avg(1,2,3)
--> 2.0
```

- **paramCount** function

- **paramCount**

Syntax: the paramCount

This function determines 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 = paramCount()
```

- **pass**

Syntax: `pass`

This command passes an event message to the next location in the message hierarchy. Otherwise, an event message stops at the first location that contains a handler for the event. (The exception is for messages such as `mouseDown` and `timeOut` that go to primary event handlers first. Lingo always passes these messages on after the primary event handler, unless the primary event handler includes the `dontPassEvent` command.)

Passing an event message to other locations in the message hierarchy lets you execute more than one handler for a given event.

Examples:

Used together, these handlers are both activated by a `mouseUp` event because the first handler contains a `pass` command.

This `on mouseUp` handler attached to sprite 3 executes the handler and then passes the `mouseUp` message on:

```
on mouseUp
  if sprite 3 intersects sprite 4 ↵
    then set the text of member 10 = ↵
      "You placed it correctly"
  pass
end
```

This `on mouseUp` handler in the frame script executes because the `on mouseUp` handler assigned to the sprite script contains the `pass` command:

```
on mouseUp
  go to "Next test"
end
```

- **dontPassEvent** command

● **pasteClipboardInto**

Syntax: `pasteClipboardInto member whichCastMember`

This command pastes the contents of the Clipboard into the cast member specified by *whichCastMember*. When you paste into an occupied cast window slot, the old cast member is completely erased. For instance, pasting a bitmap into a field cast member makes the bitmap the cast member and erases the field cast member.

You can paste any item that is in a format that Director can use as a cast member. When you copy strings from another application, the string's formatting is not retained.

The `pasteClipboardInto cast` command provides a convenient way to copy objects from other movies and from other applications into the cast window.

Example:

This statement pastes the contents of the Clipboard into the bitmap cast member Shrine:

```
pasteClipboardInto member "Shrine"
```

- **pathName**

Syntax: the pathName

This function returns a string containing the full pathname of the folder in which the current movie is located.

You can write pathnames that work on both Windows and Macintosh computers by using the @ operator.

Example:

This statement checks whether the pathname contains the term System and has the computer beep if it does:

```
if the pathName contains "System" then beep
```

Example 2:

These statements check whether the movie is playing in Windows or on the Macintosh, and then plays the sound file Crash.AIF in the Sounds subfolder of the current movie's folder. By checking which platform the movie is playing on, the movie uses the `sound playFile` statement that has the appropriate folder delimiter:

```
case (the platform) of
  "Windows,32", "Windows,16" :sound playFile 1, -
  the pathname&"sounds/crash.aif"
  otherwise sound playFile 1, the pathname&-
  "sounds:crash.aif"
end case
```

- **movie** function

● pattern

Syntax: the pattern of member *whichCastmember*

This shape cast member property determines the pattern associated with the specified shape. Possible values are the numbers that correspond to the chips in the tools window's patterns palette. If the shape cast member is unfilled, the pattern is applied to the cast member's outer edge. This property can be tested and set.

Example:

The following statements make the shape cast member Target Area a filled shape and assigns it the pattern numbered 0, which is a solid color:

```
set the filled of member "Target Area" = TRUE  
set the pattern of member "Target Area" = 0
```

- **pause**

Syntax: `pause`

This command causes the playback head to halt. Typically, you would put the `pause` command in the script channel of a frame, and then assign `continue` or `go` commands to one or more sprite scripts in that frame.

In many cases, using `pause` is recommended over looping on the same frame, or looping between two frames. This is because a pause uses much less processor time than repeatedly moving the playback head to the beginning of the frame. Some exceptions to this general rule are when you are moving sprites or are using the `perFrameHook`, which requires that the playback head keeps going to the same frame.

The `pause` command is useful for halting the movie while a menu is displayed or for letting the user look at a screen as long as she or he wants.

Example:

The following `on mouseUp` handler for a button alternately pauses and continues the animation, like the pause button on a videocassette recorder:

```
on mouseUp
  if the pauseState = TRUE then
    continue
  else
    pause
  end if
end mouseUp
```

- **continue** command; **pauseState** function

- **pausedAtStart of member**

Syntax: the pausedAtStart of member *whichDVMovie trueOrFalse*

This digital video cast member property specifies whether the Paused at Start checkbox in the **Digital Video Cast Member Properties** dialog box is checked or not.

- When the pausedAtStart of member property is TRUE, the Paused at Start checkbox is checked.
- When the pausedAtStart of member property is FALSE, the Paused at Start checkbox is not checked.

The pausedAtStart of member property can be tested and set.

Example:

This statement turns on the Paused at Start checkbox in the Digital Video Cast Member Info dialog box for the QuickTime movie Rotating Chair:

```
set the pausedAtStart of member "Rotating Chair" = TRUE
```

- **pauseState**

Syntax: the pauseState

This function returns TRUE when the movie is currently paused.

Example:

This statement checks whether the movie is currently paused and has the movie continue if it is:

```
if the pauseState = TRUE then continue
```

- **pause** and **continue** commands

- **perFrameHook**

Syntax: the perFrameHook

The `perFrameHook` property was required in earlier versions of Director. However, you can now achieve the same results by placing Lingo that you want to execute at every frame in an `on enterFrame` handler. It is now only relevant for managing the `actorList`.

The `perFrameHook` property designates an object created by an `XObject` that is called every frame with a special message called `mAtFrame`. You specify which routines and procedures are used in `mAtFrame`.

The `perFrameHook` property can be used to call a certain set of procedures (using `mAtFrame`) each frame. Without the `perFrameHook`, you would have to type this set of procedures (using a handler) into the script channel of every single frame in which you wanted it to occur. With the `perFrameHook`, you need only set the proper object to the `perFrameHook` once and the procedures (contained in the `mAtFrame` method) will be executed at every frame. When you no longer want to use the `perFrameHook`, set it to 0 to turn it off. The `perFrameHook` is especially useful when recording animations frame-per-frame to videotape.

At every frame, the `perFrameHook` object is sent the `mAtFrame` message. Therefore, you must create an `XObject` that defines an `mAtFrame` method (in the same `XObject` that creates the object you set to the `perFrameHook`).

When recording frame-per-frame to videotape, you can define two arguments for `mAtFrame` that specify the frame and subframe (subframes occur during transitions; each change during the transition is a subframe):

```
method mAtFrame frame, subframe
```

The frame argument is sent for each frame and the subframe argument is sent for each subframe. You can name the arguments whatever you like, if you prefer not to use `frame` or `subframe`. You can also define additional arguments for `mAtFrame`, whether you are recording frame-per-frame to videotape or not.

The `perFrameHook` is primarily designed to be used with `XObjects` that have an `mAtFrame` argument. If you do use the `perFrameHook` with an `XObject`, do not use the `updateStage` command. Otherwise, unexpected results could occur.

- **method** keywords

• pi

Syntax: `pi()`

This function gives the value of π , the ratio of a circle's circumference to its diameter. The value of π is given as a floating point number to the number of decimal places set by the `floatPrecision` property.

Example:

This statement uses the pi function as part of an equation for calculating the area of a circle:

```
set vArea = pi()*power(vRadius,2)
```

- **picture of member**

Syntax: the picture of member *whichCastmember*

This cast member property determines which image is associated with a bitmap, text, or PICT cast member. To update changes to a cast member's registration point or update changes to an image after relinking it using the `fileName` property, use the following statement:

```
set the picture of member whichCastmember = the picture ↵  
of member whichCastmember
```

where you replace *whichCastmember* with the name or number of the affected cast member.

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

Example:

This statement sets the variable named `pic` to the image in the cast member named `Sunset`:

```
put the picture of member "Sunset" into pic
```

- **type of sprite** property

● pictureP

Syntax: `pictureP(pictureValue)`

This function tells the state of the `picture of member` property for the specified cast member.

- When the `picture of member` property is TRUE, `pictureP` is TRUE (1).
- When the cast member is not a picture data type, `pictureP` is FALSE (0).

Because `pictureP` doesn't directly check whether a cast member has a picture, you must test whether a cast member has a picture by checking the cast member's `picture of member` property.

Example:

The first statement assigns the value of the `picture of member` property for the cast member Shrine, which is a bitmap, to the variable `pictureValue`. The second statement checks whether Shrine is a picture by checking value assigned to `pictureValue`:

```
put pictureP(pictureValue)
```

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

● platform

Syntax: the platform

This system property indicates which platform the movie was created on. It can be tested, but not set. Possible values are the following:

Possible value	Corresponding platform
Macintosh,68k	Original 68K Macintosh
Macintosh,PowerPC	PPC Macintosh
Windows,16	Windows 3.1 or earlier
Windows,32	Windows 95 or WinNT

Example:

This statement checks whether the movie was created in Windows 95 and assigns the cast Win95 Art the name Interface if it is:

```
if the platform = "Windows,32" then set the name ↵  
of castLib "Win95 Art" to "Interface"
```

- **play**

Syntax: `play [frame] whichFrame`
 `play movie whichMovie`
 `play frame whichFrame of movie whichMovie`

This command causes the playback head to jump to the specified frame of the specified movie. The expression *whichFrame* can be either a string marker label or an integer frame number. The expression *whichMovie* must be a string that specifies a movie file. When the movie is in another folder, *whichMovie* must specify a pathname.

The `play` command is similar to the `go to` command, but with the `play` command, when the sequence being played is over, the playback head automatically returns to the frame where the `play` command was called. If the `play` command is issued from a frame script, the playback head returns to the next frame; if the `play` command comes from a sprite script or handler, the playback head returns to the same frame. A sequence is over when the playback head reaches the end of the movie, or the `play done` command is given.

The `play` command can also be used for playing several movies from a single handler. The handler is suspended while each movie plays, but resumes when the movie is over. Contrast this with a series of `go` commands that, when called from a handler, play the first frame of each movie. The handler is not suspended while the movie plays but immediately continues executing.

If a `play done` command isn't used to indicate the end of a segment started by a `play` command, memory gets used up because the original calling script isn't deleted. If you aren't sure that each `play` command has a matching `play done` command, consider avoiding `play` commands. Instead, use a global list to record where the movie should return to.

Examples:

This statement moves the playback head to the marker named blink:

```
play "blink"
```

This statement moves the playback head to the next marker:

```
play marker(1)
```

This statement moves the playback head to a separate movie:

```
play movie "My Drive:More Movies:" & newMovie
```

- **go** and **play done** commands; **marker** function

- **play done**

Syntax: `play done`

This command indicates that the sequence being played is complete when the current movie or sequence was started using the `play` or `go to` commands. The `play done` command causes the playback head to return to where the sequence was started from. If the `play` command is issued from a frame script, the playback head returns to the next frame; if the `play` command is issued from a sprite script, the playback head returns to the same frame.

If a `play done` command isn't used to indicate the end of a segment started by a `play` command, memory gets used up because the original calling script isn't deleted. If you aren't sure that each `play` command has a matching `play done` command, consider avoiding `play` commands. Instead, use a global list to record where the movie should return to.

Note: The `play done` command has no effect in a movie that is playing in a window.

- **play command**

- **point**

Syntax: `point(horizontal, vertical)`

This function yields a point that has the horizontal coordinate specified by *horizontal* and the vertical coordinate specified by *vertical*.

Example:

This statement sets the variable `lastLocation` to the point (250, 400):

```
put point(250, 400) into lastLocation
```

- **rect** function

- **power**

Syntax: `power(base, exponent)`

This function calculates the value of the number specified by *base* to the exponent specified by *exponent*.

Example:

This statement sets the variable `vResult` the value of 4 to the third power:

```
set vResult = power(4,3)
```

- **preLoad**

Syntax: `preLoad`
 `preLoad toFrameNum`
 `preLoad fromFrame, toFrameNum`

This command preloads cast members in the specified frame or range of frames into memory. Preloading stops when memory is full or when all of the specified cast members have been preloaded.

When used without arguments, the `preLoad` command causes a preload of all cast members used from the current frame to the last frame of a movie.

When used with one argument, *toFrame*, the `preLoad` command causes a preload of all cast members used in the range of frames from the current frame to the frame *toFrame*, as specified by frame number or label name.

When used with two arguments, *fromFrame* and *toFrame*, the `preLoad` command causes a preload of all cast members used in the range of frames from the frame *fromFrame* to the frame *toFrame*, as specified by frame number or label name.

The `preLoad` command also returns the number of the last frame successfully loaded. To access this value, use the `result` function.

Examples:

This statement preloads the cast members used from the current frame to the frame that has the next marker:

```
preLoad marker (1)
```

This statement preloads the cast members used from frame 10 to frame 50:

```
preLoad 10, 50
```

- **preLoadMember** command

● **preLoad of member**

Syntax: the preLoad of member *whichCastmember*

This digital video cast member property determines whether the digital video cast member specified by *whichCastmember* can preload into memory.

- When the digital video cast member can be preloaded into memory, the preLoad of member is TRUE.
- When the digital video cast member can not be preloaded into memory, the preLoad of member is FALSE.

Setting the preLoad of member to TRUE has the same effect as selecting Enable Preload in the **Digital Video Cast Member Properties** dialog box.

Example:

This statement has the message window display whether the QuickTime movie Rotating Chair can be preloaded into memory:

```
put the preLoad of member "Rotating Chair"
```

● preLoadMember

Syntax: preLoadMember
 preLoadMember *whichCastmember*
 preLoadMember *fromCastmember, toCastmember*

This command preloads cast members. Preloading stops when memory is full or when all of the specified cast members have been preloaded.

When used without arguments, the `preLoadMember` command preloads all cast members in the movie.

When used with the *whichCastmember* argument, the `preLoadMember` command preloads that cast member.

When used with the arguments *fromCastmember* and *toCastmember*, the `preLoadMember` command preloads all cast members in the range specified by the cast member numbers or names.

The `preLoadMember` command returns the cast member number of the last cast member successfully loaded. To obtain this value, use the `result` function.

Example:

This statement preloads cast member 20:

```
preLoadMember 20
```

This statement preloads cast member Shrine and the ten cast members after it in the cast window:

```
preLoadMember "Shrine", (the number of member "Shrine" + 10)
```

- **preLoadEventAbort**

Syntax: the preLoadEventAbort

This property specifies whether pressing keys or clicking the mouse can stop preloading of cast members.

- When the preLoadEventAbort property is TRUE, pressing keys or clicking the mouse can stop preloading of castmembers.
- When the preLoadEventAbort property is FALSE, pressing keys or clicking the mouse cannot stop preloading of cast members.

The default value is FALSE. The setting of this property affects the current movie.

The preLoadEventAbort property can be tested and set.

Example:

This statement lets the user stop preloading of cast members by pressing keys or clicking the mouse:

```
set the preLoadEventAbort = TRUE
```

- **preLoad** and **preLoadMember** commands

- **preLoadMode of CastLib**

Syntax: the preLoadMode of castLib *whichCast*

This cast property determines the specified cast's preload mode. This has the same effect as setting Load Cast in the **Movie Properties** dialog box. Possible values are the following:

- 0--When Needed
- 1--Before Frame One
- 2--After Frame One

An `on startMovie` 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 has Director load the members of the cast Buttons before the movie enters frame one:

```
set the preLoadMode of castLib "Buttons" = TRUE
```

- **Movie Properties** dialog box

● **preLoadMovie**

Syntax: `preLoadMovie whichMovie`

This command preloads the cast members associated with the first frame of the specified movie. Preloading a movie helps it start faster when it is started by a `go to movie` or `play movie` command.

Example:

This statement preloads the movie Introduction:

```
preLoadMember "Introduction"
```

- **preLoadRAM**

Syntax: the preLoadRAM

This property specifies the amount of RAM that can be used for preloading a digital video. It can be set and tested.

This is useful for managing memory, so that digital video cast members are not given more than a certain limit of memory, and other types of cast members can still be preloaded. When the preLoadRAM is FALSE, all available memory can be used for preloading digital video cast members.

Example:

This statement allocates the amount of RAM available for preloading to 3 times the size of the cast member Interview:

```
set the preLoadRAM to 3 * (the size of member "Interview")
```

- **loop** and **next** keywords

- **previous**

- **go_previous**

● **printFrom**

Syntax: `printFrom fromFrame [, toFrame] [, reduction]`

This command prints whatever is displayed on the stage in each frame starting at the frame specified by *fromFrame*. Optionally you can supply the *toFrame*, and the reduction (100, 50, or 25 percent).

When printing at less than 100 percent, the document prints as a bitmap, so text does not print as sharply as it would at full size.

Example:

This statement prints what is on the stage in every frame starting at frame 1:

```
printFrom 1
```

This statement prints what is on the stage in every frame from the marker Intro to the marker Tale. The reduction is 50 percent:

```
printFrom label("Intro"), ("Tale"), 50
```

- **property**

Syntax: `property [property1][, property2][, property3] [...]`

This keyword declares that the properties specified by *property1*, *property2*, and so on are property variables. Property variables, which are used in parent scripts, serve the same purpose as instance variables in Xtras and XObjects.

You declare property variables at the beginning of the parent script. You can access them from outside the parent script by using the `the` operator.

Example:

This statement allows each child object created from a single parent script to have its own location and velocity setting:

```
property location, velocity
```

- **ancestor** property, **child-parent scripts**

- **puppet of sprite**

Syntax: the puppet of sprite *whichSprite*

This sprite property determines whether the sprite specified by the integer expression *whichSprite* is a puppet.

A puppet sprite is controlled by Lingo instead of the score. For example, Lingo can switch the cast member assigned to a sprite or turn on and off whether the sprite is moveable. For more information on using puppets, see Chapter 4, "Manipulating the Score from Lingo," in *Learning Lingo*.

The sprite channel must contain a sprite before you can make the channel a puppet.

Making the sprite channel a puppet lets you control any of the sprite properties--such as `memberNum of sprite`, `locH of sprite`, and `width of sprite`--from Lingo.

Setting the `puppet of sprite` property is equivalent to using the `puppetSprite` command. For example, the statement:

```
set the puppet of sprite 1 to TRUE
```

has the same effect as:

```
puppetSprite 1, TRUE
```

The `puppetSprite` property can be tested and set. The default value is FALSE.

Example:

This statement makes the sprite numbered `i + 1` a puppet:

```
set the puppet of sprite (i + 1) to TRUE
```

This statement records whether sprite 5 is a puppet by assigning the value of the `puppet of sprite` to the variable. When sprite 5 is a puppet, `isPuppet` is set to TRUE. When sprite 5 is not a puppet, `isPuppet` is set to FALSE:

```
put the puppet of sprite 5 into isPuppet
```

- **puppetSprite command, Puppeting**

- **puppetPalette**

Syntax: `puppetPalette whichPalette [, speed] [, nFrames]`

This command causes the palette channel to act as a puppet. When the palette channel is a puppet, Lingo can override the palette setting in the palette channel of the score and assign palettes to the movie.

The `puppetPalette` command sets the current palette to the palette cast member specified by the expression *whichPalette*. If *whichPalette* evaluates to a string, it specifies the cast name of the palette. If *whichPalette* evaluates to an integer, it specifies the cast number of the palette.

Optionally, you can fade in the palette by replacing *speed* with an integer expression, with 1 being slowest and 60 being fastest. You can also fade in the palette over several frames by replacing *nFrames* with an integer expression for the number of frames.

A puppet palette remains in effect until you turn it off with the command `puppetPalette 0`. No subsequent palette changes in the score are obeyed when the puppet palette is in effect.

Example:

This statement makes Rainbow the movie's palette:

```
puppetPalette "Rainbow"
```

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")
```

- **Puppeting**

- **puppetSound**

Syntax:

```
puppetSound whichChannel, "whichCastMember"  
puppetSound "whichCastMember"  
puppetSound member "whichCastMember"  
puppetSound 0
```

This command makes the sound channel a puppet and plays the sound cast member specified by *whichCastMember*. When the sound is a puppet, Lingo can override any sounds assigned in the score's sound channels.

For internal sound cast members, you can specify a sound channel by replacing *whichChannel* with a channel number. However, you can't specify a channel for linked sound cast members. Linked sound cast members always play in sound channel 1.

The sound starts playing after the playback head moves or the `updateStage` command is executed. The statement `puppetSound 0` stops a sound from playing. It also turns off the puppet status of the sound and returns control of the sound to the sound channel in the score. Use `puppetSound` to restore 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.

Example:

This statement plays the sound Wind under control of Lingo:

```
puppetSound "Wind"
```

- **sound fadeIn, sound fadeOut, sound playFile, sound stop commands, **Puppeting****

- **puppetSprite**

Syntax: `puppetSprite whichSprite, state`

This command controls whether the sprite specified by *whichSprite* is a puppet. When a sprite is a puppet, any sprite property can be controlled by Lingo instead of the score. For example, Lingo can switch the cast member assigned to a sprite or turn on and off whether the sprite is moveable.

- When *state* is TRUE, Lingo controls the sprite and the score is ignored.
- When *state* is FALSE, the sprite is controlled by the score.

The initial properties of the puppet are taken from whatever sprite is in the channel when the `puppetSprite` command is executed. Subsequent control of the sprite properties through Lingo can change these properties.

The channel must contain a sprite when you use the `puppetSprite` command.

You must provide the command `puppetSprite whichSprite, FALSE` when you are finished with your puppet; otherwise unpredictable results can occur when the playback head returns to sprites in frames that aren't intended to be puppets.

For more information on using puppets, see Chapter 4 of *Learning Lingo*.

Examples:

This statement makes the sprite in channel 15 a puppet:

```
puppetSprite 15, TRUE
```

This statement removes the puppet condition from the sprite in the channel numbered *i* + 1:

```
puppetSprite i + 1, FALSE
```

- **backColor of sprite, bottom of sprite, memberNum of sprite, constraint of sprite, cursor, foreColor of sprite, height of sprite, ink of sprite, left of sprite, lineSize of sprite, locH of sprite, locV of sprite, puppet of sprite, right of sprite, stretch of sprite, top of sprite, type of sprite, and width of sprite** sprite properties; **puppetSprite** property, **Puppeting**

- **puppetTempo**

Syntax: `puppetTempo framesPerSecond`

This command causes the tempo channel to act as a puppet. When the tempo channel is a puppet, Lingo can override the tempo setting in the score and change the tempo assigned to the movie.

The `puppetTempo` command sets the tempo to the number of frames specified by *framesPerSecond*. The maximum frames per second is 60.

You do not need to turn off the puppet tempo condition to have subsequent tempo changes in the score take effect.

Examples:

This statement set the movie's tempo to 30 frames per second:

```
puppetTempo 30
```

This statement increases the movie's old tempo by ten frames per second:

```
puppetTempo oldTempo + 10
```

- **Puppeting**

● puppetTransition

Syntax: puppetTransition member *whichCastMember*
puppetTransition member *castmemberReference*
puppetTransition *whichTransition* [, *time*]-
[, *chunkSize*] [, *changeArea*]

This 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 one of the following values:

Code Transition

- 01** Wipe right
- 02** Wipe left
- 03** Wipe down
- 04** Wipe up
- 05** Center out, horizontal
- 06** Edges in, horizontal
- 07** Center out, vertical
- 08** Edges in, vertical
- 09** Center out, square
- 10** Edges in, square
- 11** Push left
- 12** Push right
- 13** Push down
- 14** Push up
- 15** Reveal up
- 16** Reveal up, right
- 17** Reveal right
- 18** Reveal down, right
- 19** Reveal down
- 20** Reveal down, left
- 21** Reveal left
- 22** Reveal up, left
- 23** Dissolve, pixels fast *
- 24** Dissolve, boxy rectangles
- 25** Dissolve, boxy squares
- 26** Dissolve, patterns
- 27** Random rows
- 28** Random columns
- 29** Cover down
- 30** Cover down, left
- 31** Cover down, right
- 32** Cover left
- 33** Cover right
- 34** Cover up
- 35** Cover up, left

36	Cover up, right
37	Venetian blinds
38	Checkerboard
39	Strips on bottom, build left
40	Strips on bottom, build right
41	Strips on left, build down
42	Strips on left, build up
43	Strips on right, build down
44	Strips on right, build up
45	Strips on top, build left
46	Strips on top, build right
47	Zoom open
48	Zoom close
49	Vertical blinds
50	Dissolve, bits fast *
51	Dissolve, pixels *
52	Dissolve, bits *

Transitions marked with an asterisk (*) in the table will not work on monitors that are set to 32 bits.

Replace *time* with the number of 1/4 seconds used to complete the transition. The minimum is 0; the maximum is 120 (30 seconds). Replace *chunkSize* with the number of pixels in each chunk of the transition. The minimum is 1; the maximum is 128. Smaller chunk sizes give smoother transitions but are slower.

There is no direct relationship between a low time and a fast transition. The actual speed of the transition depends on the relation of *chunkSize* and *time*. As an example, if the *chunkSize* is one pixel, the transition takes a long time no matter how low the time, because the computer has to do a lot of work. To make transitions occur faster you should use a larger chunk size, instead of setting a shorter time.

Replace *changeArea* with a value that determines whether the transition occurs only in the changing area. The *changeArea* is an area within which sprites have changed.

- To have the transition occur only in the areas that change, replace *changeArea* with FALSE, which is the default setting.
- To have the transition occur over the entire stage, replace *changeArea* with TRUE.

Example:

This statement performs a wipe from right transition. Because no value is specified for *changeArea*, the transition occurs only on the changing area, which is the default:

```
puppetTransition 1
```

This statement performs a wipe from right transition that lasts 1 second, has a chunk size of 20, and occurs over the entire stage:

```
puppetTransition 2, 4, 20, TRUE
```

- **Puppeting**

● **purgePriority of member**

Syntax: the purgePriority of member *whichCastMember*

This cast member property specifies the purge priority of the cast member specified by *whichCastMember*.

Cast members' purge priorities determine the priority that Director follows when choosing which cast members to delete from memory when memory is full. The higher the purge priority, the more likely that the cast member is deleted. The following `purgePriority` settings are available:

- 0**--Never purge
- 1**--Purge last
- 2**--Purge next
- 3**--Purge normal

Normal allows Director to purge cast members from memory at random (just as Director 3.1 did). Next, Last, and Never allow users some control over purging. However, if you set a lot of cast members to Last or Never, your movie may run out of memory.

Setting `purgePriority of member` for cast members is useful for managing memory when the size of the movie's cast exceeds the available memory. As a general rule, you can minimize pauses while the movie loads cast members by assigning a low purge priority to cast members that are frequently used in the course of the movie. This reduces the number of times that Director reloads the cast member when the movie plays.

Example:

This statement sets the purge priority of cast member Background to 2, which makes it one of the first cast members to be purged when memory is needed:

```
set the purgePriority of member "Background" to 2
```

- **put**

Syntax: `put expression`

This command evaluates the expression specified by *expression* and displays the result in the message window. This can be used as a debugging tool by tracking the values of variables as the movie plays.

Examples:

This statement displays the time in the message window:

```
put the time
-- "9:10 AM"
```

This statement displays the value assigned to the variable `vBid` in the message window:

```
put vBid
-- "Johnson"
```

- **put...after, put...before, and put...into** commands

- **put...after**

Syntax: `put expression after chunkExpression`

This command evaluates a Lingo expression, converts the value to a string, and inserts the resulting string after a specified chunk in a container. (If *chunkExpression* specifies a nonexistent target chunk, the string value is inserted as appropriate into the container.) The previous contents of the container remain.

Chunk expressions can refer to any character, word, item, or line in any container of characters. Containers include fields (field cast members) and variables that hold strings, and specified characters, words, items, lines, and ranges in containers.

Example:

This statement adds the string "fox dog cat" after the contents of the variable `animalList`:

```
put "fox dog cat" after animalList
```

- **char...of, item...of, line...of, put...before, put...into, and word...of** chunk expression keywords

- **put...before**

Syntax: `put expression before chunkExpression`

This command evaluates a Lingo expression, converts the value to a string, and inserts the resulting string before a specified chunk in a container. (If *chunkExpression* specifies a nonexistent target chunk, the string value is inserted as appropriate into the container.) The previous contents of the container remain.

Chunk expressions can refer to any character, word, item, or line in any container. Containers include fields (field cast members) and variables that hold strings, and specified characters, words, items, lines, and ranges in containers.

Example:

These statements set the variable named `animalList` to the string "fox dog cat" and then insert the word `elk` before the second word of the list:

```
put "fox dog cat" into animalList
put "elk " before word 2 of animalList
```

The result is the string "fox elk dog cat".

- **char...of**, **item...of**, **line...of**, **put...after**, **put...into**, and **word...of** chunk expression keywords

- **put...into**

Syntax: `put expression into variable`
 `put expression into chunkExpression`

This command has two different usages.

The first usage evaluates a Lingo expression and stores its value in a local, global, property or instance variable. The value can be an integer, a floating point number, a string, an object, or a symbol; it resides unchanged in the variable.

The second usage evaluates a Lingo expression, converts the value to a string, and uses the resulting string to replace a specified chunk in a container. (If *chunkExpression* specifies a nonexistent target chunk, the string value is inserted as appropriate into the container.)

Chunk expressions can refer to any character, word, item, or line in any container. Containers include field cast members and variables that hold strings, and specified characters, words, items, lines, and ranges in containers.

Note: In Lingo, you can use `set...to` and `set...=` as well as `put...into` for variable assignments. However, unlike `set`, you can't use `put` to specify values of properties.

Example:

This statement sets the variable `x` to the square root of 2:

```
put sqrt(2.0) into x
```

The result is 1.4142.

- **char...of, item...of, line...of, put...after, put...before, and word...of** chunk expression keywords; **set to** command

- **quickTimePresent**

Syntax: the quickTimePresent

This function determines whether the QuickTime extension is currently loaded into memory.

- When the extension is present, the `quickTimePresent` function is TRUE (1).
- When the extension is not present, the `quickTimePresent` function is FALSE (1).

Example:

This statement determines whether the QuickTime extension is in memory and sets the `movieRate` for a QuickTime sprite indicated by `QTSprite`:

```
if the quickTimePresent = 1 then -  
    set the movieRate of sprite QTSprite
```

- **quit**

Syntax: quit

This command exits from Director or a projector to the Finder.

Example:

This statement has the computer exit to the Desktop when the user presses Control+Q:

```
if the key = "q" and the commandDown then quit
```

- **restart** and **shutDown** commands

• QUOTE

Syntax: QUOTE

This character constant represents the quote character. It is needed to refer to the literal quote character in a string, since the quote character itself is used by Lingo scripts to delimit strings.

Example:

This statement inserts quote characters in the string:

```
put "Can you spell" && QUOTE & "Macromedia" →  
    & QUOTE & "?"
```

The result is quotes around the word Macromedia, as in the following string:

Can you spell "Macromedia"?

- **ramNeeded**

Syntax: `ramNeeded (firstFrame, lastFrame)`

This function determines, in bytes, the memory needed to display a range of frames. For example, you can test the size of frames containing 32-bit artwork. If the `ramNeeded` is larger than the `freeBytes`, then go to frames containing 8-bit artwork. Divide by 1024 to convert bytes to kilobytes (K).

Example:

This statement sets the variable `frameSize` to the number of kilobytes needed to display frames 100 to 125 of the movie:

```
put ramNeeded (100, 125) into frameSize
```

This statement determines whether the memory needed to display frames 100 to 125 is more than the available memory and branches to a movie using cast members that have lower color depth if it is:

```
if ramNeeded (100, 125) > freeBytes then ↵  
  play frame "8-bit"
```

- **freeBytes** function; **size of member** cast member property

• random

Syntax: `random(integerExpression)`

This function returns a random integer from 1 to the value specified by *integerExpression*.

The `random` function is useful when you want to randomly vary values in a movie. Some possible uses are varying the path through a game, assigning random numbers, or changing the color or position of sprites.

Examples:

This statement assigns random values to the variable `diceRoll`:

```
put random(6) + random(6) into diceRoll
```

This statement randomly changes the foreground color of sprite 10:

```
set the foreColor of sprite 10 = random(256) - 1
```

This handler randomly chooses which of two movie segments to play in the "Noh Tale":

```
on selectMovie
  if random(2) = 2 then play frame "11a"
  else
    play frame "11-b" of movie "NT.Other Movie"
  end if
end
```

● **randomSeed**

Syntax: `the randomSeed`

This property specifies seed for generating random numbers. Using the same seed produces the same sequence of random numbers.

The `randomSeed` property can be tested and set.

Example:

This statement displays the random seed number in the message window:

```
put the randomSeed
```

- **rect**

Syntax: `rect(left, top, right, bottom)`
 `rect(point1, point2)`

This function has two uses:

- When you use four arguments, the `rect` function defines a rectangle that has the sides specified by *left*, *top*, *right*, and *bottom*. The *left* and *right* values specify numbers of pixels from the left edge of the stage. The *top* and *bottom* values specify numbers of pixels from the top of the stage.
- When you use two arguments, the `rect` function defines a rectangle that encloses the points specified by *point1* and *point2*.

Examples:

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:

```
put rect(100, 150, 300, 400) into newArea
```

This statement sets the variable `newArea` to the rectangle defined by the points `firstPoint` and `secondPoint`. The coordinates of `firstPoint` are (100, 150); the coordinates of `secondPoint` are (300, 400). Note that this statement creates the same `rect` as the rectangle created in the previous example:

```
put rect(firstPoint, secondPoint)
```

- **point** function

- **the rect of member**

Syntax: `the rect of member whichCastmember`

This function gives the left, top, right, and bottom coordinates for the rectangle of any graphic cast membersuch as a bitmap, shape, movie, or digital video. The coordinates are returned as a rect.

The `rect of member` property can be tested but not set.

Example:

This statement displays the coordinates of bitmap cast member 20:

```
put the rect of member 20
```

• **rect of sprite**

Syntax: `the rect of sprite whichSprite`

This function gives the left, top, right, and bottom coordinates for the rectangle of any graphic sprite such as a bitmap, shape, movie, or digital video. The coordinates are returned as a rect.

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

Example:

This statement displays the coordinates of bitmap sprite 20:

```
put the rect of sprite 20
```

- **rect of window**

Syntax: the rect of window *whichWindow*

This window property determines the left, top, right, and bottom coordinates of the window specified by *whichWindow*. The coordinates are given as a rect.

The `rect of window` property can be tested and set.

Example:

This statement displays the coordinates of the window Control Panel:

```
put the rect of window "Control Panel"
```

● **regPoint of member**

Syntax: the regPoint of member *whichCastMember*

This cast member property specifies the registration point of a bitmap cast member. The registration points are listed as horizontal and vertical coordinates in a point that has the form `point (horizontal, vertical)`.

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

Example:

This statement displays the registration points of the bitmap cast member `Desk` in the message window:

```
put the regPoint of member "Desk"
```

This statement changes the registration points of the bitmap cast member `Desk` to the values in the list:

```
set the regPoint of member "Desk" = ↵  
point(300, 400)
```

- **repeat while**

Syntax: repeat while *testCondition*
 [*statements...*]
 end repeat

This keyword structure repeatedly executes the statements as long as the condition specified by *testCondition* is TRUE. Some possible uses for this structure are for Lingo that continues to read strings until the end of a file is reached, checks items until the end of a list is reached, or repeatedly performs an action until the user clicks or releases the mouse button.

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
```

- **exit, exit repeat, and repeat with** keywords

- **repeat with**

Syntax: repeat with *counter* = *start* to *finish*
 [*statements...*]
 end repeat

This keyword structure executes the Lingo specified by *statements* the number of times specified by *counter*. The value of *counter* is the difference between the value specified by *start* and the value specified by *finish*. The counter is incremented by 1 each time Lingo goes through the repeat loop.

The `repeat with` structure is useful for repeatedly applying the same effect to a series of puppets or calculating a series of numbers, such as a number to some exponent.

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
```

- **exit, exit repeat, and repeat while** keywords

- **repeat with...down to**

Syntax: repeat with *variable* = *startValue* down to *endValue*

This keyword counts down by increments of 1 from *startValue* to *endValue*.

Example:

This handler contains a repeat loop that counts down from 20 to 15:

```
on countdown
  repeat with i = 20 down to 15
    set the memberNum of sprite 6 to (10 + i)
    updateStage
  end repeat
```

- **repeat with...in list**

Syntax: repeat with *variable* in *someList*

This keyword assigns successive values from the specified list to the variable.

Example:

This statement displays four values in the message window:

```
repeat with x in [1, 2, 3, 4]
  put i
end repeat
```


- **restart**

Syntax: restart

This command restarts the Macintosh computer. It is equivalent to choosing Restart in the Macintosh Finder's Special menu. The restart command has no effect in Windows.

Example:

This statement restarts the Macintosh when the user presses Command-period:

```
if the key = "r" and the commandDown then restart
```

- **quit** and **shutDown** commands

- **result**

Syntax: the result

This function gives the value of the return expression in 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.

Examples:

The following handler returns a random roll for two dice:

```
on diceRoll
    return random(6) + random(6)
end diceRoll
```

The two statements:

```
diceRoll
    put the result into roll
```

are equivalent to this statement:

```
put diceRoll() into roll
```

Note that

```
put diceRoll into roll
```

does not call the handler because there are no parentheses following `diceRoll`; `diceRoll` here is considered a variable reference.

- **return** keyword

● **RETURN constant**

Syntax: RETURN

This character constant represents the Return key.

Example:

This statement has a paused movie continue when the user presses the Return key:

```
if the key = RETURN then continue
```

This statement uses the Return character constant to insert a return between two lines in an alert:

```
alert "Last line in the file." & RETURN & ↵  
      "Click OK to exit."
```

In Windows, writing to a file requires an additional line feed character at the end of each line. This statement creates a two-character string named CRLF that provides the additional line feed:

```
put Return&numToChar(10) into CRLF
```

- **return**

Syntax: `return expression`

This keyword is used in handlers and methods that return values. It returns the value of *expression* and exits from a handler or method. The expression can be an integer, floating point number, string, object, or symbol.

When calling a handler or method that serves as a user-defined function and has a return value, you must use parentheses around the argument list. This is necessary even when there are no arguments, as in the `diceRoll` function handler discussed under the entry `the result`.

Example 1:

The following handler returns the greater of two expressions:

```
on max a, b
  if a > b then
    return a
  else
    return b
  end if
end max
```

If 3 and 7 were used for `a` and `b`, the result would be as follows:

```
put max(3, 7)
-- 7
```

Example 2:

In Windows, this statement creates a two-character string named `CRLF` that provides the additional line feed:

```
put RETURN&numToChar(10) into CRLF
```

- **result** keyword

- **right of sprite**

Syntax: the right of sprite *whichSprite*

This sprite property indicates the number of pixels that the right edge of the sprite specified by *whichSprite* is from the left edge of the stage.

The `right of sprite` property can be tested, but not set directly. The right horizontal coordinate of a sprite can be set using the `spriteBox` command.

Example:

This statement calls the handler `offRightEdge` when the right edge of sprite 3 is past the right edge of the stage:

```
if the right of sprite 3 > (the stageRight -  
- the stageLeft) then offRightEdge
```

Note: Sprite coordinates are expressed relative to the upper-left corner of the stage.

- **bottom of sprite, height of sprite, left of sprite, locH of sprite, locV of sprite, top of sprite, and width of sprite** sprite properties; **spriteBox** command

• the rightMouseDown

Syntax: the rightMouseDown

This system property indicates the current state of the right mouse button on a Windows computer. On the Macintosh, if the `emulateMultiButtonMouse` property is set to TRUE, this property indicates whether the user is pressing the mouse button and the Control key.

- When `rightMouseDown` is TRUE, the right mouse button (Windows) or the mouse button and Control key (Macintosh) is being pressed.
- When `rightMouseDown` is FALSE, the right mouse button (Windows) or the mouse button and Control key (Macintosh) is not being pressed.

Example:

This statement checks whether the right mouse button is being pressed and plays the sound Oops if it is:

```
if the rightMouseDown then puppetSound 2, "Oops"
```

• the rightMouseUp

Syntax: the rightMouseUp

This system property indicates the current state of the right mouse button on a Windows computer. On the Macintosh, if the `emulateMultiButtonMouse` property is set to TRUE, this property indicates whether the user is pressing the mouse button and the Control key.

- When `rightMouseUp` is TRUE, the right mouse button (Windows) or the mouse button and Control key (Macintosh) is currently not being pressed.
- When `rightMouseUp` is FALSE, the right mouse button (Windows) or the mouse button and Control key (Macintosh) is currently being pressed.

Example:

This statement checks whether the right mouse button is released and plays the sound Click Me if it is:

```
if the rightMouseDown then puppetSound 2, "Oops"
```

● **rollOver**

Syntax: `rollOver (whichSprite)`

This function indicates whether the cursor is currently over the bounding rectangle of the sprite specified by *whichSprite*.

- When `rollOver` is TRUE (1), the cursor is currently over the sprite.
- When `rollOver` is FALSE (0), the cursor is not currently over the sprite.

The `rollOver` function is typically used in frame scripts. It is useful for creating handlers that perform an action when the user places the cursor over a specific sprite. It can also simulate additional sprite channels by splitting the stage into regions that send the playback head to a different frame that subdivides the region for the available sprite channels.

When the cursor is over the location of a sprite that has been removed, the rollover still occurs. Avoid this problem by not doing rollovers over these locations or by relocating the sprite up above the menu bar before deleting it.

Example:

This statement changes the content of field cast member Message to "This is the place." when the cursor is over sprite 6:

```
if rollOver(6) then -  
  put "This is the place." into field "Message"
```

This handler sends the playback head to different frames when the cursor is over certain sprites on the stage. The three sprites in this case could be invisible rectangles in different parts of the stage. Putting additional subdivisions within each of the frames lets you work with more sprites than there are available channels:

```
on enterFrame  
  if rollover(1) then go to frame "Left"  
  if rollover(2) then go to frame "Middle"  
  if rollover(3) then go to frame "Right"  
end enterFrame
```

- **mouseCast** function

● romanLingo

Syntax: `the romanLingo`

This property specifies whether Lingo uses a single-byte or double-byte interpreter.

- When the `romanLingo` is TRUE, Lingo uses a single-byte interpreter.
- When the `romanLingo` is FALSE, Lingo uses a double-byte interpreter.

The Lingo interpreter is faster with single-byte character sets. Some versions of Macintosh system software--Japanese, for example--use a double-byte character set. U.S. system software uses a single-byte character set. Normally, `the romanLingo` is set when starting up Director and is determined by the local version of Macintosh system software.

If you are using a non-roman script system but don't use any double-byte characters in your script, set this property to TRUE to get faster execution of your Lingo scripts.

Example:

This statement sets the `romanLingo` to TRUE, which has Lingo use a single-byte character set:

```
set the romanLingo to TRUE
```

- **sampleRate of member**

Syntax: the sampleRate of member *whichCastmember*

This sound cast member property determines the sample rate of the specified cast member. The result is in kHz. This property can be tested but not set.

Example:

This statement checks the sample rate of the sound cast member Voice Over and assigns the value to the variable soundRate:

```
put the sampleRate of member "Voice Over" into soundRate
```

- **sampleSize of member**

Syntax: the sampleSize of member *whichCastmember*

This sound cast member property determines the sample size of the specified cast member. The result is usually 8- or 16-bit. This property can tested but not set.

Example:

This statement checks the sample size of the sound cast member Voice Over and assigns the value to the variable soundSize:

```
put the sampleSize of member "Voice Over" into soundSize
```

- **save castLib**

Syntax: `save castLib whichCast {, pathname:newFileName}`

This command saves any changes to the cast. Including the optional *pathname:newFileName* parameter saves the file to a new file that uses the *pathname:newFileName* parameter as its file name. (When the *pathname:newFileName* parameter isn't included, changes to the cast are saved in the cast's original file.

Example:

This statement has Director save the revised version of the cast Buttons in the new file UpdatedButtons in the same folder:

```
save castLib "Buttons" , "UpdatedButtons"
```

● saveMovie

Syntax: `saveMovie [pathname:filename]`

This command saves the current movie. Including the optional parameter saves the movie to the file specified by *pathname:filename*.

Example:

This statement saves the current movie to the file Update:

```
saveMovie "Update"
```

- **score**

Syntax: the score

This movie property determines which score is associated with the current movie. The score must be a film loop cast member. The property can be tested and set.

Example:

This statement assigns the film loop cast member "Waterfall" to the score of the current movie.

```
set the score to media of member "Waterfall"
```

- **scoreColor of sprite**

Syntax: the scoreColor of sprite *whichSprite*

This sprite property indicates the score color assigned to the sprite specified by *whichSprite*. The possible values correspond to color chips 0 to 5 in the current palette.

The `scoreColor of sprite` property can be tested but not set.

Example:

This statement has the message window display the value for the score color assigned to sprite 7:

```
put the scoreColor of sprite 7
```

● scoreSelection

Syntax: `the scoreSelection`

This movie property determines which channels are selected in the score window. The selection is in a list consisting of the starting channel number, the ending channel number, the starting frame number, and the ending frame number. Specify sprite channels by their channel number. Use the following numbers to specify the other channels:

To specify:	Use:
Frame script channel	0
Sound channel 2	-1
Sound channel 1	-2
Transition channel	-3
Palette channel	-4
Tempo channel	-5

You can select discontinuous channels, but you can't select discontinuous frames. This property can be tested and set.

Examples:

This statement selects sprite channels 15 through 25 in frames 100 through 200:

```
set the scoreSelection = [[15, 25, 100, 200]]
```

This statement selects sprite channels 15 through 25 and sprite channels 40 through 50 in frames 100 through 200:

```
set the scoreSelection = [[15, 25, 100, 200] , [40, 50, 100, 200]]
```

This statement selects the frame script in frames 100 through 200:

```
set the scoreSelection = [[0, 0, 100, 200]]
```


- **script of menuItem**

Syntax: the script of menuItem *whichItem* of menu *whichMenu*

This menu item property determines which Lingo statement is executed when the specified menu item is selected. The *whichItem* expression can be either a menu item name or a menu item number; the *whichMenu* expression can be either a menu name or a menu number.

When the menu is installed, the script is set to the text following the "¼" character in the menu definition.

The script property can be tested and set.

Example:

This statement makes the handler named `goHandler` the handler that is executed when the user chooses the command Go from the custom menu Control:

```
set the script of menuItem "Go" of menu "Control" to  
to "goHandler"
```

- **checkMark of menuItem** and **enabled of menuItem** properties; **installMenu** command; **menu** keyword

- **scriptNum of sprite**

Syntax: `scriptNum of sprite whichSprite`

This sprite property indicates the number of the script assigned to the sprite specified by *whichSprite*.

The `scriptNum of sprite` property can be tested, but not set.

Example:

This statement displays the number of the script attached to sprite 4:

```
put the scriptNum of sprite 4
```

- **scriptsEnabled of member**

Syntax: the `scriptsEnabled` of member *whichCastmember*

This movie cast member property determines whether scripts in a linked movie are enabled.

- When the `scriptsEnabled` of member is TRUE (1), the linked movie's scripts are enabled.
- When the `scriptsEnabled` of member is FALSE (0), the linked movie's scripts aren't enabled.

This property is available for Director movie cast members only. While it can be tested and set for Director movies, it can't be tested or set for other cast members.

Example:

This statement turns off scripts in the linked movie Jazz Chronicle:

```
set the scriptsEnabled of member "Jazz Chronicle" = FALSE
```

● **scriptText of member**

Syntax: the scriptText of member *whichCastmember*

This cast member property indicates the content of the script, if any, assigned to the cast member specified by *whichCastmember*.

Movies lose their values for the scriptText of member property when they are converted into projectors. Therefore, the movie's values for the scriptText of member can't be used by a projector. However, Director can set new values for the scriptText of member inside the projector.

The scriptText of member property can be tested and set.

Example:

This statement makes the contents of field cast member 20 the script of cast member 30:

```
set the scriptText of member 30 = the text of member 20
```

- **scriptType of script**

Syntax: the scriptType of member *whichScript*

This script cast member property indicates the specified script's type. Possible values are #MOVIE, #SPRITE, #FRAME, and #PARENT.

Example:

This statement makes script member Main Script a movie script:

```
set the scriptType of member "Main Script" to #movie
```

- **scrollByLine**

Syntax: `scrollByLine member whichCastmember, amount`

This command scrolls the specified field cast member up or down by the number of lines specified in *amount*.

- When *amount* is positive, the field scrolls down.
- When *amount* is negative, the field scrolls up.

Examples:

This statement scrolls the field cast member Today's News down five lines:

```
scrollByLine member "Today's News", 5
```

This statement scrolls the field cast member Today's News up five lines:

```
scrollByLine member "Today's News", -1
```

- **scrollByPage**

Syntax: `scrollByPage member whichCastMember, amount`

This command scrolls the specified field cast member up or down by the number of pages specified in *amount*.

- When *amount* is positive, the field scrolls down.
- When *amount* is negative, the field scrolls up.

Examples:

This statement scrolls the field cast member Today's News down one page:

```
scrollByPage member "Today's News", 1
```

This statement scrolls the field cast member Today's News up one page:

```
scrollByPage member "Today's News", -1
```

● **scrollTop of member**

Syntax: the scrollTop of member *whichCastmember*

This property of text and field cast members determines the distance, in pixels, from the top of a field cast member to the top of the field that is currently visible in the scrolling box. By changing the value for the `scrollTop of member` while the movie plays, you can change the section of the field that appears in the scrolling field.

Example:

This repeat loop makes the field Credits appear to scroll by continuously increasing the value of the `scrollTop of member`:

```
set the scrollTop of member "Credits" = 1
repeat with count = 1 to 150
  set the scrollTop of member "Credits" = the scrollTop of member "Credits" + count
end repeat
```


● **searchCurrentFolder**

Syntax: `the searchCurrentFolder`

This function determines whether Director searches the current folder when searching filenames.

- When the `searchCurrentFolder` function is TRUE (1), Director searches the current folder when resolving filenames.
- When the `searchCurrentFolder` function is FALSE (0), Director does not search the current folder when resolving filenames.

The `searchCurrentFolder` function can be tested and set.

Examples:

This statement has the message window display whether the `searchCurrentFolder` function is on.

```
put the searchCurrentFolder
```

The result is the number 1, which is the numeric equivalent of TRUE.

This statement sets the `searchCurrentFolder` function to TRUE, which has Director search the current folder when resolving filenames:

```
set the searchCurrentFolder to TRUE
```

- **searchPath**

Syntax: the searchPath

This function provides a list of the pathnames that are searched when Director resolves filenames. When Director cannot find the file in the current folder, it searches for it in the folders listed in the `searchPath`.

The `searchPath` content is a regular list that you can handle the same as any other list by using commands such as `add`, `addAt`, `append`, `deleteAt`, and `setAt`. Items in the list are separated by commas. Trailing colons and backslashes are allowed but not necessary.

Adding a large number of paths to the `searchPath` slows searching. Try to minimize the number of paths in the list.

The `searchPath` function can be tested and set.

Example 1:

This statement displays the pathnames that Director searches when resolving filenames:

```
put the searchPath
```

Example 2:

This statement assigns two folders to the `searchPath` in Windows. This version includes optional trailing backslashes:

```
set the searchPath = ["c:\director\projects\", ↵ d:\cdrom\sources\"]
```

This statement is the same, except that trailing backslashes have been omitted:

```
set the searchPath = ["c:\director\projects", d:\cdrom\sources"]
```

Example 3:

This statement assigns two folders to the `searchPath` on a Macintosh. This version includes optional trailing colons:

```
set the searchPath = ["hard drive:director:projects:", ↵ cdrom:sources:"]
```

This statement is the same, except that trailing colons have been omitted:

```
set the searchPath = ["hard drive:director:projects"↵ cdrom:sources"]
```

- **searchCurrentFolder** function

● searchPaths

Syntax: `the searchPaths`

This global property is a list of paths that Director searches. Each item in the list is a fully qualified pathname as it appears on the current platform at run time.

The value of `the searchPaths` value is a regular list that you can handle the same as any other list by using commands such as `add`, `addAt`, `append`, `deleteAt`, and `setAt`.

Adding a large number of paths to `the searchPaths` slows searching. Try to minimize the number of paths in the list.

Example:

These statements have Director look inside a folder named Sounds, which is in the same folder as the current Director movie:

```
put the moviePath & "Sounds" into soundsPath
add the searchPaths, soundPath
```

- **selection**

Syntax: `the selection`

This function returns a string containing the highlighted portion of the current editable field. It is useful for testing what a user has selected in a field.

The `selection` function only determines which string of characters are selected; you cannot use `the selection` to select a string of characters.

Example:

This statement checks whether any characters are selected and displays the alert "Please select a word." if none are:

```
if the selection = EMPTY then ↵  
    alert "Please select a word."
```

- **selStart** and **selEnd** properties

● selection of castLib

Syntax: the selection of castLib *whichCast*

```
set the selection of castLib whichCast = [startMember1,  
endMember1] , { [startMember2, endMember2] ,  
[startMember3, endMember3]... }
```

This cast property determines which cast members are selected in the specified cast window. The specified range appears as a list of the starting and ending cast member numbers for the selected range. You can specify more than one selection by specifying additional ranges of cast members. (Specifying more than one selection is done dragging while pressing the Control key (Windows) or the Command key (Macintosh). This property can be tested and set.

Example:

This statement selects cast members 1 through 10 in castLib number 1:

```
set the selection of castLib 1 = [1, 10]
```

This statement selects cast members 1 through 10 and 30 through 40 in castLib number 1:

```
set the selection of castLib 1 = [1, 10], [30,40]
```

- **selEnd**

Syntax: the selEnd

This field property specifies the ending character of a selection. It is used with the `selStart` to determine a selection from the currently editable field, counting from the beginning character.

The `selEnd` field property can be tested and set, and the default value is 0.

Examples:

These statements select "cde" from the field "abcdefg":

```
set the selStart to 3
```

```
set the selEnd to 5
```

This statement calls the handler `noSelection` when the `selEnd` is the same as the `selStart`:

```
if the selEnd = the selStart then noSelection
```

This statement makes a selection 20 characters long:

```
set the selEnd to the selStart + 20
```

- **editable of sprite** and **hilite** commands; **selection** function; **selStart** and **text of member** properties

- **selStart**

Syntax: the selStart

This field property specifies the starting character of a selection. It is used with the selEnd to determine a selection from the currently editable field, counting from the beginning character.

The selStart field property can be tested and set. The default value is 0.

Examples:

These statements select "cde" from the field "abcdefg":

```
set the selStart to 3
```

```
set the selEnd to 5
```

This statement calls the handler noSelection when the selEnd is the same as the selStart:

```
if the selEnd = the selStart then noSelection
```

This statement makes a selection 20 characters long:

```
set the selEnd to the selStart + 20
```

- **editable of sprite** and **hilite** commands; **selection** function; **selEnd** and **text of member** properties

- ◆ **set...to, set...=**

Syntax: set the *property* to *expression*
 set the *property* = *expression*
 set *variable* to *expression*
 set *variable* = *expression*

This command evaluates the expression specified by *expression* and puts the result into the property specified by *property* or the variable specified by *variable*.

Examples:

This statement sets the ink effect for sprite 3 to the ink effect specified by the number 8:

```
set the ink of sprite 3 to 8
```

This statement sets the `soundEnabled` property to the opposite of its current state. When the `soundEnabled` is TRUE (the sound is on), this statement turns it off. When the `soundEnabled` is FALSE (the sound is off), this statement turns it on.

```
set the soundEnabled = not (the soundEnabled)
```

This statement sets the variable named vowels to the string "aeiou":

```
set vowels to "aeiou"
```

- ◆ **property keyword**

• setaProp

Syntax: `setaProp list, property, newValue`

This command replaces the value assigned to *property* with the value specified by *newValue* in the list specified by *list*. When the property isn't already in the list, Lingo adds the new property and value.

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

Example:

These statements create a property list and then adds the item `#c:10` to the list:

```
set newList = [#a:1, #b:5]
put newList
-- [#a:1, #b:5]
setaProp newList, #c, 10
put newList
-- [#a:1, #b:5, #c:10]
```

- **setAt**

Syntax: `setAt list, orderNumber, value`

This command replaces the item specified by *orderNumber* with the *value* specified by value in the list specified by *list*.

- When *orderNumber* is greater than the number of items in a linear list, the list is expanded with blank entries to provide the number of places specified by *orderNumber*.
- When *orderNumber* is greater than the number of items in a property list, an error alert occurs.

The `setAt` command works with linear and property lists.

If the list contains fewer items than the position specified by *orderNumber*, Lingo gives a script error.

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]

● **setCallBack**

Syntax: `setCallBack XCMDname, value`

This command specifies how Lingo handles unsupported callbacks from the HyperTalk XCMD or XFCN specified by *XCMDname* when the movie plays on a Macintosh. In Windows the `setCallBack` command has no effect.

- When *value* is TRUE (1), unsupported callbacks from the specified XCMD or XFCN cause a generic alert to be displayed.
- When *value* is FALSE (0), unsupported callbacks from the specified XCMD or XFCN are ignored.
- When *value* is an object created from an XObject, unsupported callbacks from the specified XCMD or XFCN cause various messages to be sent to the object.

Example:

This statement has Lingo ignore unsupported callbacks from the `SuperDuperXCMD` command:

```
setCallBack SuperDuperXCMD, 0
```

- **setProp**

Syntax: `setProp list, property, newValue`

This command replaces the value assigned to property with the value specified by `newValue` in the list specified by `list`. If the list doesn't contain the specified property, `setProp` produces a script error.

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

This command is similar to the `setaProp` command, except that this command gives an error when the property is not already in the list.

Example:

This statement changes the value assigned to the age property of property list `x` to 11:

```
set x = [#age:10, #sex:0]
setProp x #age, 11
```

- **setaProp** command

- **setTrackEnabled**

Syntax: `setTrackEnabled(sprite whichSprite, whichTrack, trueOrFalse)`

This digital video sprite property determines whether the specified track of a digital video is enabled to play.

- When `setTrackEnabled` is TRUE, the specified track is enabled.
- When `setTrackEnabled` is FALSE, the specified track is disabled.

Example:

This statement enables track 3 of the digital video assigned to sprite channel 8:

```
setTrackEnabled(sprite 8, 3, TRUE)
```

● **shapeType**

Syntax: the shapeType of member *whichCastmember*

This shape cast member property indicates the specified shape's type. Possible types are #rect, #roundRect, #oval, or #line. This property is useful for specifying a shape cast member's type after the shape cast member is created from Lingo.

Example:

These statements create a new shape cast member numbered 100 and then define it as an oval:

```
new(#shape, member 100)
set shapeType of member 100 = #oval
```

- **shiftDown**

Syntax: the shiftDown

This function indicates whether the user is pressing the Shift key.

- When the `shiftDown` is TRUE, the user is pressing the Shift key.
- When the `shiftDown` is FALSE, the user is not pressing the Shift key.

Example:

This statement checks whether the Shift key is being pressed and calls the handler `doShiftKey` if it is:

```
if the shiftDown then doShiftKey (the key)
```

- **commandDown**, **controlDown**, **key**, and **optionDown** functions

- **short**

- date and time functions

- **showGlobals**

Syntax: showGlobals

This command has the message window display all global variables. It is useful for debugging scripts.

- **clearGlobals** and **showLocals** commands; **global** keyword

- **showLocals**

Syntax: `showLocals`

This command has the message window display all local variables. This command can only be used within handlers or parent scripts.

Local variables in handlers are abandoned after the handler executes. This command is useful for debugging scripts.

- **clearGlobals** and **showGlobals** commands; **global** keyword

- **showResFile**

Syntax: showResFile [*whichFile*]

This command, when used on the Macintosh, displays a list of resources in the resource file specified by the string *whichFile*. The file must be already open. If the resource file is in a different folder than the current movie, *whichFile* must specify a pathname. If no file is specified, all open resource files are listed. In Windows, the `showResFile` command has no effect.

There may be many open resource files, and the listing may be very long. To cancel the listing, press the mouse button.

Example:

This statement displays the resource file Special Fonts:

```
showResFile "Special Fonts"
```

- **closeResFile, openResFile, openXlib, and showXlib** commands

- **showXlib**

Syntax: `showXlib [Xlibfilename]`

This command shows all Xtras and XObjects in *Xlibfilename* (it must be open), or all open Xlibraries if no file is specified. Xlibrary files are resource files that contain XCOD (XObjects) resources. If the file is in another folder than the current movie, specify the pathname.

Xlibrary files are resource files that contain XCOD (Xtras and XObjects) resources (Macintosh) or .DLLs (Windows). Because the type of Xlibrary files on the Macintosh and in Windows differs, the list of files that the `showXlib` command generates can be different on different platforms.

The `mDescribe` method displays on line documentation for an XObject.

To use `mDescribe`:

1. Type `showXlib` in the message window and press Return.
This displays all open Xlibrary resource files and all Xtras and XObjects contained in those Xlibraries.
2. Using the list of Xtras and XObjects displayed in the message window, type `XObjectName(mDescribe)` and press Return.
This displays the on-line documentation for that XObject.

Example:

This statement displays the Xtras and XObjects in the VideoDisc Library:

```
showXlib "VideoDisc Xlibrary"
```

- **closeXlib** and **openXlib** commands

- ◆ **shutDown**

Syntax: shutDown

This command has different effects on the Macintosh and in Windows.

- ◆ On the Macintosh, the `shutDown` command closes all open applications and turns the computer off.
- ◆ In Windows, the `shutDown` command exits Director or the projector and then exits Windows.

Example:

This statement checks whether the user has pressed Control+R and shuts down the computer if he or she has:

```
if the key = "s" and the commandDown then shutDown
```

- ◆ **quit** and **restart** commands

• sin

Syntax: `sin(angle)`

This function calculates the sine of the specified angle. The angle must be expressed in radians as a floating point number.

Example:

The following statement calculates the sine of pi/2:

```
put sin (pi()/2.0)
-- 1
```

Note: The symbol π cannot be used in a Lingo expression.

- **size of member**

Syntax: the size of member *castName*

This cast member property permits you to learn the size, in bytes, of a specific cast member number or name. Divide bytes by 1024 to convert to kilobytes.

Example:

```
put the size of member "Shrine" into field "How Big"
```

- **sort**

Syntax: `sort list`

This command puts the items in the list specified by *list* into alphanumeric order.

- When the list is a linear list, the list is sorted by values.
- When the list is a property list, the list is sorted alphabetically by properties.

Once a list is sorted, it maintains its sort order even when you add new variables using the `add` command.

Example:

This statement puts the list `Values`, which consists of `[#a: 1, #d: 2, #c: 3]`, into alphanumeric order. The result appears below the statement:

```
put values
-- [#a: 1, #d: 2, #c: 3]
sort Values
put Values
--[#a: 1, #c: 3, #d: 2]
```


- **sound close**

Syntax: `sound close soundChannel`

This command stops the sound playing in and then closes the sound channel specified by *soundChannel*.

Example:

This statement stops any sound playing in and closes sound channel 1:

```
sound close 1
```

- **sound fadeIn**

Syntax: `sound fadeIn whichChannel`
 `sound fadeIn whichChannel, ticks`

This command fades in a sound in the specified sound channel over a period of frames or ticks.

- When ticks is specified, then the fade in occurs evenly over that period of time.
- When ticks is not specified, the default number of ticks is calculated as $15 * (60 / (\text{Tempo setting}))$ 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 the sound in the specified channel changes.

Example:

This statement fades in the sound in channel 1 over 5 seconds:

```
sound fadeIn 1, 5 * 60
```

- **sound fadeOut** command

- ◆ **sound fadeOut**

Syntax: `sound fadeOut whichChannel`
 `sound fadeOut whichChannel, ticks`

This command fades out a sound in the specified sound channel over a period of frames or ticks.

- ◆ When ticks is specified, then 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 fadeout continues at a predetermined rate until the number of ticks has elapsed, or the sound in the specified channel changes.

Example:

This statement fades in the sound in channel 1 over 5 seconds:

```
sound fadeIn 1, 5 * 60
```

- ◆ **sound fadeIn** command

- **sound of member**

Syntax: the sound of member *whichCastmember*

This movie and digital video cast member property determines whether the sound for the specified movie or digital video plays.

- When the `sound of member` is TRUE (1), the sound plays.
- When the `sound member` is FALSE (0), the sound doesn't play.

This property can be tested and set for Director movies and digital video cast members.

Example:

This statement turns on the sound for the Director movie cast member Movie Clip:

```
set the sound of member "Movie Clip" to 1
```

- **sound playFile**

Syntax: `sound playFile whichChannel, whichFile`

This command plays the AIFF or WAVE sound located at *whichFile* in the sound channel specified by *whichChannel*.

When the sound file is in a different folder than the movie, *whichFile* must specify the full pathname to the file.

The `sound playFile` command streams files from disk rather than playing them from RAM the way Director plays sound cast members. 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.

On the Macintosh, this command requires System 6.0.7 or later to work; otherwise the sound playback will not occur.

Examples:

This statement plays the file named Thunder in channel 1:

```
sound playFile 1, "Thunder"
```

This statement plays the file named Thunder in channel 3:

```
sound playFile 3, the pathName &"Thunder"
```

- **sound stop** command

- **sound stop**

Syntax: `sound stop whichChannel`

This command stops the playing of the sound playing in the specified channel.

Example:

This statement checks whether a sound is playing in sound channel 1 and stops the sound if it is:

```
if soundBusy(1) then sound stop 1
```

- **soundBusy** function

- **soundBusy**

Syntax: `soundBusy (whichChannel)`

This function determines whether a sound is playing in the sound channel specified by *whichChannel*.

- When a sound is playing in the specified sound channel, the `soundBusy` function is TRUE (1).
- When no sound is playing in the specified sound channel, the `soundBusy` function is FALSE (0).

Make sure that you allow enough time for the sound to start playing before using `soundBusy` to check the sound channel.

Example:

This statement checks whether a sound is playing in sound channel 1 and loops in the frame if it is. This would allow the sound to finish before the playback head goes to another frame:

```
if soundBusy(1) then go to the frame
```

- **sound playFile** and **sound stop** commands

- **soundEnabled**

Syntax: `the soundEnabled`

This property determines whether the sound is on or off. TRUE means that the sound is on.

The `soundEnabled` property can be tested and set; and the default value is TRUE. When you set this property to FALSE, the volume setting of the sound is not changed but you do not hear the sound.

Example:

This statement set turns to the opposite of its current setting. It turns the sound on if it is off and turns it off if it is on:

```
set the soundEnabled to not (the soundEnabled)
```

- **soundLevel, volume of sound, volume of sprite** properties

- **soundLevel**

Syntax: the soundLevel

This property determines the volume level of the sound that is played through the computer's speaker. Settings range from 0 (no sound) to 7 (maximum sound volume).

The `soundLevel` property can be tested and set. The default value is 7.

Examples:

This statement sets the variable `oldSound` equal to the current sound level:

```
put the soundLevel into oldSound
```

This statement sets the sound level to 5:

```
set the soundLevel to 5
```

- **soundEnabled** and **volume of sound** property

- **sourceRect**

Syntax: the sourceRect of window *whichWindow*

This window property specifies the coordinates of the rectangle that the movie that plays in the window specified by *whichWindow* was originally created for.

Example:

This statement displays the original coordinates of the movie Control Panel in the message window:

```
put the sourceRect of "Control Panel"
```

- **sprite**

Syntax: the *property* of sprite *whichSprite*

This keyword tells Lingo that the value specified by *whichSprite* is a sprite number. It is used with every sprite property.

A sprite is an occurrence of a cast member in an animation channel of the score.

Examples:

This statement sets the variable named `horizontal` to the `locH` of sprite 1:

```
put the locH of sprite 1 into horizontal
```

This statement turns on the puppet condition for the sprite that has sprite number `i + 1`:

```
set the puppet of sprite (i + 1) to TRUE
```

- **puppetSprite** command

- **sprite...intersects**

Syntax: *sprite* *sprite1* intersects *sprite2*

This operator compares the position of two sprites. It is TRUE if the bounding rectangle of *sprite1* touches the bounding rectangle of *sprite2*.

If both sprites have matte ink, their actual outlines are used, not the bounding rectangles. A sprite's outline is defined by the non-white pixels that make up its border.

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

Example:

This statement checks whether two sprites intersect and changes the contents of the field cast member Notice to "You placed it correctly." if they do:

```
if sprite i intersects j then ↵  
  put "You placed it correctly." into field "Notice"
```

- **sprite within** comparison operator

- **sprite...within**

Syntax: `sprite sprite1 within sprite2`

This comparison operator compares the position of two sprites. It is true if the bounding rectangle of *sprite1* is entirely inside the bounding rectangle of *sprite2*.

If both sprites have matte ink, their actual outlines are used, not the bounding rectangles. A sprite's outline is defined by the non-white pixels that make up its border.

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

Example:

This statement checks whether two sprites intersect and calls the handler `doInside` if they do:

```
if sprite 3 within 2 boundary -  
  then doInside
```

- **sprite intersects** comparison operator

- **spriteBox**

Syntax: `spriteBox whichSprite, left, top, right, bottom`

This command sets the bounding rectangle coordinates of the puppet sprite specified by the integer expression *whichSprite*. The `spriteBox` command gives you a way to set the left, top, right, and bottom sprite properties of a sprite directly without having to convert it into `locH`, `locV`, `width`, and `height`. This is useful because the left, top, right, and bottom sprite properties cannot be set directly.

This command works only on puppet sprites. For bitmap sprites, the `stretch` of `sprite` property must be TRUE to use this command.

A sprite's coordinates change based on their registration points. For bitmap sprites, it may be necessary to move the registration points in order to obtain proper results.

Example:

This statement sets the coordinates of sprite 3's bounding rectangle to 50, 50, 200, and 250:

```
spriteBox 3, 50, 50, 200, 250
```

This statement sets the bounding rectangle of the sprite whose number is `mySprite` to the starting values and the current cursor location. This creates a rectangle that stretches from the specified point to the mouse cursor:

```
spriteBox mySprite, -  
startH, startV, the mouseH, the mouseV
```

- **bottom of sprite, height of sprite, left of sprite, right of sprite, stretch of sprite, top of sprite, and width of sprite** sprite properties; **puppetSprite** and **updateStage** command

- **sqrt**

Syntax: `sqrt(number)`
 the sqrt of *number*

This function yields the square root of the number specified by *number*.

- When *number* is a floating point number, the result is a floating point number.
- When *number* is an integer, the result is rounded to the nearest integer.

The value of *number* must be a decimal number that is greater than zero.

Example:

This statement displays the square root of 3.0 in the message window:

```
put sqrt(3.0)
-- 1.7321
```

- **floatPrecision** property

● stage

Syntax: the stage

This system property is used to refer to the main movie in commands and functions that relate to windows. This is useful when using the `tell` command to send a message to the main movie from a child movie.

Example:

This statement causes the main movie to stop animating:

```
tell the stage to pause
```

This statement displays the current setting of the stage:

```
put the rect of the stage  
--rect (0, 0, 640, 480)
```


- **stageBottom**

Syntax: the stageBottom

This function--along with the `stageLeft`, the `stageRight`, and the `stageTop`-- indicates where the stage is positioned on the desktop. It returns the bottom vertical coordinate of the stage, relative to the upper left corner of the main screen. The height of the stage in pixels is given by `the stageBottom - the stageTop`.

The `stageBottom` function can be tested but not set.

Example:

These two statements position sprite 3 a distance of 50 pixels from the bottom edge of the stage:

```
put the stageBottom - the stageTop into -
    stageHeight
set the locV of sprite 3 to stageHeight - 50
```

Note: Sprite coordinates are expressed relative to the upper-left corner of the Stage. See "Changing Sprite Properties" in Chapter 2, "Using the Stage and Score," of *Using Director*.

- **stageLeft**, **stageRight**, and **stageTop** functions; **locH of sprite** and **locV of sprite** sprite properties

- ◆ **stageColor**

Syntax: the stageColor

This property determines the color of the movie background.

The value of the `stageColor` ranges from 0 to 255 for 8-bit color, or from 0 to 15 for 4-bit color. You can click a color in the color palette to see that color's index number in the lower left corner of the window. Setting the `stageColor` in a Lingo script is equivalent to choosing the stage color from the pop-up palette in the panel window.

Example:

This statement sets the variable `oldColor` to the index number of the current stage color:

```
put the stageColor into oldColor
```

This statement sets the stage color to the color assigned to chip 249 on the current palette:

```
set the stageColor to 249
```

- ◆ **backColor of sprite** and **foreColor of sprite** properties

- **stageLeft**

Syntax: the stageLeft

This function--along with the stageRight, the stageTop, and the stageBottom--indicates where the stage is positioned on the desktop. It equals the left horizontal coordinate of the stage, relative to the upper left corner of the main screen. When the stage is flush with the left side of the main screen, this coordinate is zero.

The stageLeft function can be tested but not set.

Sprite coordinates are expressed relative to the upper-left corner of the Stage.

Example:

This statement checks whether the left edge of the stage is beyond the left edge of the screen and calls the handler leftMonitorProcedure if it is:

```
if the stageLeft < 0 then leftMonitorProcedure
```

- **stageBottom**, **stageRight**, and **stageTop**, functions; **locH of sprite** and **locV of sprite** sprite properties

- **stageRight**

Syntax: the stageRight

This function--along with the stageLeft, the stageTop, and the stageBottom--indicates where the stage is positioned on the desktop. It returns the right horizontal coordinate of the stage, relative to the upper left corner of the main screen's desktop. The width of the stage in pixels is given by the stageRight - the stageLeft.

The stageRight function can be tested but not set.

Sprite coordinates are expressed relative to the upper-left corner of the Stage.

Example:

These two statements position sprite 3 a distance of 50 pixels from the right edge of the stage:

```
put the stageRight - the stageLeft into stageWidth
set the locH of sprite 3 to stageWidth - 50
```

- **stageBottom**, **stageLeft**, and **stageTop** functions; **locH of sprite** and **locV of sprite** sprite properties

- **stageTop**

Syntax: the stageTop

This function--along with the stageBottom, the stageLeft, and the stageRight--indicates where the stage is positioned on the desktop. It returns the top vertical coordinate of the stage, relative to the upper left corner of the main screen's desktop. If the stage is in the upper left corner of the main screen, this coordinate is zero.

The stageTop function can be tested but not set.

Example:

This statement checks whether the top of the stage is beyond the top of the screen and calls the handler upperMonitorProcedure if it is:

```
if the stageTop < 0 then upperMonitorProcedure
```

Sprite coordinates are expressed relative to the upper-left corner of the Stage.

- **stageBottom**, **stageLeft**, and **stageRight** functions; **locH of sprite** and **locV of sprite** sprite properties

- **startMovie**

See [on startMovie](#) movie handler.

- **starts**

Syntax: *string1* starts *string2*

This comparison operator compares two strings.

- When *string1* starts with *string2*, the condition is TRUE (1).
- When *string1* does not start with *string2*, the condition is FALSE (0).

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

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

Example:

This statement has the message window display whether the word Macromedia starts with the string Macro:

```
put "Macromedia" starts "Macro"
```

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

- **contains** comparison operator

- **startTimer**

Syntax: startTimer

This command sets the timer property to zero. It also resets all the accumulating timers for the `lastClick`, `lastEvent`, `lastKey`, and `lastRoll` functions to zero.

Example:

This handler set the timer to zero when a key is pressed:

```
on keyDown
  startTimer
end keyDown
```

- **lastClick**, **lastEvent**, **lastKey**, and **lastRoll** functions; **timeoutLength**, **timeoutMouse**, **timeoutPlay**, **timeoutScript**, and **timer** properties

- **stillDown**

Syntax: the stillDown

This function indicates whether the user is pressing the mouse button.

- When the user is pressing the mouse button, the `stillDown` is TRUE.
- When the user is not pressing the mouse button, the `stillDown` is FALSE.

This function is useful within a `mouseDown` script to trigger certain events only after the `mouseUp`.

Lingo cannot test the `stillDown` when it is used inside a repeat loop. Use the `mouseDown` function inside of 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
```

- **mouseDown** function

- **stretch of sprite**

Syntax: the stretch of sprite *whichSprite*

This sprite property determines whether the bitmap sprite specified by *whichSprite* can be stretched by using the `spriteBox` command or the `width of sprite` and `height of sprite` properties. If it is TRUE, the bitmap sprite can be stretched.

The `stretch of sprite` property can be tested and set, and the default value is FALSE. When FALSE, the bitmap sprite always stays at its default or normal size.

The `stretch of sprite` property applies to bitmap, digital video, and picture cast members, but not to shape, field, or button cast members. Shapes can be stretched at any time by setting their `height of sprite` and `width of sprite` properties, regardless of the setting of their `stretch` property. Field and button cast members cannot be stretched in any case.

Director requires much more processor time to draw stretched sprites than regular sprites, which can affect movie performance.

In order to have its properties set using Lingo, the sprite must be a puppet.

Example:

This statement checks whether sprite 3 is stretchable and sets the sprite's width to 10 pixels if it is:

```
if the stretch of sprite 3 = TRUE then ↵
    set the width of sprite 3 to 10
```

- **spriteBox** and **updateStage** commands; **height of sprite** and **width of sprite** properties

- **string**

Syntax: `string(expression)`

This function converts an integer, floating point number, or symbol expression to a string.

Example:

This statement adds 2 + 2 and has the message window display the results:

```
put string(2 + 2)
```

This statement converts the symbol #123 to a string:

```
put string(123)  
-- "123"
```

- **value** function

- **stringP**

Syntax: `stringP(expression)`

This function determines whether the expression specified by *expression* is a string.

- When the expression is a string, the result is TRUE.
- When the expression is not a string, the result is FALSE.

The "P" in `stringP` stands for predicate.

Examples:

This statement checks whether "3" is a string:

```
put stringP("3")
```

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

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.

- **floatP, ilk, integerP, objectP, and symbolP** functions

• **switchColorDepth**

Syntax: the switchColorDepth

This property, when the movie plays on the Macintosh, determines whether Director automatically switches the color depth when loading a movie. The `switchColorDepth` property has no effect in Windows.

- When the `switchColorDepth` is TRUE, Director switches the monitor(s) that the stage occupies to the color depth of the movie that is being loaded.
- When the `switchColorDepth` is FALSE, Director leaves the color depth of the monitor(s) unchanged when a movie is loaded.

When the `switchColorDepth` is TRUE, nothing happens until a new movie is loaded.

Setting the monitor's color depth to that of the movie is good practice.

- When the monitor's color depth is set below that of the movie, resetting it to the color depth of the movie (assuming that the monitor can provide that color depth) helps maintain the movie's original appearance.
- When the monitor's color depth is higher than that of the movie, reducing the color depth lets you use the minimum amount of memory to play movies. At minimum memory, loading cast members is more efficient and animation can occur faster.

The `switchColorDepth` property can be tested and set. The default value is the setting for the Reset Monitor to Movie's Color Depth checkbox in the **General Preferences** dialog box.

Examples:

This statement sets the variable named `switcher` to the current setting of `switchColorDepth`:

```
put the switchColorDepth into switcher
```

This statement checks whether the current color depth is 8-bit and turns the `switchColorDepth` property on if it is:

```
if the colorDepth = 8 then -  
    set the switchColorDepth to TRUE
```

- **colorDepth** property; **colorQD** function

- **symbolP**

Syntax: `symbolP(expression)`

This function determines whether the expression specified by *expression* is a symbol.

- When the expression is a symbol, the result is TRUE.
- When the expression is not a symbol, the result is FALSE.

The "P" in symbolP stands for predicate.

Example:

This statement checks whether #3 is a symbol:

```
put symbolP(#3)
```

- **TAB**

Syntax: TAB

This character constant represents the Tab key.

Example:

This statement checks whether the character typed is the Tab character and calls the handler `doNextField` if it is:

```
if the key = TAB then doNextField
```

This statement advances or retreats the playback head depending on whether the user presses Tab or Shift-Tab:

```
if the key = TAB then
  if the shiftDown then
    go the frame -1
  else
    go the frame +1
  end if
end if
```

- **BACKSPACE, EMPTY, RETURN** character constants

• tan

Syntax: `tan(angle)`

This function yields the tan of the specified angle. The angle must be expressed in radians as a floating point number. A radian is an arc in a circle, equal in length to the radius. It is 57.295 degrees. There are 2π or 6.2833 radians in a circle.

Example:

The following function yields the tangent of $\pi/4$:

```
tan (pi()/4.0) = 1
```

Note: That the π symbol cannot be used in a Lingo expression.

• tell

Syntax: `tell object to statement`

or

```
tell object
statement(s)
end tell
```

This command communicates the statement or statements specified by *statement(s)* to the object specified by *object*.

The `tell` command is useful for allowing movies to interact. It can be used within a main movie to send a message to a movie playing in a window, or to send a message from a movie playing in a window to the main movie. For example, the `tell` command can let a button in a control panel call a handler in a movie playing in a window. The movie playing in a window could react to the first movie handler by executing the handler. The movie playing in the window could interact with the main movie by sending some value back to the movie.

When you use the `tell` command to send a message to a movie playing in a window, it is important to use an object name that identifies the window by using the full pathname or its number in the `windowList`. If you use the `windowList`, use the expression `getAt(the windowList, windowNum)`, where *windowNum* is a variable that contains the number of the window's position in the list. Because opening and closing windows may change the order of the `windowList`, it is a good idea to store the full pathname as a global variable.

A multiple line `tell` command resembles a handler. It needs an end statement:

```
global childMovie
tell window childMovie
  go to frame 5
  set the stageColor to 100
  set the memberNum of sprite 4 to 45
  updateStage
end tell
```

When a message calls a handler, a value returned from the handler can be found in the global property `the result` after the called handler is done:

```
global childMovie
tell window childMovie to calcBalance
-- a handler name
put the result into myBalance
-- return value from "calcBalance" handler
```

When you use the `tell` command to send a message from a movie playing in a window to the main movie, use the system property `the stage` as the object name:

```
tell the stage to go to frame "Main menu"
```

When you use the `tell` command to call a handler in another movie, make sure that you

do not have a handler by the same name in the same script in the local movie. If you do, the local script will be called. This applies only to handlers in the same script in which you are using the tell command.

Example:

This statement has the window Control Panel instruct the movie Simulation to branch to another frame:

```
tell window "Simulation" to go to frame "Save"
```

- **text of member**

Syntax: the text of member *whichCastmember*

This cast member property determines the character string that is contained in the field cast member specified by *whichCastmember*.

The `text of member` property is useful for displaying messages and recording what the user types.

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

Lingo changes to the text of a cast member remove any special formatting you have applied to individual words or lines. Altering the `text of member` reapplies global formatting.

Example:

This statement places the phrase "Thank you." in the empty cast member Response:

```
if the text of member "Response" = EMPTY then -  
    set the text of member "Response" to "Thank You."
```

This statement sets the content of cast member Notice to "You have made the right decision!":

```
set the text of member "Notice" = "You have -  
made the right decision!"
```

- **selEnd** and **selStart** field properties;

• the

Syntax: `the` *property*

All Lingo properties and many sprite properties and functions require the keyword `the` to precede the property. This distinguishes the property from a variable or object name.

Properties have "super-global" scope, which means they are available within handlers and methods even without a global declaration. Like global variables, Lingo system properties are available between different movies in the same presentation (unless they are changed by system events). Sprite properties would change when a new movie is loaded.

- **ticks**

Syntax: the ticks

This function returns the current time in ticks (1/60th of a second). Counting ticks begins from the time the computer is started.

Example:

This statement converts ticks to minutes by dividing the number of ticks by 60 twice and then sets the variable `minutesOn` to the result:

```
put the ticks/60/60 into minutesOn
```

- **time** and **timer** functions

- **time**

Syntax: the time
 the short time
 the long time
 the abbreviated time
 the abbrev time
 the abbr time

This function returns the current time in the system clock as a string in one of three formats: short, long, or abbreviated. If no format is specified, the default is short. The abbreviated format can also be referred to as `abbrev` and `abbr`. In the United States, the short and abbreviated formats are the same.

Examples:

These statements have the message window display the time in different formats. Possible results appear below each statement:

```
put the short time  
--"1:30 PM"
```

```
put the long time  
--"1:30:24 PM"
```

```
put the abbreviated time  
--"1:30 PM"
```

Note: The three time formats vary, depending on the individual computer's time format. The examples above are for the United States.

- **date** function

- **timeoutKeyDown**

Syntax: the timeoutKeyDown

When this property is TRUE, keyDown events set the timeoutLapsed property to zero.

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

Example:

This statement sets the variable timing to the value of the timeoutKeyDown:

```
put the timeoutKeyDown into timing
```

This statement turns off the timeoutKeyDown:

```
set the timeoutKeyDown to FALSE
```

- **keyDownScript** property

• timeoutLapsed

Syntax: the timeoutLapsed

This property indicates the number of ticks elapsed since the last timeout. A timeout event occurs when the `timeoutLapsed` property reaches the time specified by the `timeoutLength` property.

The `timeoutLapsed` property can be tested, but not set directly in Lingo.

Example:

This statement sets the field of member Countdown to the value of the `timeoutLapsed` property. Dividing the `timeOutLapsed` by 60 converts it to seconds:

```
put the timeoutLapsed / 60 into field "Countdown"
```


• timeoutLength

Syntax: the timeoutLength

This property determines the number of ticks before a timeout event occurs. A timeout occurs when the `timeoutLapsed` property reaches the time specified by the `timeoutLength` property.

The `timeoutLength` property can be tested and set. The default value is 10,800 ticks, which is 3 minutes.

Example:

This statement sets the `timeOutLength` to 10 seconds:

```
set the timeoutLength to 10 * 60
```

- **timeoutMouse**

Syntax: the timeoutMouse

This property determines whether `mouseDown` events reset the `timeoutLapsed` property to zero. When this property is `TRUE`, `mouseDown` events reset the `timeoutLapsed` property.

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

Examples:

This statement records the current setting of the `timeoutMouse` by setting the variable named `timing` to the `timeoutMouse`:

```
put the timeoutMouse into timing
```

This statement sets the `timeoutMouse` property to `FALSE`. The result is that the `timeoutLapsed` property keeps its current value when the mouse button is pressed:

```
set the timeoutMouse to FALSE
```

- **mouseDownScript** and **mouseUpScript** properties

● timeoutPlay

Syntax: the timeoutPlay

This property determines whether the `timeoutLapsed` property is reset to zero when a movie is played. When `timeoutPlay` is `TRUE`, playing a movie resets the `timeoutLapsed` property to zero. This allows timeouts to occur only when the animation is paused.

The `timeoutPlay` property can be tested and set. The default value is `FALSE`.

Example:

This statement sets the `timeoutPlay` to `TRUE`, which has Lingo reset the `timeoutLapsed` property to zero when a movie is played:

```
set the timeoutPlay to true
```

● timeoutScript

Syntax: `the timeoutScript`

This property determines the Lingo that Director executes as a primary event handler when a timeout occurs.

Define a primary event handler for timeouts by setting `the timeoutScript` to a string of the appropriate Lingo. The Lingo can be a simple Lingo statement or a calling statement for a handler. When the event script you've assigned is no longer appropriate, turn it off with the statement `set the timeoutScript to EMPTY`.

The `timeoutScript` property can be tested and set. The default value is `EMPTY`.

Example:

This statement sets `the timeoutScript` to a calling script for the handler `timeoutProcedure`:

```
set the timeoutScript to "timeoutProcedure"
```

- **timer**

Syntax: the timer

This property is a free-running timer that counts time in ticks (60ths of a second). It has nothing to do with the `timeOutScript`. It is only for convenience in timing certain events. The `startTimer` command zeroes the value of the timer property.

The `timer` property is useful for setting up delays within handlers. (The `delay` command works only in frame scripts.) For example, you can use the `timer` to synchronize pictures to a soundtrack by inserting a delay that makes the movie wait until a sound is finished.

Example:

This handler creates a 1 second delay:

```
on countTime
  startTimer
  repeat while the timer < 60
    nothing
  end repeat
end countTime
```

This statement sets the variable `startTicks` to the current value of the timer:

```
set the timer = startTicks
```

- **lastClick, lastEvent, lastKey, and lastRoll** functions; **startTimer** command

- **timeScale of member**

Syntax: the timeScale of member *whichCastmember*

This digital video cast member property gives the time unit per second that the digital video's frames are based on. For example, a QuickTime digital video is measured in 1/600s of a second. This property can be tested but not set.

- **digitalVideoTimeScale**

- **title of window**

Syntax: the title of window *whichWindow*

This window property is the title of the window specified by *whichWindow*.

The `title of window` property can be tested and set.

Example:

This statement makes Action View the title of window X:

```
set the title of window "X" to "Action View"
```

● **titleVisible of window**

Syntax: the titleVisible of window *whichWindow*

This window property specifies whether the window specified by *whichWindow* displays the window title in the window's title bar.

The titleVisible of window property can be tested and set.

Example:

This statement displays the title of the window Control Panel by setting the window's titleVisible property to TRUE:

```
set the titleVisible of window "Control Panel" to TRUE
```


- **to**

The word `to` occurs in a number of Lingo constructs.

- **char...of**, **item...of**, **line...of**, and **word...of** chunk expression keywords; **repeat with**, and **set...to/set...=** commands

- **top of sprite**

Syntax: the top of sprite *whichSprite*

This sprite property returns the top vertical coordinate of the bounding rectangle of the sprite specified by *whichSprite*. The coordinate is the number of pixels from the upper left corner of the stage.

The `top of sprite` property can be tested, but not set directly. The top vertical coordinate of a sprite can be set with the `spriteBox` command.

Example:

This statement checks whether the top of sprite 3 is above the top of the stage and calls the handler `offTopEdge` if it is:

```
if the top of sprite 3 < 0 then offTopEdge
```

- **bottom of sprite, height of sprite, locH left of sprite, of sprite, locV of sprite, right of sprite, and width of sprite** sprite properties; **spriteBox** command

- **trace**

Syntax: `the trace = trueOrFalse`

This property specifies whether the movie's trace function is on or off.

- When `the trace` is TRUE (1), the trace function is on.
- When `the trace` is FALSE (0), the trace is off.

Example:

This statement turns the trace function on:

```
set the trace = TRUE
```

● **traceLoad**

Syntax: `the traceLoad`

This property specifies the amount of information that is displayed about cast members as they are loaded. The possible values for the `the traceLoad` property have the following effect:

0--Displays no information

1--Displays cast members' names

2--Display cast members' names, number of the current frame, movie name, and file seek offset.

The `traceLoad` property can be tested and set.

Example:

This statement has the movie display the names of cast members as they are loaded:

```
set the traceLoad to 1
```

● **traceLogFile**

Syntax: the traceLogFile

This property specifies the name of the file that the message window display is written to. You can close the file by setting the `traceLogFile` property to `EMPTY ("")`.

Example:

This statement has Lingo write the display of the message window to the file messages:

```
set the traceLogFile = "Messages"
```

This statement closes the file that the message window display is being written to:

```
set the traceLogFile = ""
```

• **trackCount(member)**

Syntax: `trackCount(member whichCastmember)`

This digital video cast member property returns the number of tracks on the specified digital video cast member. This property can be tested but not set.

Example:

This statement determines the number of tracks in the digital video cast member Jazz Chronicles and displays the answer in the message window:

```
put trackCount(member "Jazz Chronicle")
```

- **trackCount(sprite)**

Syntax: `trackCount(sprite whichSprite)`

This digital video sprite property returns the number of tracks on the specified digital video sprite. This property can be read but not set.

Example:

This statement determines the number of tracks in the digital video sprite assigned to channel 10 and displays the answer in the message window:

```
put trackCount(sprite 10)
```

- ◆ **trackEnabled**

Syntax: `trackEnabled(sprite whichSprite, whichTrack)`

This digital video sprite property indicates whether the specified track of a digital video is enabled to play.

- ◆ When `trackEnabled` is TRUE, the specified track is enabled.
- ◆ When `trackEnabled` is FALSE, the specified track is disabled.

This property cannot be set. You must use the `setTrackEnabled` property

- ◆ **setTrackEnabled** sprite property

• **trackNextKeyTime**

Syntax: `trackNextKeyTime(sprite whichSprite, whichTrack)`

This digital video sprite property indicates the time of the key frame that follows the current time in the specified digital video track. This property can be tested but not set.

Example:

This statement determines the time of the key frame that follows the current time in track 5 of the digital video assigned to sprite channel 15 and displays the result in the message window:

```
put trackNextKeyTime(sprite 15, 5)
```

• trackNextSampleTime

Syntax: `trackNextSampleTime(sprite whichSprite, whichTrack)`

This digital video sprite property indicates the time of the next sample that follows the digital video's current time. It is useful for locating text tracks in a digital video. This property can be tested but not set.

Example:

This statement determines the time of the next sample that follows the current time in track 5 of the digital video assigned to sprite 15:

```
put trackNextSampleTime(sprite 15, 5)
```

• trackPreviousKeyTime

Syntax: trackPreviousKeyTime(sprite *whichSprite*, *whichTrack*)

This digital video sprite property reports the time of the prior key frame that precedes the current time. This property can be tested but not set.

Example:

This statement determines the time of the key frame 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 previousKeyTime(sprite 15, 5)
```

• **trackPreviousSampleTime**

Syntax: `trackPreviousSampleTime(sprite whichSprite, whichTrack)`

This digital video sprite property indicates the time of the sample preceding the digital video's current time. It is useful for locating text tracks in a digital video. This property can be tested but not set.

Example:

This statement determines the time of the sample in track 5 that precedes the current time in the digital video sprite assigned to channel 15 and displays the result in the message window:

```
put trackPreviousSampleTime(sprite 15, 5)
```

• **trackStartTime(member)**

Syntax: `trackStartTime(member whichCastMember, whichTrack)`

This digital video cast member property gives the start time of the specified track of the specified digital video cast member. It can be tested but not set.

Example:

This statement determines the start time of track 5 in the digital video cast member Jazz Chronicle and displays the result in the message window:

```
put trackStartTime(member "Jazz Chronicle", 5)
```

- **trackStartTime(sprite)**

Syntax: `trackStartTime(sprite whichSprite, whichTrack)`

This sprite property sets the starting time of a digital video movie in the specified sprite channel. The value of `trackStartTime` is measured in ticks.

When a digital video movie is played, `trackStartTime` determines where playback begins.

Example:

In the message window, this statement tells you when track 5 in sprite channel 10 starts playing -- at 120 ticks (2 seconds) into the track:

```
put trackStartTime(sprite 10, 5)
-- 120
```

- duration of member property; movieRate of sprite and movieTime of sprite sprite properties

• **trackStopTime(member)**

Syntax: `trackStopTime(member whichCastmember, whichTrack)`

This digital video cast member property gives 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 trackStopTime(member "Jazz Chronicle", 5)
```

- ◆ **trackStopTime(sprite)**

Syntax: `trackStopTime(sprite, whichSprite, whichTrack)`

This digital video sprite property gives the stop time of the specified track of the specified digital video sprite. It can be tested but not set.

When a digital video movie is played, the `trackStopTime` is where playback halts or loops if the `loop` property is turned on.

Example:

This statement determines the stop time of track 5 in the digital video assigned to sprite 6 and displays the result in the message window:

```
put trackStopTime(sprite 6, 5)
```

- ◆ **movieRate of sprite, movieTime of sprite, and trackStartTime (member)** properties

• **trackText**

Syntax: `trackText(sprite whichSprite, whichTrack)`

This digital video sprite property gives the text that is at the current time in the specified track of the digital video. The result is a string value, which can be up to 32k long. This property applies to text tracks only. It can be tested but not set.

Example:

This statement assigns the text at the current time in track 5 of the digital video assigned to sprite 20 to the field cast member Archives:

```
put trackText(sprite 20, 5) into member "Archives"
```

• **trackType(member)**

Syntax: trackType (member *whichCastmember*, *whichTrack*)

This digital video cast member property tells which type of media is in the specified track of the specified cast member. Possible values are #video, #sound, #text, #music. This property can be tested but not set.

Example:

The following handler checks whether track 5 of digital video cast member Today's News is a text track and runs the handler `textFormat` if it is:

```
on checkForText
  if trackType(member "Today's News", 5) = #text →
    then textFormat
  end
```

• **trackType(sprite)**

Syntax: trackType(sprite *whichSprite*, *whichTrack*)

This digital video sprite property gives 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 the trackType(sprite 10, 5) = #text -
    then textFormat
end
```

- **trails of sprite**

Syntax: the trails of sprite *whichSprite*

This property turns the trails ink effect on and off for the sprite specified by *whichSprite*. In order to set this property, the sprite must have the `puppetSprite` property set to TRUE before setting the `trails` property. Set the `trails` to 0 to turn trails off; set the `trails` to 1 to turn trails on.

To erase trails:

- Animate another sprite across these pixels.
- Use a transition.

Example:

This statement sets the trails on for sprite 7:

```
set the trails of sprite 7 to 1
```

- **directToStage of member** cast member property

- **transitionType of member**

Syntax: the transitionType of member *whichCastmember*

This transition cast member property determines a transition's type, which is given as a specific number. The possible values are the same as the code numbers assigned to transitions for the `puppetTransition` command.

Example:

This statement sets the type of transition cast member 3 to 51, which is a pixel dissolve cast member:

```
set the transitionType of member 3 to 51
```

- **TRUE**

Syntax: TRUE

This logical constant represents the value of a logically true expression, such as $2 < 3$. It has a numerical value of 1.

Example:

This statement turns on the `soundEnabled` property by setting it to TRUE:

```
set the soundEnabled to TRUE
```

- **FALSE** logical constant

● type of member

Syntax: the type of member *whichCastmember*
 the type of member *whichCastmember* ↵
 of castLib *whichCast*

This cast member property indicates the specified cast member's type. This property replaces `castType`, which appeared in earlier versions of Director.

The `type of member` can be one of the following values:

#bitmap	#picture
#button	#richText
#digitalVideo	#script
#field	#shape
#filmLoop	#sound
#movie	#transition
#palette	

You can also define custom cast member types for custom cast members.

The type cast member 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 runs the handler `fieldFormat` if it is:

```
on checkFormat
  if the type of member "Today's News" = #field ↵
    then fieldFormat
  end
```

● **type of sprite**

Syntax: `the type of sprite`

This sprite property lets you clear sprite channels during score recording by setting the type of sprite value for that sprite to 0. In earlier versions of Director, this sprite property determined the type of the sprite specified by *whichSprite*.

The `the type of sprite` property can be tested and set.

Examples:

This statement clears sprite channel 1 when issued during a score recording session:

```
set the type of sprite 1 to 0
```


- **union rect**

Syntax: `union rect rect1, rect2`

This function returns the smallest rectangle that encloses the two rectangles *rect1* and *rect2*.

Example:

```
put union (rect (0, 0, 10, 10), -  
rect (15, 15, 20, 20))  
-- rect (0, 0, 20, 20)
```

- **map** and **rect** functions

- **unLoad**

Syntax: unLoad

unLoad *theFrameNum*

unLoad *fromFrameNum, toFrameNum*

This command clears the cast members used in a specified frame from memory. When used without an argument, the unLoad command clears the cast members in all the frames of a movie from memory--except any being used in the current frame.

When used with one argument, *theFrameNum*, the unLoad command clears from memory the cast members in that frame. Director automatically unloads the least recently used cast members to accommodate preLoad commands or normal cast loading.

When used with two arguments, *fromFrameNum* and *toFrameNum*, the unLoad command unloads all cast members in the range specified. You can specify a range of frames by frame numbers or frame labels.

Example:

This statement clears the cast members used in frame 10 from memory:

```
unLoad 10
```

This statement clears the cast members used from the frame labeled first to the frame labeled last:

```
unLoad "first","last"
```

- **preLoad**, **preLoadMember**, and **unloadMember** commands; **purgePriority of member** property

- **unLoadMember**

Syntax: unLoadMember member *whichCastmember*
 unLoadMember member *whichCastmember* of castLib *whichCast*

 unLoadMember member *fromCastName, toCastName*

This command clears the specified cast members from memory. When used without an argument, `unLoadMember` causes all cast members in a movie to be cleared from memory--except for any being used in the current frame.

When used with one argument, *whichCastmember*, the `unLoadMember` command clears from memory the cast member name or number that you specify.

When used with two arguments, *fromCastName* and *toCastName*, the `unLoadMember` command unloads all cast members in the range specified.

Example:

This statement clears the cast member Screen 1:

```
unLoadMember member "Screen1"
```

This statement clears from memory all cast members from cast member 11 to cast member 18:

```
unLoadMember 11, 18
```

- **preLoad** and **preLoadMember** commands; **purgePriority of member** property

• **unloadMovie**

Syntax: unloadMovie *whichMovie*

This command removes the specified movie from memory. This can be useful for prioritizing which movies to unload when memory is low.

Example:

This statement checks whether the largest contiguous block of free memory is less than 100K and unloads the movie Parsifal if it is:

```
if the freeBlock < 100 * 1024 then unLoadMovie "Parsifal"
```

• **updateFrame**

Syntax: updateFrame

This command enters the changes that have been made to the current frame and steps to the next frame. Any objects that were already in the frame when the update session started remain in the frame. You must issue an `updateFrame` command for each frame that you are updating.

This command works during a score generation session only.

Example:

When used in the following handler, the `updateFrame` command enters the changes that have been made to the current frame and steps to the next frame each time Lingo reaches the end of the repeat loop. The number of frames is determined by the argument `numberOfFrames`.

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

• **beginRecording, endRecording, parent-child scripts**

- **updateLock**

Syntax: the updateLock

This movie property determines whether the stage is updated during score recording

- When the updateLock is TRUE, the stage is not updated.
- When the updateLock is FALSE, the stage is updated.

It is important that the updateLock is set to FALSE before updating the score. You can also use the updateLock to prevent unintentional score updating when leaving a frame, such as when temporarily leaving a frame to examine properties in another frame.

You can keep the stage display constant during a score recording session by setting the updateLock movie property to TRUE before Lingo updates the score. If the updateLock is FALSE, the stage updates to show a new frame each time the frame is entered by the updateLock command. If the updateLock is TRUE, the stage doesn't change as new frames are generated.

- **updateMovieEnabled**

Syntax: the updateMovieEnabled

This property specifies whether changes made to the current movie are automatically saved when the movie branches to another movie.

- When the `updateMovieEnabled` property is `TRUE`, changes to the movie are automatically saved when the movie branches to another movie.
- When the `updateMovieEnabled` property is `FALSE`, changes to the movie are not automatically saved when the movie branches to another movie.

The default value is `FALSE`.

Example:

This statement has Director save changes to the current movie whenever the movie branches to another movie.

```
set the updateMovieEnabled = TRUE
```

● **updateStage**

Syntax: `updateStage`

This command redraws the stage immediately. Normally the stage is updated only between frames, but the `updateStage` command updates the stage any time the command is executed from a handler or method.

The `updateStage` command is useful for creating animation within one frame, which is common when animating puppets.

Do not use `updateStage` with the `perFrameHook` property. Otherwise, unexpected results could occur.

Example:

This handler makes the sprite specified by `whichSprite` a puppet sprite, changes the sprite's horizontal and vertical locations, and redraws the stage so that the sprite appears in the new location:

```
on moveRight whichSprite, howFar
  puppetSprite whichSprite, TRUE
  set the locH of sprite whichSprite to
    to the locH of sprite whichSprite + howFar
  updateStage
end moveRight
```


- **value**

Syntax: value(*string*)

This function returns the numerical value of a string. This is useful when making use of a numerical string that the user has typed into a field cast member or data from Xtras and XObjects that return numerical strings.

Examples:

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.

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.

- **string** function

- **version**

Syntax: version

This system variable contains the version string for Macromedia Director. The same string appears the Finder's Get Info dialog box.

Example:

This statement displays the version of Macromedia Director in the message window:

```
put version  
-- "5.0"
```

- **video of member**

Syntax: the video of member *whichCastmember*

This digital video cast member property enables or disables playing the specified digital video cast member.

- When the video of member is TRUE (1), the digital video is enabled.
- When the video of member is FALSE (0), the digital video is disabled.

Example:

This statement turns off the video associated with the cast member Interview:

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

● **videoForWindowsPresent**

Syntax: the videoForWindowsPresent

This movie property indicates whether Video for Windows is present on the computer. It can be tested but not set.

Example:

This statement checks whether Video for Windows is missing and has the playback head go the marker Alternate Scene if it isn't:

```
if the videForWindows = FALSE then go to "Alternate Scene"
```

- **visible of sprite**

Syntax: the visible of sprite *whichSprite*

This sprite property determines whether the sprite specified by *whichSprite* is visible.

- When the `visible of sprite` property is TRUE, the sprite is visible.
- When the `visible of sprite` property is FALSE, the sprite is not visible.

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

Example:

This statement makes sprite 8 visible:

```
set the visible of sprite 8 to TRUE
```

- **visible of window**

Syntax: the visible of window *whichWindow*

This window property determines whether the window specified by *whichWindow* is visible.

- When the visible of window property is TRUE, the window is visible.
- When the visible of window property is FALSE, the window is not visible.

The visible of window property can be tested and set.

Example:

This statement makes the window Control Panel visible:

```
set the visible of window "Control Panel" to TRUE
```

- **voidP**

Syntax: `voidP(variableName)`

This property specifies whether the variable specified by *variableName* has been given an initial value.

- When the `voidP` property is TRUE, the variable has not been given an initial value.
- When the `voidP` property is FALSE, the variable has been given an initial value.

The `voidP` property can be tested but not set.

Example:

This statement checks whether the variable `answer` has been given an initial value:

```
put voidP (answer)
```

- **volume of sound**

Syntax: the volume of sound *whichChannel*

This sound property determines the volume of the sound channel specified by *whichChannel*. Sound channels are numbered 1, 2, 3, 1 and 2 are the channels that appear in the score.

The value of the `volume of sound` property ranges from 0 (silent) to 255 (maximum volume).

The lower the value of the `volume of sound`, the more static or noise you're likely to hear. Using the `soundLevel` may produce less noise, although it offers less control.

Example:

This statement sets the volume of sound channel 2 to 130, which is a medium setting:

```
set the volume of sound 2 to 130
```

- **fadeIn** and **fadeOut** commands; **soundEnabled** and **soundLevel** properties

- **volume of sprite**

Syntax: the volume of sprite *whichSprite*

This property can be used to control the volume of a digital video movie cast member. You can use a cast name or number. The values for volume range from -256 to 256. Values of zero or less are silent.

Example:

This statement sets the volume of the QuickTime movie playing in sprite channel 7 to 256, which is the maximum sound volume:

```
set the volume of sprite 7 to 256
```

- **soundLevel** property

- **width of member**

Syntax: the width of member *whichCastMember*

This cast member property determines the width in pixels of the cast member specified by *whichCastMember*. The `width of member` applies only to bitmap and shape cast members. It does not affect field or button cast members.

The `width of member` property can be tested, but not set.

Example:

This statement assigns the width of member 50 to the variable `height`:

```
put the width of member 50 into height
```

- **height of member** property

- **width of sprite**

Syntax: the width of sprite *whichSprite*

This sprite property determines the horizontal size in pixels of the sprite specified by *whichSprite*. The width applies only to bitmap and shape cast members. It does not affect field or button cast members.

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

Setting this property has no effect on bitmap sprites unless the sprite's `stretch of sprite` property is set to TRUE.

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. When you are changing several sprite properties--or several sprites--you have to use only one `updateStage` command at the end of all the changes.

Examples:

This statement sets the width of sprite 10 to 26 pixels:

```
set the width of sprite 10 to 26
```

This statement assigns the width of sprite number `i + 1` to the variable `howWide`:

```
put the width of sprite (i + 1) into howWide
```

- **height of sprite** and **stretch of sprite** sprite properties; **spriteBox** command

- **window**

Syntax: window *whichWindow*

This keyword refers to the movie window--a window that contains a Director movie--specified by *whichWindow*.

Windows that play movies are useful for creating floating palettes, separate control panels, and windows of different shapes. By using windows that play movies, you can have several movies open at once and allow them to interact.

Examples:

This statement opens the window Control Panel:

```
open window "Control Panel"
```

This statement moves the window Control Panel to the front:

```
moveToFront window "Control Panel"
```

- **close window, moveToBack, moveToFront, and open window** commands

• **windowList**

Syntax: `the windowList`

This property is a list of all the known movie windows.

Examples:

This statement displays all the known movie windows in the message window:

```
put the windowList
```

This statement clears the windowList:

```
set the windowList = []
```

● **windowPresent**

Syntax: `windowPresent ("windowName")`

This function tells whether the object specified by *windowName* is running as a movie in a window. The *windowName* argument must be the window's name as it appears in the `windowList` property.

- When `windowPresent` returns TRUE (1), the object is a movie in a window.
- When `windowPresent` returns FALSE (0), the object isn't a movie in a window.

Example:

This statement tests whether the object `myWindow` is a movie in a window and displays the result in the message window:

```
put windowPresent(myWindow)
```

● **windowType of window**

Syntax: the windowType of window *whichWindow*

This window property specifies the display style of the window specified by *whichWindow*. The possible values are the following:

Numbers 0 to 16, which specifies the window types that correspond to the Macintosh Standard Tool Box numbers.

Number 49 specifies a floating palette window.

In Windows, these numbers create windows with the same functionality but a Windows appearance. Other values for the windowType are possible, but use them with caution, because some modal windows can only be exited by restarting the computer.

Example:

This statement sets the value of the display style of the window Control Panel to 8:

```
set the windowType of window "Control Panel" to 8
```

- **word...of**

Syntax: word *whichWord* of *chunkExpression*
 word *firstWord* to *lastWord* of *chunkExpression*

This chunk expression keyword specifies a word or a range of words in a chunk expression. A word chunk is any sequence of characters delimited by spaces. (Any non-visible character--such as a Tab or Enter--is considered a space.)

The expressions *whichWord*, *firstWord*, and *lastWord* must evaluate to integers that specify a word in the chunk.

Chunk expressions refer to any character, word, item, or line in any source of characters. Sources of characters include fields (field cast members) and variables that hold strings.

Example:

These statements set the variable named `animalList` to the string "fox dog cat" and then insert the word elk before the second word of the list:

```
put "fox dog cat" into animalList  
put "elk" before word 2 of animalList
```

The result is the list "fox elk dog cat".

This statement has the message window display the fifth word of the same string:

```
put word 5 of "fox elk dog cat"
```

Because there is no fifth word in this string, the message window displays two quote marks (""), which indicate an empty string.

- **char...of**, **line...of** , and **item...of** chunk expression keywords; the **number of words in** in chunk function

- **wordWrap of member**

Syntax: the wordWrap of member *whichCastmember*

This field cast member property controls line wrapping.

- When TRUE, the wordWrap of member allows line wrapping.
- When FALSE, the wordWrap of member prevents line wrapping.

Example:

This statement turns the line wrapping off for the field cast member Rokujo:

```
set the wordWrap of member "Rokujo" to FALSE
```

- **xFactoryList**

Syntax: xFactoryList(*whichLibrary*)

This function returns a string list of all the currently available Xtras and XObjects in the XLibrary file specified by the string *whichLibrary*. The XLibrary must have been previously opened with the `openXlib` command. If you specify `EMPTY` for *whichLibrary*, this function returns a list of all Xtras and XObjects in all open XLibraries.

The Xtras and XObjects appear one per line in the returned string list. Each line ends with an Enter character.

Example:

This statement displays the Xtras and XObjects available in the XLibrary named AppleCD XObj:

```
put xfactoryList("AppleCD XObj") "
```

This statement displays the first line of the list of all available Xtras and XObjects in all open XLibraries:

```
put line 1 of xfactoryList(EMPTY)
```

- **Using Xtras**

- **xtra**

Syntax: `xtra whichXtra`

This function returns an instance of the specified Xtra.

Example:

This statement uses the new function to create a new instance of the Xtra Math Whiz and assigns it to the variable `tool`:

```
set tool = new(xtra "Math Whiz")
```

- **Using Xtras**

- **xtras**

- number of xtras

● zoomBox

Syntax: `zoomBox startSprite, endSprite [, delayTicks]`

This command creates a zooming effect, like the expanding windows in the Finder. The zoom effect starts at the bounding rectangle of *startSprite* and finishes at the bounding rectangle of *endSprite*. `zoomBox` uses the following logic when executing:

- 1--Looks for *endSprite* in the current frame, otherwise,
- 2--Looks for *endSprite* in the next frame.

Note, however, that the `zoomBox` command does not work for an *endSprite* in the same channel as *startSprite*.

delayTicks 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
```

• **File menu**

The File menu contains commands for creating Director movies and casts, opening and saving movies, importing and exporting files, creating projectors, and printing.

Click the name of a menu command for more information:

<u>New Movie</u>	<u>Page Setup</u>
<u>New Cast</u>	<u>Print</u>
<u>Open</u>	<u>Send Mail</u>
<u>Close Window</u>	<u>Preferences: General</u>
<u>Save</u>	<u>Preferences: Score</u>
<u>Save As</u>	<u>Preferences: Cast</u>
<u>Save and Compact</u>	<u>Preferences: Paint</u>
<u>Save All</u>	<u>Exit</u>
<u>Revert</u>	
<u>Import</u>	
<u>Export</u>	
<u>Create Projector</u>	

- **New Movie command (File menu)**

Control+N

The New Movie command opens a new, untitled movie. Name the untitled movie by choosing Save from the File menu.

- **New Cast command (File menu)**

Control+Alt+N

The New Cast command creates new internal and external casts.

A cast is a database of graphics, sounds, color palettes, Lingo scripts, buttons, transitions, digital video movies, and text used in a movie.

An internal cast is the cast automatically created when you create a new movie. The internal cast is stored inside the movie file. When you create a projector, internal casts are stored inside the projector file.

External casts are casts stored outside of the movie file and can be shared between movies.

To create a new cast, enter a name for the new cast, choose either Internal or External, and then click Create.

If you choose External, the Used in Current Movie checkbox is checked by default. Click the checkbox if you don't want the external cast used in the current movie.

- **Understanding internal and external casts**
Cast window
Cast Properties

- **Open command (File menu)**

Control+O

The Open command opens an existing Director movie or external cast. When you choose Open, a directory dialog box appears.

The dialog box only lists movies and casts created by the Macintosh version of Director, or movies with a .DIR extension created with the Windows version of Director.

When you open a movie with linked external casts, Director opens the external casts as well. If it cannot find them in the specified location, it prompts you to choose a new location.

- **Update Movies command (Xtras menu)**

- **Close Window command (File menu)**

Control+F4

This command closes the current window.

- **Save command (File menu)**

Control+S

The Save command saves the current movie, replaces the previous version, and saves all casts (internal and external) linked to the movie.

If an external cast is the front-most window, then the external cast is saved, not the movie.

The first time you save a movie with linked external casts, Director prompts you to enter a name and location for each cast.

- To change the movie's color depth, change the monitor's color depth before saving the movie.

- **Save As command (File menu)**

Save All command (File menu)

Update Movies command (Xtras menu)

Understanding internal and external casts

- **Save As command (File menu)**

Control+Shift+S

Use Save As to name and save a movie, save the movie under a different name, or to save it to a different disk. Enter the new name of the movie to name it, or click the Drive button to save to a different drive.

The first time you save a movie with linked external casts, Director prompts you to enter a name and location for each cast.

- To change the movie's color depth, change the monitor's color depth before saving the movie.

- **Save All command (File menu)**

Save and Compact (File menu)

Update Movies command (Xtras menu)

Understanding internal and external casts

- **Save and Compact (File menu)**

Control+Option+S

Use save and compact to save the movie so that it is optimized for playback. Since this operation reorders the cast and compacts the file, it takes longer, especially if you are saving a very large file. However, this command produces smaller and more efficient movies.

- **Save As command (File menu)**

- **Save All command (File menu)**

- **Update Movies command (Xtras menu)**

- **Understanding internal and external casts**

- **Save All command (File menu)**

Use Save All to save the movie and all external cast files, linked and unlinked. If the movie or any casts have not been saved before, a directory appears.

- **Save As command (File menu)**
Save and Compact (File menu)
Understanding internal and external casts

- **Revert command (File menu)**

The Revert command opens the last saved version of the current movie. This command is dimmed if you have not made any changes or if you are working on a new untitled movie that you have not yet saved.

- **Import command (File menu)**

Control+R

When you choose Import, Director opens a cast window and displays a dialog box so you can choose the file you want to import.

Note: Director 5.0 for Windows supports the following file types in addition to those also supported by Director for Macintosh: JPEG, CompuServe GIF, TIFF, EPS, Photo CD, Windows metafiles, and FCC and FCI.

- **Dialog box options**

- **Linked** gives you the option to create a link to a PICT file, AIFF sound file, or a Director movie when you import rather than copying the contents of the file into your movie.
- **As PICT** option is only available when you import a Macintosh PICT file. If checked, the image is imported and pasted into the cast as a PICT cast member. If not checked, Director converts the imported image to a bitmapped cast member.
Note: Digital video movies are always linked.
- **Add** and **Add All** buttons let you build a list of multiple files to import.
- **Options** provides further options specific to the type of file you're importing.

- **Linking to a file**
Importing text
Importing PICT files
Importing Director movies
Importing digital video movies

- **Image Options**

If you import a bitmap with a color depth or color palette different than the current movie, the Image Options dialog box appears. You can choose to import the bitmap at its original color depth or at the stage color depth. You also have the choice of importing the image's color palette, or remapping the image's colors to a palette already in the movie.

- In many cases, it's easiest to change the image's color depth to the depth of the stage, and remap the image to the color palette used in the rest of the movie. If you are not concerned with the exact colors that display on stage, remap everything to the system palette.

Dialog box options

Color Depth selects the color depth you want for the cast member.

- **Image** specifies the color depth of the current image. Select this option if you want to import the image at this color depth.
- **Stage** specifies the color depth of the current stage. Select this option if you want to import the file at this color depth.

Palette specifies options for importing 2-, 4-, or 8-bit images.

- **Import** specifies importing the image with its color palette.
- **Remap To** makes Director replace the image's colors with the most similar solid colors in palette you select from the pop-up menu.
- **Dither** blends the colors in the new palette to approximate the original colors in the graphic..

Same Settings for Remaining Images applies the current settings to all the remaining files you selected for importing.

Note: If you change 16-, 24-, or 32-bit cast members to 8-bits or less, you need to remap them to an existing color palette.

- **Understanding color depth**
Understanding color palettes

● **Export command (File menu)**

Control+Shift+R

Exports frames of Director movies from Director so you can save images as stills, or create digital video movies. You can choose to export frames of movies as AVI and PICT files.

Dialog box options

Export options determine which frames to export:

- Current Frame exports the current frame on the stage. This is the default.
- Selected Frames exports the selected frames in the score window.
- All exports all frames.
- Frame Range exports only the range of frames that begin and end with the frame numbers you enter in the Begin: and End: boxes.

Include:

- Every Frame exports all frames in the selected range.
- One in Every *N* Frames exports only the frames at the interval you specify in the box.
- Frames With Markers exports frames with markers set in the score window.
- Frames with Artwork Changes in Channel exports frames only when a cast member changes in the channel you specify in the box.

Format specifies the exported file format from the pop-up menu. File formats you can export are Video for Windows (AVI) and DIB (Device Independent Bitmap).

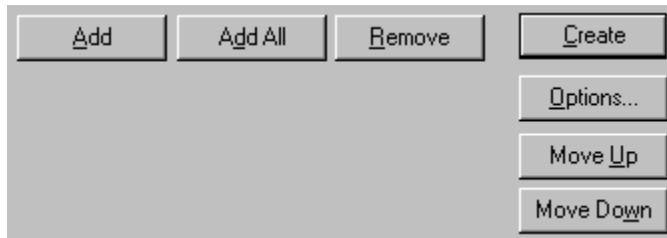
- Video for Windows Options is dimmed unless you choose Video for Windows from the Format pop-up. Specify the export frames per second (fps) interval in the box.

When you click Export, a directory dialog box appears, allowing you to name the file. If you are saving in AVI format, only one file will be created. If you are saving in DIB format, Director automatically creates one file for each frame, attaching the corresponding frame number to each file. For example, if the name of the exported file is "Myfile", Frame 1 will be exported to a file named "Myfi0001.dib for Windows 3.1 and NT" or "Myfile0001" for Windows '95.

- **Create Projector command ([File menu](#))**

The Create Projector command creates a play-only version of a Director movie, called a projector. Use the Create Projector dialog box to add movies and external casts from the Source Folder to the projector's play list. See [Creating projectors](#)

Click a dialog box option for more information:



Movies created with previous versions of Director are not listed in the dialog box. To include movies that were created with a previous version of Director, you must first open the movies in the current version of Director and save them, or use the [Update Movies](#) command to convert them.

If a movie is open when you choose this command, Director closes the current movie.

Add moves the selected source movie to the projector's play list.

Add All adds all movies in the current folder to the play list.

Move Up and Move Down moves a selected movie higher up or further down in the play list.

You can include more than one interactive movie in the projector file list. You can also use Lingo to "go to" or "play" files while running a projector under Lingo control.

Remove takes the selected movie off the play list.

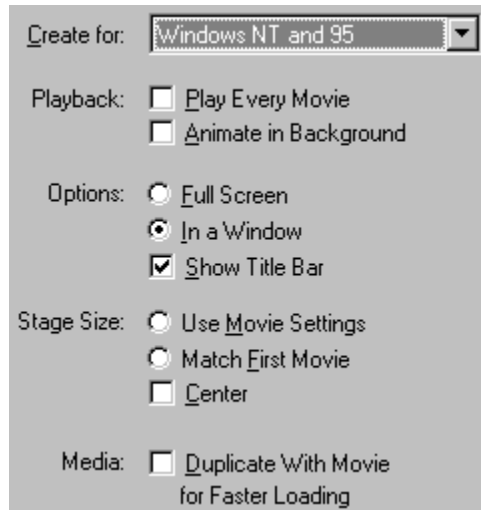
Create assembles the projector file. Click Done to dismiss the dialog box.

Options opens the **Projector Options dialog box** that specifies additional preferences for creating the projector.

● Projector Options dialog box

The Projector Options dialog box specifies additional preferences for creating the projector.

Click a dialog box option for more information:



Note: These settings override any movie preferences you set in Movie Properties and apply to all movies in the projector.

Create for--Select the type of systems you wish to support:

- Windows 95 and NT allows the projector to run on all Windows 32-bit operating systems.
- Windows 3.1 allows the projector to run on Windows 3.1. A Windows 3.1 projector will also run on Windows 95 and NT.

Playback:

- Play Every Movie specifies that the projector plays all movies in the play list. Otherwise, the projector only plays the first movie in the play list (unless other movies are called by Lingo from the first movie). In a projector with Play Every Movie checked, pressing Control+period will go to the next movie and Control+Q will quit.
- Animate in Background allows the movie to continue playing if a user clicks outside the stage. This is useful if you are using Apple Events. If not checked, the movie stops playing if the user clicks outside the stage.

Options:

- Full Screen displays the movie full screen, placing the menu bar (if there is one) at the top of the screen and hiding all of the desktop. If there's a menu, it overlays the top of the stage.
- In a Window displays the movie in a normal window, without taking over the screen. The window is not resizeable.
- Show Title Bar is available only if In a Window is selected. If checked, the window where the movie appears has a title bar. The window is only moveable if it has a title bar.

Stage Size:

- Use Movie Settings uses the same stage size of the new movie or matches the size of the current movie.
- Match First Movie repositions and resizes based on the first movie in the projector.
- Center centers the stage on the screen, which is useful if the stage size is smaller than the screen size. Otherwise, the movie plays using its original stage position. In Windows, projectors are always centered.

Media:

- Duplicate Cast Members for Faster Loading makes copies of cast members in the order they are used in the movie. This makes the file larger, but improves speed of loading.

- **Page Setup command (File menu)**

Control+Shift+P

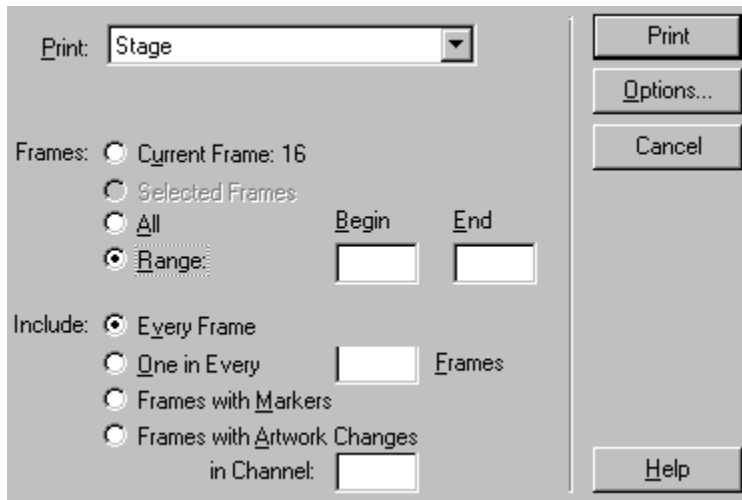
The Page Setup command offers options for determining how a page is to be printed. The dialog box that you see depends on the type of printer you use.

● **Print command (File menu)**

Control+P

The Print command lets you print your movie in a variety of ways.

Click a dialog box option for more information:



Print lets you choose what part of your movie you want to print. You can print an image of the stage, the score, all scripts or a range of scripts (movie, cast, score, and sprite scripts), cast text, cast art, cast thumbnails, and the comments in the markers window.

The Scripts, Cast Text, Cast Art, or Cast Thumbnails print options let you choose from a range of cast and cast members--internal or external. Information displayed in the print dialog depends on the selection to be printed.

When you use the Print command, cast members in each frame are merged into a single image. If you press the Alt key while clicking the Print button, each cast member in a given frame is imaged as a separate PICT. If you are printing multiple frames on a page, printing with this option causes shape elements (text, rectangles, lines, and circles) to print better. However, printing with this option may take much longer.

Frames controls which frames of your movie are printed.

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

Include lets you specify which frames to print.

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

Options displays a dialog box that lets you adjust the layout of the items you choose to print. The image at the left of the dialog box previews the layout options.

- Image Size prints your document at 100% or scales it to 50% or 25% size.
- Frame Printing Options allows you to print a border around each frame, the frame number, registration marks, storyboard format, and marker comments associated with each frame.
- Storyboard format checkbox is only available when you select 50%- or 25%-size images to print.
- Date and Filename in Header prints a header on each page. The header consists of the name of the Director movie and the current date.
- Custom Footer prints a footer on each page. Type the footer in the field.

• **Send Mail (File menu)**

Control+P

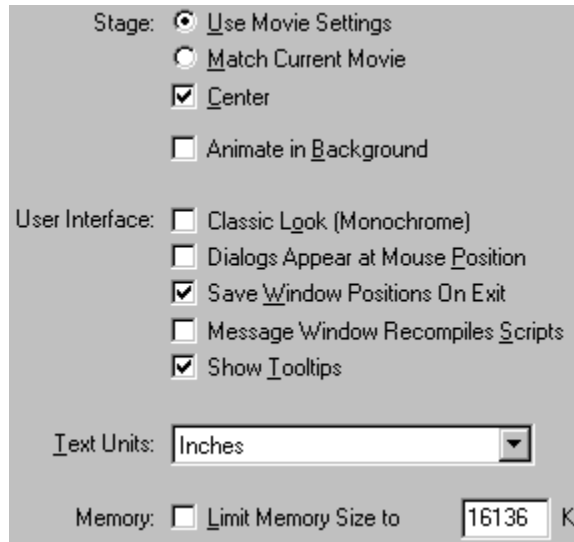
Send Mail mails an open movie, together with any other information you provide, to the user you specify in a dialog. (The dialog is supplied by the mail-system you have installed, and may differ between systems.) The command is only enabled when electronic mail software is installed on your machine under Windows 95 and NT. It is not supported on Windows 3.1.

• **General Preferences command (File menu)**

Control+U

The General Preferences command allows you to modify some of Director's default settings.

Click a dialog box option for more information:



The screenshot shows the General Preferences dialog box with the following settings:

- Stage:**
 - Use Movie Settings
 - Match Current Movie
 - Center
 - Animate in Background
- User Interface:**
 - Classic Look (Monochrome)
 - Dialogs Appear at Mouse Position
 - Save Window Positions On Exit
 - Message Window Recompiles Scripts
 - Show Tooltips
- Text Units:** Inches (selected in a dropdown menu)
- Memory:** Limit Memory Size to 16136 K

Stage size specifies the size and location of the stage.

- Use **Movie Settings** sets the stage size to the movie's stage size and location.
- **Match Current Movie** opens the new movie in the stage of the movie that's currently open.
- **Center** positions the stage in the center of the screen, which is useful if the stage size is smaller than the screen size. Otherwise, the movie plays using its original stage position.
- **Animate in Background** runs your animation in the background while you are working with other applications. When you are running animation in the background, the stage remains on the screen and the active application window appears in front of the stage.

Use **Movie Properties** on the **Modify** menu to specify the exact size of the stage.

User Interface

- Classic Look (Monochrome) switches to a black and white user interface. Using a black and white user interface improves performance if you switch color palettes, since Director doesn't have to update its color user interface to match the colors in the new palette each time you switch palettes. In addition, working with a black and white user interface may be less distracting as you work with the color palettes in your movie. For example, if you are working on an animation that uses multiple palettes and/or color cycling, using a black and white user interface may be less distracting.
- Dialogs Appear At Mouse Position displays dialog boxes at the mouse position. If this option is not checked, dialog boxes are centered on the monitor that contains the menu bar.
- Save Window Positions on Exit saves the positions of all open windows every time you quit.
- Message Window Recompiles Scripts is checked by default. If deselected, scripts should be manually recompiled using the Recompile All Scripts command before entering Lingo in the message window.
- Show Tooltips is checked by default. Deselect to turn off the definitions that appear when you hold the pointer over tools.

Text Ruler specifies inches, centimeters, or pixels for the units of measure displayed on the ruler in the text and field windows.

Memory limits the amount of memory Director uses. Click Limit Memory Size to, and then enter the number of Kilobytes in the box on the right.

● **Score Window Preferences command (File menu)**

This command controls display options in the score window.

Click a dialog box option for more information:

Display Options:	<input type="checkbox"/>	A <u>llow</u> Colored Cells
	<input type="checkbox"/>	E <u>nlarged</u> Cells
	<input checked="" type="checkbox"/>	P <u>layback</u> Head Follows Selection
	<input checked="" type="checkbox"/>	D <u>rag</u> and Drop Enabled
Extended Display:	<input checked="" type="checkbox"/>	Cast Member <u>T</u> ype, Motion, Blend
	<input checked="" type="checkbox"/>	Cast Member <u>N</u> umber
	<input checked="" type="checkbox"/>	I <u>nk</u> Mode
	<input checked="" type="checkbox"/>	S <u>cript</u> Code
	<input checked="" type="checkbox"/>	X and Y Location
	<input type="checkbox"/>	C <u>hange</u> in X and Y Location

Display Options determines how the score window will look on screen.

- **Allow Colored Cells** allows you to choose a color for selected cells using the cell color selector on the left side of the score window. Otherwise, the cell color selector is hidden. If you've already applied color to cells, unchecking this option hides cell colors but doesn't remove them. Score window scrolling performance is faster if you hide cell colors.
- **Magnified Cells** enlarges the cells in the score window for better viewing. You can perform all the regular score operations while the view is magnified.
- **Playback Head Follows Selection** toggles between two ways of selecting frames in the score. If checked, the playback head travels as you make a selection in the score window. You can see the selected frames on the stage as you select them. If not checked, your selection in the score has no effect on the playback head and does not advance the movie as you make a selection. You might want to turn this option off if the score includes frames that take a long time to draw, such as frames with blend ink applied to them.

You can temporarily switch this option to its opposite setting by pressing the Alt key while making a selection in the score window.
- **Drag and Drop** allows you to turn off drag and drop in the score. Pressing the Spacebar while the score window is open temporarily overrides this setting.

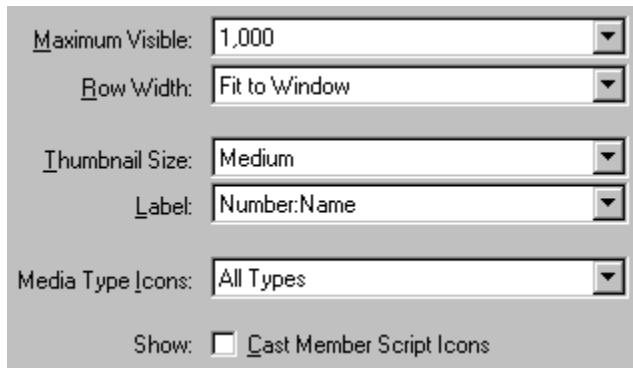
Extended Display Information options let you choose the notation information that appears in the numbered sprite channels when extended display is on. You can turn Extended display on with the score window's Display pop-up menu.

- Cast Member Type, Motion, Blend displays the cast member type (text, PICT, or bitmap) and a directional arrow to indicate the cast member's position with respect to the previous cast member in that channel. It also indicates whether a cast member has a blend percentage applied to it.
- Cast Member Number displays the cast member position number from the cast window. If the cast member displays its name (as chosen in the Cast Window Preferences dialog box), then the first few letters of the cast member name are instead displayed.
- Ink Mode displays the type of ink applied to the cast member in that cell.
- Script Code displays the number of the script associated with that cell or a plus (+) sign if the script is a cast member script.
- X & Y Coordinates show the screen coordinates of the cast member.
- Change in X and Y Location indicates the change in X and Y coordinates relative to the previous cast member in that channel.

- **Cast Window Preferences command ([File menu](#))**

Cast Window Preferences displays a dialog box that lets you control the appearance of the current cast window. Before you choose Cast Preferences, make sure that the cast you want to change is active. The title bar displays the name of the cast you are changing.

Click a dialog box option for more information:



The image shows a dialog box with the following settings:

- Maximum Visible: 1,000
- Row Width: Fit to Window
- Thumbnail Size: Medium
- Label: Number:Name
- Media Type Icons: All Types
- Show: Cast Member Script Icons

Maximum Visible specifies the maximum number of cast members displayed in the cast window. Note that this option does not limit the actual number of cast members that can exist in the cast. If you have a small number of cast members, you can hide the remaining unused cast positions and make better use of the vertical scroll bar. The default is 1000.

Row Width determines how many thumbnails are displayed in each row in the cast window. Eight, Ten, and Twenty specify fixed-row widths that are independent of the window size; if the cast window is smaller horizontally than the width of the cast row, you must use the horizontal scroll bar to reveal the rest of the cast. The Fit to Window option automatically adjusts the number of cast members per row to fit the current width of the cast window. In this mode, the horizontal scroll bar is disabled, since the entire width of the cast is always in view. The default is Fit to Window.

Thumbnail Size sets the size of each cast thumbnail image displayed in the cast window. Thumbnails always maintain the standard 4:3 aspect ratio.

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

- If thumbnails appear "fuzzy", they were probably created at a small size and are now set to a larger size. To fix this problem, change the cast window preferences thumbnail setting to a larger size. Click OK when the alert message asks you if thumbnails should be regenerated.

Label selects the display format of the cast member ID displayed below each cast thumbnail image in the cast window. The chosen format is also used in other windows, whenever a cast ID is displayed. The default is Number:Name.

- Number displays cast number in decimal format.
- Name displays cast name, if one exists; otherwise displays cast number in decimal format.
- Number:Name displays cast number (in decimal format) and cast name, separated by a colon, i.e., "340:Snoopy". If a name does not exist, it just displays the cast number in decimal format. This is the default.

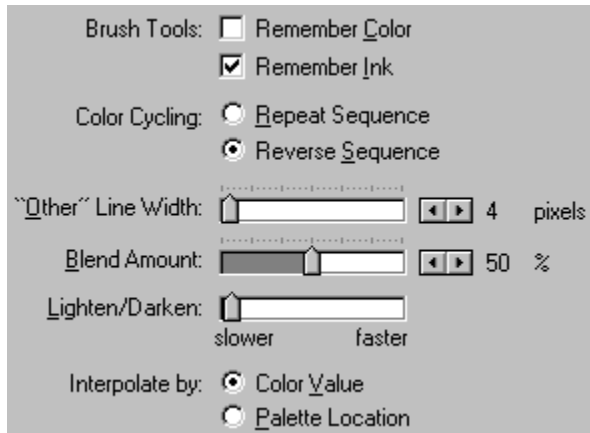
Media Type Icons determines if Director displays an icon in the lower right corner of each cast member, indicating the cast member's type.

Cast Member Script Icons indicate that a cast member has a script attached to it by displaying a script indicator in its lower left corner.

- **Paint Window Preferences command (File menu)**

The Paint Window Preferences command allows you to modify the settings of a number of tools and drawing methods in the paint window.

Click a dialog box option for more information:



- **Paint window**

Brush Tools allows you to set brush tools to remember the last color or ink used.

- Remember Color remembers the last color used with a tool and stays selected for the next time you use the paintbrush or air brush.
- Remember Ink remembers the last ink used with a tool and stays selected for the next time you use the paintbrush or air brush.

- Cycling** controls the way colors cycle when you draw with cycling ink.
- Repeat Sequence causes colors to cycle from foreground to destination and then repeat foreground to destination.
 - Reverse Sequence causes colors to cycle from the foreground to the destination color and then destination to foreground.

Other Line Width allows you to set a thicker line width than the widths available in the paint window. The width you set will be the width that appears when you draw a line (after selecting Other in the tool palette line width selector).

Blend Amount sets the opacity of the selected color when using the Blend ink effect in the paint window. You can vary the blend value between 0 and 100 percent.

Lighten/Darken sets the rate at which artwork changes when you use the Darken or Lighten effect in the paint window.

Interpolate by determines how colors are used when using smooth, lighten, darken, or cycle effects.

- Color Value ignores the order of the colors in the palette and produces a continuous blend of the foreground and destination colors.
- Palette Location uses all the colors in the palette between the foreground and destination colors.

- **Exit command (File menu)**

Alt+F4

The Exit command exits Director.

• **Edit menu**

The Edit menu contains standard commands for editing.

Click a command for more information:

Undo

Repeat

Cut

Copy

Paste

Paste Special

Clear

Duplicate

Select All

Invert Selection

Find Text

Find Handler

Find Cast Member

Find Selection

Find Again

Replace Again

Exchange Cast Members

Edit Cast Member

- **Undo command (Edit menu)**

Control+Z

Undo reverses your last action. Undo works with most commands you use while writing, drawing, and animating.

- **Repeat command (Edit menu)**

Control+Y

Repeat command repeats your last action. Repeat works with paint effects commands.

If using **Photoshop filters** to modify cast members, this command repeats the effect of the last filter used.

- **Cut command (Edit menu)**

Control+X

The Cut command removes the selected object from its current location and places it on the Clipboard. It can then be pasted to another location.

- **Copy command (Edit menu)**

Control+C

The Copy command makes a copy of the selected colors, text, art, or sequence of art and places that copy on the Clipboard.

- **Paste command (Edit menu)**

Control+V

The Paste command pastes the contents of the Clipboard in a selected location.

- **Paste Special command (Edit menu)**

Use this command to paste a sequence of sprites at the point on the stage where a previous sequence ended. When you use this command, Director automatically adjusts the positions of cast members on the stage so that the first cast member in the pasted sequence follows the last cast member in the original sequence.

For example, to animate a ball bouncing across the stage with a sequence of five sprites that describe one bounce, you can use Paste Relative to make the second sequence of sprites start where the first sequence ended. The effect is that the two sequences are chained, one after another, in one smooth, continuous motion. This works for any repetitive sequence of sprites.

When selecting cells in a sequence to be pasted relative to the original sequence, make sure the same cast member is at the beginning and end of the sequence, and that you overlap the first cell in the copy with the last cell in the original sequence.

- **Using OLE Objects in Director, OLE Object command**

- **Clear command (Edit menu)**

Clear removes the selected cells or cast members without saving to the Clipboard. When the score window is active, Clear removes the contents of selected cells. When a cast window is active, Clear removes selected cast members.

Delete is the keyboard shortcut for this command.

- **Duplicate command (Edit menu) Control+D**

The Duplicate command duplicates the selected cast member and pastes the duplicate into the next available position in the cast window.

If the paint, text, digital video, or script window is open, this command duplicates the selected cast member and places it in the edit window. The name and registration point of the duplicate is the same as the name and registration point of the original cast member.

This command cannot be used with a cast member that is part of a shared cast.

This is a quick way to create a series of cast members for frame-by-frame animation. Duplicate a cast member, change it slightly, duplicate the changed cast member, alter and duplicate it again, and so on.

- If the cast window is front-most, you can select multiple cast members and use this command to duplicate the entire selection at the same time.

You can also duplicate by Option-dragging selected cast members into empty cast spaces.

- **New Window**

- **Select All command (Edit menu)**

Control+A

Select All highlights all the selectable items in the active window.

- **Invert Selection command (Edit menu)**

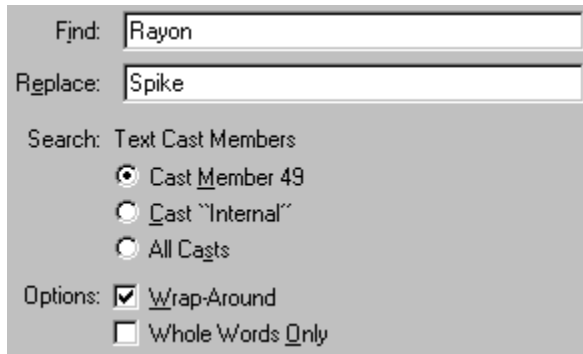
When you choose Invert Selection after choosing a color or range of colors in the color palettes window, your selection is replaced by a new selection, which consists of all the colors that were not part of your original selection. This command only works when working with color palettes.

● Find Text command (Edit menu)

Control+F

Find Text lets you quickly search for and replace text in the text, field, or script windows. All searches start at the insertion point and work forward.

Click a dialog box option for more information:



The image shows a dialog box for finding and replacing text. It has two input fields: "Find:" containing "Rayon" and "Replace:" containing "Spike". Below these are search options: "Text Cast Members" (selected), "Cast Member 49", "Cast 'Internal'", and "All Casts". At the bottom are options: "Wrap-Around" (checked) and "Whole Words Only" (unchecked).

Find:	Rayon
Replace:	Spike
Search:	Text Cast Members
	<input checked="" type="radio"/> Cast Member 49
	<input type="radio"/> Cast "Internal"
	<input type="radio"/> All Casts
Options:	<input checked="" type="checkbox"/> Wrap-Around
	<input type="checkbox"/> Whole Words Only

Find field specifies the text you want to find. Searching is not case-sensitive: ThisHandler, thisHandler, and THISHANDLER are all the same for search purposes.

Replace field specifies the replacement text.

Search

- Cast Member specifies the cast member to be searched.
- Cast specifies whether Director searches Internal or External casts.
- All Casts specifies whether Director searches only the current cast member, or searches all cast members of the same type (either text or script, depending on where you initiated the search). If checked, Director searches all cast members of the same type, beginning with the current cast member, and wrapping around to the first cast member if Wrap-Around Search is checked. If All Cast Members is not checked, Director only searches the current cast member's text.

Options

- Wrap-Around specifies whether or not Director begins the search again once it reaches the end of the current text. If this option is checked but All Casts is not checked, Director continues searching from the top of the current text after it reaches the bottom of the window. If both options are checked, Director searches all cast members of the same type (either text or script, depending on where you initiated the search), beginning with the currently selected cast member, and wrapping around to the first cast member of that type if necessary.
- Whole Words Only searches for occurrences of the specified whole word.

- **Find Handler command (Edit menu)**

Control+Shift+;

Find Handler lets you view the names of all handlers in the current script or movie. You can also use this command to open the script window that contains the selected handler.

To find a handler:

Choose Find Handler from the Edit menu. Click options in the dialog box to display different lists of handlers. Select a handler and click OK to open the script window in which the handler is defined.

Dialog box options

Search determines what Script will be searched.

- The Current Script Only lists the handlers defined in the current script and is only available if you choose Find Handler from a script window.

View by determines how handlers will be listed.

- Name lists handlers alphabetically, by name.
- Script Order lists handlers in the order in which they appear in their respective scripts. Handlers from different scripts are listed in the order in which the scripts appear in the cast window.

● **Find Cast Member command (Edit menu)**

Control+;

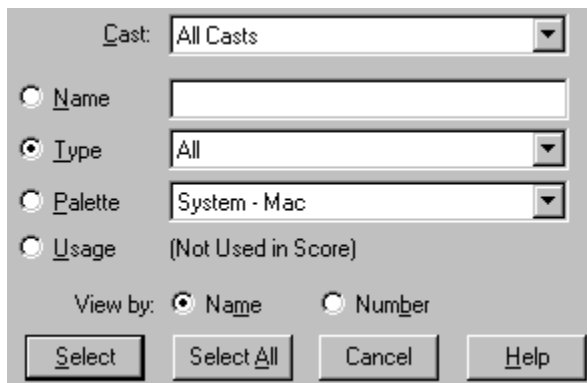
Use this command to find cast members by name, by type, by color palette, or by their use in the current score. This command is useful for identifying cast members to clear from your movie or to remap to another palette.

To find cast members:

Select the cast you want to search from and then choose one of the search options.

- Select a cast member in the list and click Select (or double-click a cast member) to close the dialog box and select the cast member in the cast.
- Click Select All to close the dialog box and select all matching cast members in the cast.
- To quickly select cast members by name, type the first few letters of the name, and the dialog box automatically displays a list of cast members whose name begins with the letters you type.

Click a dialog box option for more information:



The image shows a dialog box for finding cast members. It has a title bar and several controls:

- Cast:** A dropdown menu currently set to "All Casts".
- Name:** A radio button option, currently unselected.
- Type:** A radio button option, currently selected, with a dropdown menu set to "All".
- Palette:** A radio button option, currently unselected, with a dropdown menu set to "System - Mac".
- Usage:** A radio button option, currently unselected, with the text "(Not Used in Score)" next to it.
- View by:** Two radio button options: "Name" (selected) and "Number".
- Buttons:** Four buttons at the bottom: "Select", "Select All", "Cancel", and "Help".

Cast selects the cast to search.

Name searches for all cast members with names that begin with the characters you enter. Click Name and enter the cast member name you want to search for in the field to the right.

Type finds all the cast members of a particular type. Select Type and choose an option from the menu.

Palette searches for all cast members using a certain palette.

Usage finds all cast members not used in the score. Keep in mind, however, that a cast member that is not used in the score may still be used in a Lingo command.

View by determines how cast members are displayed on the list.

- Name views cast members by name.
- Number views by number.

Select button selects the cast member highlighted in the Find Cast Member dialog box.

Select All button selects all the cast members in the Find Cast Member dialog box.

- **Find Selection command (Edit menu)**

Control+H

Select a cast member in either the cast window or the score and use Find Selection to find the next occurrence of that cast member in the score.

Find Again repeats the previous find, initiated in either the cast or score window.

- **Find Again command (Edit menu)**

Control+Alt+F

Find Again finds the next occurrence of the text you entered in the Find field in the Find dialog box.

• **Replace Again command (Edit menu)**

Control+Alt+E

Choose Replace Again to replace the next instance of the text you entered in the Find field in the Find Text dialog box.

● **Exchange Cast Members command (Edit menu)** **Control+E**

The Exchange Cast Members command replaces the cast member selected on the stage or in the score with the cast member selected in the cast window. When you use Exchange Cast Members, the registration point of the new cast member lines up with the registration point of the old cast member.

- **Edit Cast Member command (Edit menu)**

This command is only available if there is a selection in the cast window. It displays the appropriate editing window for the selected cast member. For example, if you select a bitmap cast member and choose this command, Director opens the paint window for the selected cast member. For cast members that don't have editing windows (e.g., shape, PICT, sound, movie, and film loop cast members) Director displays the cast member's **Cast Member Properties** dialog box.

Shortcut: Double-click a cast member in the cast window

- **View menu**

Click a command or submenu for more information:

Display
Marker Next
Marker Previous
Zoom In
Zoom Out
Zoom 100, 200, 400, 800%
Panel
Rulers
Grid: Show
Grid: Snap To
Grid: Settings
Onion Skin

Depending on which window you have in front, the Panel command name changes (such as Paint Tools or Text Toolbar).

- **Display command (View menu)**

Choose the type of display for the score window.

- **Display pop-up menu** (score window)

● **Marker next command (View menu)**

Control+right

Choose an option to move to the next marker in the score, or select the name of a marker to move there directly.

- **Marker previous command (View menu)**

Control+left

Choose an option to move to the previous marker in the score, or select the name of a marker to move there directly.

- **Zoom submenu (View menu)**

Zoom in Control+Add

Choose zoom in to reduce size of the paint window. You can zoom between 100% to 800%.

When you zoom in, a smaller representation of the 100% size artwork appears in the upper right corner of the paint window. To return to normal size, click inside the smaller window or choose Zoom Out.

Zoom out Control+Subtract

Choose zoom out to enlarge the paint window. You can zoom from 100% to 800%.

To return to normal size, click inside the smaller window or choose Zoom In.

Zoom 100, 200, 400, 800%

Choose one of these to view the paint window at a specific scale.

- You can also magnify your artwork by double-clicking the pencil tool in the paint window's tool palette, or by Control-clicking the area you want to enlarge with any of the paint tools.

- **Panel command (View menu)**

Depending on which window you have in front, the Panel command name changes (such as Paint Tools or Text Toolbar).

Choose Panel from the View menu to show or hide tool panels in the text, paint, color palette, and script windows.

- **Ruler command (View menu)**

Choose Ruler from the View menu to show or hide rulers in the paint or text windows.

The default setting for the unit of measure is inches. You can change the setting for text to picas, centimeters, or pixels using **General Preferences** in the File menu.

In the paint window, change the unit of measure by clicking the upper left corner where the vertical and horizontal rulers meet. The zero point of the rulers can be moved to a new location by dragging from the corner of the rulers. The current position of the pointer is indicated by dotted lines in the rulers.

- **Show Grid command (View menu)**

Command+Shift+Alt+G

The commands on the Grid submenu control the grid on the stage. Use the grid to align and place sprites on the stage.

Show Grid shows or hides the grid on the stage. A checkmark indicates the grid is displayed.

- **Snap to Grid**
Grid Settings

- **Snap to Grid command (View menu)**

Control+Option-G

The commands on the Grid submenu control the grid on the stage. Use the grid to align and place sprites on the stage.

Snap To Grid makes all sprites move to the nearest grid line when you move them. A check mark indicates the option is on.

- **Show Grid**
Grid Settings
Align

- **Grid Settings command (View menu)**

The commands on the Grid submenu control the grid on the stage. Use the grid to align and place sprites on the stage.

Dialog box options

Grid Settings defines the settings for the grid. In the Grid Settings dialog box you can define the following settings:

- Spacing--Enter the number of pixels between the vertical and horizontal grid markings.
- Display--Choose either lines or dots.
- Color--Select a color for the grid from the pop-up color palette.

- **Snap to Grid**
Show Grid

● **Onion Skin command (View menu)**

Onion skinning allows you to view several cast members blended into an image in the paint window. The blended-in cast members serve as a reference while you paint a new cast member.

To activate onion skinning, open the paint window, and choose Onion Skin from the View menu. The onion skin toolbar appears. Click the Toggle Onion Skin button in the toolbar to enable onion skinning. See **Understanding onion skinning**.

Click a toolbar button for more information:



Toggle Onion Skinning is an on/off button. When off, no reference images are blended into the paint window's drawing area. When on, selected reference images are blended into the drawing area. The current cast member is then drawn on top. **Shortcut:** Control-Alt-K enables or disables onion skinning.

Preceding Cast Members indicates the number of cast members immediately preceding the current cast member shown as reference images in the paint window. Images further away from the current cast member are dimmer than images closer to it in the cast.

Following Cast Members indicates the number of cast members immediately following the current cast member that will be shown as reference images in the paint window. Images further away from the current cast member are dimmer than images closer to it in the cast.

Set Background selects the current cast member to be the background cast member. This cast member will be used as a reference image if Show Background is on.

When **Show Background** is on, the background cast member is blended into the paint window as a reference image.

When you click **Track Background**, Director marks the current foreground and background cast members as the beginning members of a foreground and background series of cast members. From then on, as you select different foreground cast members, the corresponding member of the background series is used as a reference image.

- **Insert menu**

Click a command or submenu for more information:

Frame command

Remove Frame command

Media Element: Bitmap

Media Element: Text

Media Element: Color Palette

Control: Push Button, Radio Button, or Check Box

Control: Field

OLE Object

Film Loop

- **Frame command (Insert menu)**

Control+]

The Insert Frame command inserts a frame into your movie at the location of the playback head and a copy of the current frame is added to the score. This can cause an apparent pause in the animation as two identical frames are played.

The new frame is inserted for all channels, and adds a frame to the movie's length.

● **Remove Frame command (Insert menu)**

Control+[

Remove Frame deletes the frame at the location of the playback head, making the movie one frame shorter in length.

- **Media Element: Bitmap command (Insert menu)**

Choose the Insert Bitmap command to create a new bitmapped cast member. When you choose this command, Director opens the paint window.

Unless you first select an empty position in the cast window, Director assigns the new cast member to the first empty position after the current cast member.

- **Media Element: Text command (Insert menu)**

Choose the Insert Text command to create a new text cast member and open the text window.

Unless you first select an empty position in the cast window, Director assigns the new cast member to the first empty position after the current cast member.

- **Understanding text and fields**
Text window

- **Media Element: Color Palette command (Insert menu)**

Choose the Color Palette command to create a new palette cast member. When you choose this command a dialog box appears and prompts you to enter the name of the new color palette. Once you enter a name and click OK, Director opens the color palettes window and you can change the new palette as needed.

Unless you first select an empty position in the cast window, Director assigns the new cast member to the first empty position after the current cast member.

- **Color Palettes command (Window menu)**

- **Control: Push Button, Radio Button, or Check Box (Insert menu)**

Choose one of these commands to create a button or checkbox cast member on the stage. First, select the channel and frame in the score where you want to create the button or checkbox, and then choose the command. A button or checkbox will appear on the stage and you can begin entering text.

Unless you first select an empty position in the cast window, Director assigns the new cast member to the first empty position after the current cast member.

You must then attach a **script** to a button or checkbox so that it responds appropriately when clicked.

- **Control: Field command (Insert menu)**

Choose the Insert Field command to create a new field cast member. First, select the channel and frame in the score where you want to create the new field, and then select the Filed command. A field will appear on the stage and you can begin entering text.

Unless you first select an empty position in the cast window, Director assigns the new cast member to the first empty position after the current cast member.

Use fields primarily for text that must be editable in a projector. For most text applications you should use normal text.

- **Understanding text and fields**
Field (Window menu)

- **OLE Object command (Insert menu)**

The OLE Object command creates cast members from OLE objects. You can create a new OLE Object using the source application, or by linking to an existing file.

Note: OLE objects only work in Windows 95 and Windows NT.

To create an OLE cast member:

1. Choose OLE Object from the Insert menu.
2. In the Insert Object dialog box, choose Create New or Create from File.

If you choose Create New, choose the type of OLE Object you want from the Object Type list. The Objects available depend on the OLE-compatible applications installed in your system. The source application of the OLE server is launched. When you finish creating the object and choose Update from the File menu (in the source application), the object appears in Director as a cast member.

If you choose Create from File, enter the path to the file or use Browse to select the file. The object is immediately placed in the cast.

4. Click OK to continue.

Shortcut: You can place OLE objects in Director by dragging a file from the Navigator into the cast window.

Using Paste Special

Use Paste Special to create cast members that display selected portions of OLE (object linking and embedding) objects. This is most useful for showing items like cells in a spreadsheet or particular fields in a database instead of the entire record.

Begin by copying a selected portion of a document to the clipboard in another application that supports OLE. Then choose Paste Special using OLE. The OLE object appears as a cast member and is updated when the source document changes.

- **Using OLE Objects in Director**

- **Film Loop command (Insert menu)**

The Film Loop command changes a selected animation sequence into a repeating loop that appears as a single cast member. A film loop is a score fragment that refers to a group of cast members.

To create a film loop, first select an animated sequence in the score and then choose Create Film Loop from the Insert menu. The sequence can include as many channels as you need. You can also select the sequence in the score and then drag it to the cast.

If you copy a film loop, you need to copy the referenced cast members, too.

- **Modify menu**

Click the name of a menu command for more information:

Cast Properties

Cast Member Properties

Cast Member Script

Sprite Properties

Sprite Script

Frame: Tempo

Frame: Palette

Frame: Transition

Frame: Sound

Frame: Script

Movie Properties

Movie Casts

In-Between

In-Between Special

Font

Paragraph

Borders

Arrange

Align

Tweak

Reverse Sequence

Sort

Cast to Time

Space to Time

Transform Bitmap

Convert to Bitmap

- **Cast Properties command (Modify menu)**

Use Cast Properties to view properties and change settings for the currently selected cast.

Dialog box options

Name displays the name of the current cast for you to view or change.

Storage indicates whether the cast is internal or external and, if the cast is external, where it is stored.

Size displays the size of the cast in kilobytes.

Preload defines how the cast is loaded into memory when the movie runs. The choices are:

- When Needed: The cast does not load into memory until it is required by the movie.
- After Frame One: The cast is loaded when the movie exits frame one.
- Before Frame One: The cast is loaded before the movie plays frame one.

- **Understanding internal and external casts**

Memory management techniques

Memory management techniques

- **Cast Member Properties command (Modify menu)**

Cast Member Properties displays a dialog box containing information about the selected cast member: its name, cast position, type, and its size in kilobytes. The Cast Member Properties dialog box also displays additional information and options for each cast member type. Use the options in the dialog box to define the behavior and appearance of the selected cast members.

The type of information in the Cast Member Properties dialog box depends on the type of cast member.

<u>Bitmap</u>	<u>PICT</u>
<u>Button</u>	<u>Script</u>
<u>Digital Video</u>	<u>Shape</u>
<u>Field</u>	<u>Sound</u>
<u>Film Loop</u>	<u>Text</u>
<u>Director Movie</u>	<u>Transition</u>
<u>Palette</u>	<u>Xtras</u>

Multiple cast members selected



In the cast window, select a cast member and click the Info button as a shortcut for choosing this command. If you are editing the cast member in the paint, text, digital video, or script window, you can click the window's Info button as a shortcut for choosing this command.

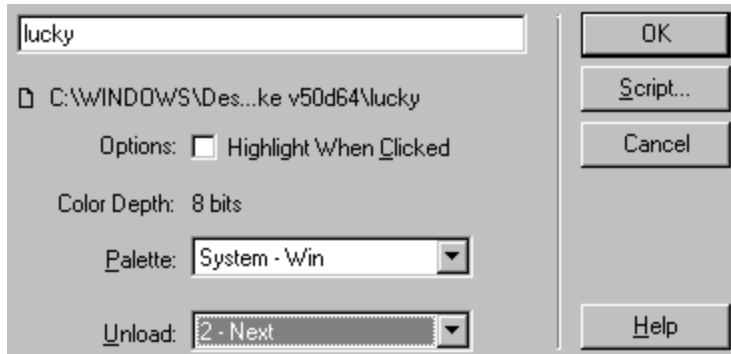
- Select a cast member in the cast or on the stage and then right-click and choose Cast Member Properties from the pop-up.

- **Cast Member Properties: Bitmap (Modify menu)**

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes
- Dimensions

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

File Name displays the location of linked external files. This appears only if the selected cast member is linked to an external file. Click the file name to choose a new location.

Options: Highlight When Clicked makes the current cast member invert when it is clicked by the user. Use this option to create buttons. Even if Highlight When Clicked is checked, the cast member will not do anything unless it is controlled by a Lingo script.

Color Depth displays the color depth of the cast member.

Palette assigns a different palette to the cast member, while maintaining the cast member's original palette references, so the image is not changed. You can change the palette assignment at any time by choosing another palette from the menu.

Unload controls how Director removes the cast member from memory if memory is low.

- 3--Normal: The selected cast member will be removed from memory as necessary.
- 2--Next: The selected cast member will be among the next to be removed from memory.
- 1--Last: The selected cast member will be the last to be removed from memory.
- 0--Never: The selected cast member remains in memory and is never purged.

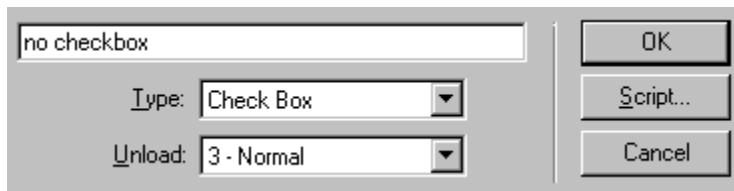
Script opens a script window for the cast member. The script remains attached to the cast member if the cast member is cut or copied and pasted.

- **Cast Member Properties: Button (Modify menu)**

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes.

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

Type lists the button types--push button, check box, or radio button.

Unload controls how Director removes the cast member from memory if memory is low. Choose one of these options from the pop-up menu.

- 3--Normal: The selected cast member will be removed from memory as necessary.
- 2--Next: The selected cast member will be among the next to be removed from memory.
- 1--Last: The selected cast member will be the last to be removed from memory.
- 0--Never: The selected cast member remains in memory and is never purged.

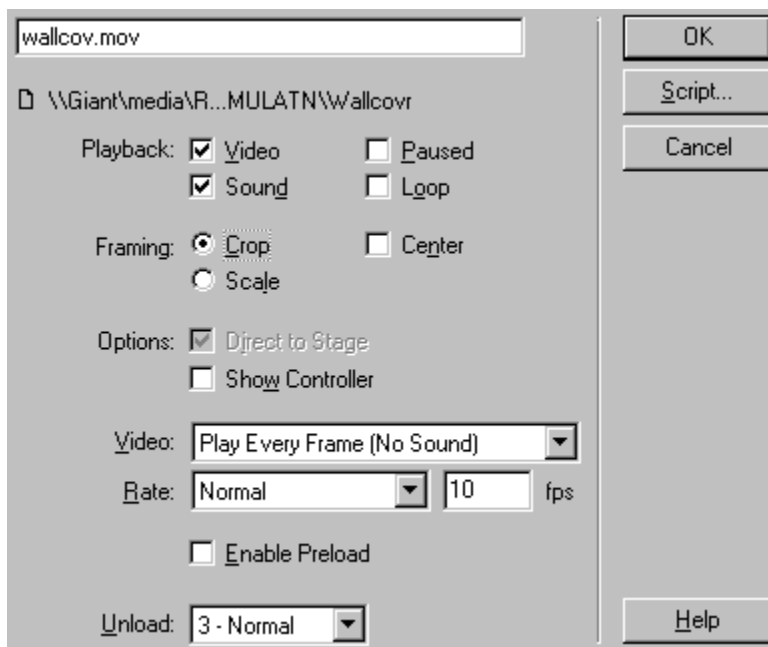
Script opens a script window for the cast member. The script remains attached to the cast member if the cast member is cut or copied and pasted.

- **Cast Member Properties: Digital Video (Modify menu)**

Use the Cast Member Properties for digital video cast members to view information about the current cast member and to change optional settings. The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The length of the movie in seconds
- The size in kilobytes
- Dimensions

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

File Name displays the location of the digital video movie. Click the file name to choose a new location.

Video plays the video portion of the digital video movie. If unchecked, the video portion does not play. Deselect this option and check the Sound if you want to play the audio-only portion of a movie.

Sound plays the sound portion of the digital video movie.

Paused pauses the digital video movie when it first appears on the stage (while playing the Director movie).

By default, a digital video movie starts playing the moment it first appears. If you check Paused, you can later start the movie using the statement:

```
set the movieRate of sprite n to R
```

where:

- n is the sprite number in the current frame
- R is a number representing the rate. For example, 0 = stop, 1 = normal speed, 2 = 2x speed, and -1 = reverse.

- **Lingo elements -- digital video**

Loop loops the digital video movie from the end back to the beginning and continues to play.

Crop retains the movie's original size if you resize the bounding rectangle. The edges of the movie may be clipped.

Center centers the movie when you resize the bounding rectangle. If Center is not checked, the loop maintains its original position when you resize its bounding rectangle. Center is only available if Crop is checked.

Scale scales the movie if you resize the bounding rectangle.

Direct to Stage plays the movie in front of any cast members on the stage, regardless of the channel that contains the movie. Inks are not visible on a movie that plays with this option. In general, use Direct to Stage when you want the best possible performance from a digital video movie and you don't need ink effects or compositing. Results may vary, so you may have to experiment.

Show Controller displays a controller bar below the movie to allow the user to start, stop, and step through the movie. This option is only available if Direct to Stage is checked.

Video defines how the movie is synchronized. If you choose Play Every Frame, every frame of the digital video movie plays. The digital video movie's soundtrack will not play, since the movie can't play the soundtrack asynchronously while the video portion plays frame-by-frame. If you choose Sync to Soundtrack, the movie skips frames as necessary to keep up with the tempo of the soundtrack. These options are only available if Direct to Stage is checked.

Rate determines at what rate the movie plays. The options on the Rate menu are only available if Play Every Frame is checked. The following options appear on the Rate pop-up menu:

- Normal: Each frame plays at its normal rate, and no frames are skipped.
- Maximum: The movie plays as fast as possible while still displaying each frame.
- Fixed: Play the movie using a specific frame rate. Enter the number of frames per second in the field to the right. Use this option only for digital video movies that use the same frame rate for each frame of the movie.

Enable Preload preloads the entire movie (or as much of the movie as can fit into available memory) using the preLoad or preLoadMember Lingo commands. If there is not enough memory to load the entire movie, Director loads only what can fit into memory. If this option is unchecked, Director does not load the movie into memory and instead plays it from disk. This results in slower animation speeds, since each frame must be retrieved from disk before it is played.

Unload controls how Director removes the cast member from memory if memory is low. Choose one of these options from the pop-up menu.

- 3--Normal: The selected cast member will be removed from memory as necessary.
- 2--Next: The selected cast member will be among the next to be removed from memory.
- 1--Last: The selected cast member will be the last to be removed from memory.
- 0--Never: The selected cast member remains in memory and is never purged.

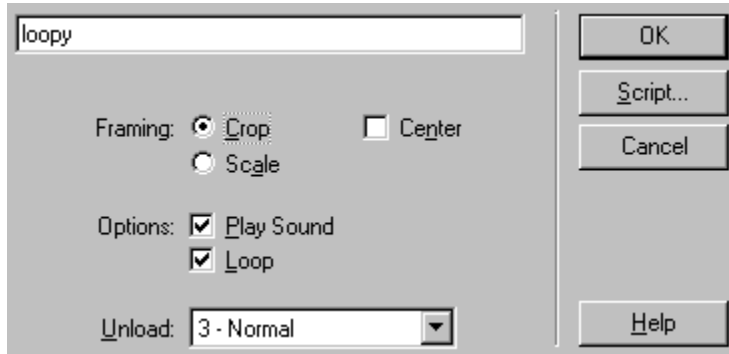
Script opens a script window for the cast. The script remains attached to the cast member if the cast member is cut or copied and pasted.

- **Cast Member Properties: Film Loop (Modify menu)**

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes.

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

Crop retains the film loop's original size if you resize the bounding rectangle of the sprite. The edges of the movie may be clipped.

Scale scales the film loop if you resize the sprite bounding rectangle.

Center centers the film loop when you resize the bounding rectangle. If Center is not checked, the loop maintains its original position when you resize its bounding rectangle. Center is only available if Crop is checked.

Play Sound enables sound during playback. If not checked, sound is disabled during playback.

Loop returns the animation from the last frame back to the first and continues to play. If this option is not checked, the animation doesn't loop, and the last frame remains on the stage.

Unload controls how Director removes the cast member from memory if memory is low. Choose one of these options from the pop-up menu.

- 3--Normal: The selected cast member will be removed from memory as necessary.
- 2--Next: The selected cast member will be among the next to be removed from memory.
- 1--Last: The selected cast member will be the last to be removed from memory.
- 0--Never: The selected cast member remains in memory and is never purged.

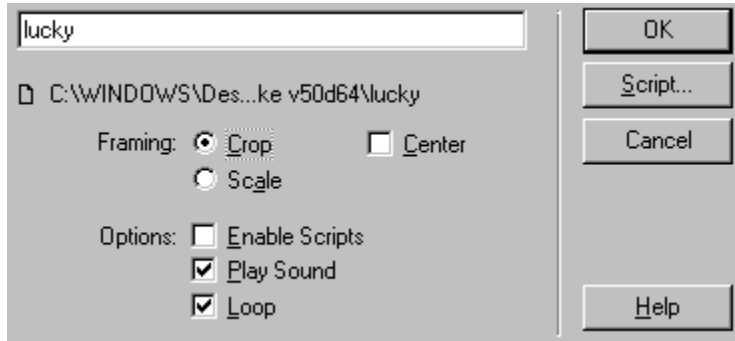
Script opens a script window for the cast member. The script remains attached to the cast member if the cast member is cut or copied and pasted.

- **Cast Member Properties: Linked Director Movie (Modify menu)**

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes.

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

File Name displays the location of the external file associated with the linked movie. Clicking the file name lets you choose a new location.

Crop retains the linked Director movie's original size if you resize the bounding rectangle. The edges of the movie may be clipped.

Scale scales the movie if you resize the bounding rectangle.

Center centers the linked Director movie when you resize the bounding rectangle. If Center is not checked, the movie maintains its original position when you resize its bounding rectangle. Center is only available if Crop is checked.

Enable Scripts activates the linked movie's scripts when the movie is used in the score. If this option is not checked, Director ignores the movie's scripts.

Play Sound enables sound during playback. If not checked, sound is disabled during playback.

Loop returns the movie from the last frame back to the beginning and continues to play. If this option is not checked, the movie doesn't loop, and the last frame remains on the stage when the movie finishes playing.

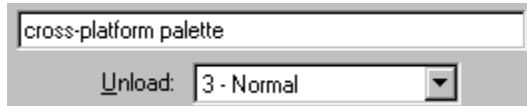
Script opens a script window for the cast member. The script remains attached to the cast member if the cast member is cut or copied and pasted.

- **Cast Member Properties: Palette (Modify menu)**

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

Unload controls how Director removes the cast member from memory if memory is low. Choose one of these options from the pop-up menu.

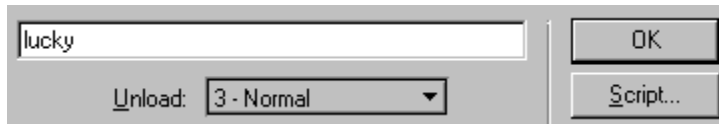
- 3--Normal: The selected cast member will be removed from memory as necessary.
- 2--Next: The selected cast member will be among the next to be removed from memory.
- 1--Last: The selected cast member will be the last to be removed from memory.
- 0--Never: The selected cast member remains in memory and is never purged.

- **Cast Member Properties: PICT (Modify menu)**

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes.
- Dimensions

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

Unload controls how Director removes the cast member from memory if memory is low. Choose one of these options from the pop-up menu.

- 3--Normal: The selected cast member will be removed from memory as necessary.
- 2--Next: The selected cast member will be among the next to be removed from memory.
- 1--Last: The selected cast member will be the last to be removed from memory.
- 0--Never: The selected cast member remains in memory and is never purged.

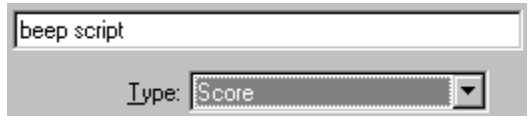
Script opens a script window for the cast member. The script remains attached to the cast member if the cast member is cut or copied and pasted.

- **Cast Member Properties: Script (Modify menu)**

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes.

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

Type displays the script type (movie, score, or parent) for the selected cast member, and lets you change it. A movie script's handlers are global, and can be called from other scripts. A score script's handlers are local, and cannot be called from other scripts. If you change a movie script into a score script, it appears in the Script pop-up menu in the score.

- **Cast Member Properties: Shape (Modify menu)**

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes.

Click a dialog box option for more information:



The dialog box contains the following elements:

- Text field: (no name)
- Shape: Oval (dropdown menu)
- Options: Filled
- Buttons: OK, Script..., Cancel

Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

Shape displays the current shape--rectangle, round rectangle, or oval--and lets you change the shape into any of the other available shapes.

Filled fills the currently selected shape with the current fill pattern and colors as specified in the **tool palette**.

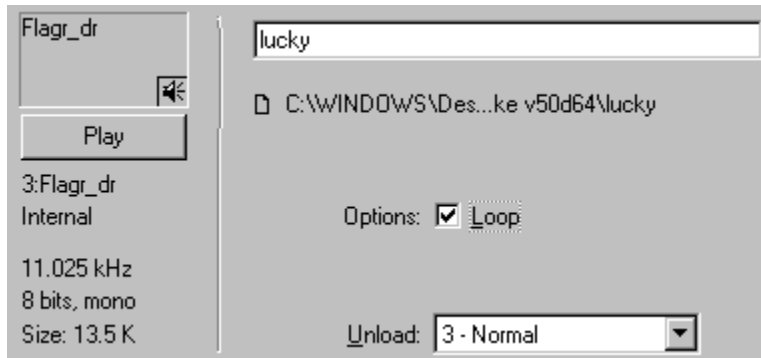
Script opens a script window for the cast member. The script remains attached to the cast member if the cast member is cut or copied and pasted.

- **Cast Member Properties: Sound (Modify menu)**

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes
- Sample rate, sample size and channels

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

File Name displays the location of linked external sound files. This appears only if the selected cast member is linked to an external file. Click the file name to choose a new location.

Loop makes the sound play continuously. If not checked, the sound plays once, even if the movie loops.

Unload controls how Director removes the cast member from memory if memory is low.

- 3--Normal: The selected cast member will be removed from memory as necessary.
- 2--Next: The selected cast member will be among the next to be removed from memory.
- 1--Last: The selected cast member will be the last to be removed from memory.
- 0--Never: The selected cast member remains in memory and is never purged.

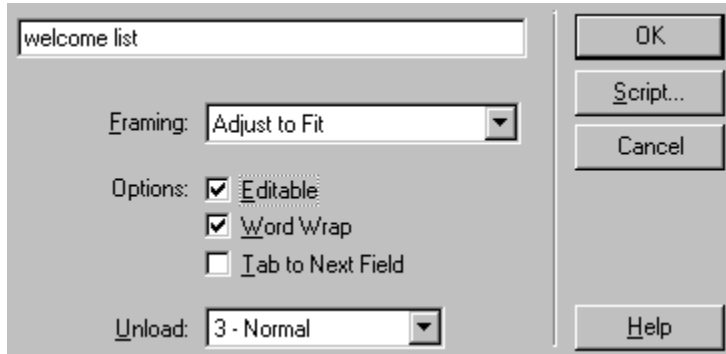
Play plays the sound at its pre-recorded sampling rate.

- **Cast Member Properties: Field (Modify menu)**

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes.

Click a dialog box option for more information:



Name field displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

Framing displays framing options for the current field.

- Adjust to Fit causes the field to expand vertically when text is entered that extends beyond the current size of the box.
- Scrolling attaches a scroll bar to the right side of the field. This is useful for a large amount of text.
- Fixed causes the box to retain its original size. If text is entered that extends beyond the limits of the box, the text is not displayed.
- Limit to Field Size sets the field's width to be fixed to the size of the field. Characters that don't fit are ignored.

Editable makes field cast members editable during movie playback. You can use this option instead of using the Lingo command set the editable of sprite to TRUE.

If you set a field cast member to be editable in the cast, it is always editable. Director overrides the score's **editable checkbox** setting for the sprite.

Word Wrap makes words move to the next line when they reach the edge of the box. If unchecked, text that extends beyond the right edge is truncated and you must use the Enter key to generate a new line.

Tab to Next Field causes the Tab key to advance the cursor to the next editable field on the stage during playback. Note that the editable checkbox must be checked, or the Lingo command set the editable of sprite to TRUE must be specified for this option to have any effect.

Unload controls how Director removes the cast member from memory if memory is low. Choose one of these options from the pop-up menu.

- 3--Normal: The selected cast member will be removed from memory as necessary.
- 2--Next: The selected cast member will be among the next to be removed from memory.
- 1--Last: The selected cast member will be the last to be removed from memory.
- 0--Never: The selected cast member remains in memory and is never purged.

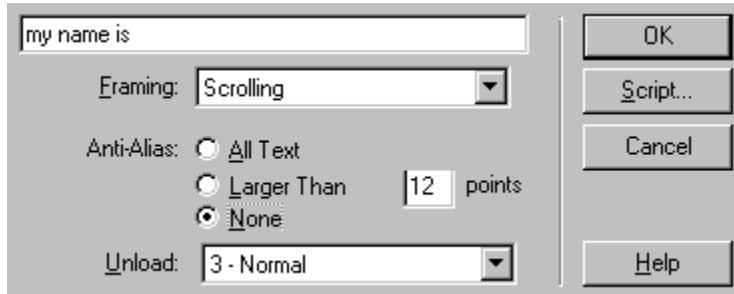
Script opens a script window for the cast member. The script remains attached to the cast member if the cast member is cut or copied and pasted.

- **Cast Member Properties: Text (Modify menu)**

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes.

Click a dialog box option for more information:



Name displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

Framing displays framing options for the current text cast member.

- Adjust to Fit causes the text box to expand vertically when text is entered that extends beyond the current size of the box.
- Scrolling attaches a scroll bar to the right side of the text box. This is useful for a large amount of text.
- Cropped causes the text box to retain its original size. If you enter text that extends beyond the limits of the box, the text is not displayed.

Anti-Alias Text: dramatically improves the appearance of large text, but it can blur or distort smaller text. Experiment with the size setting to get the best results for the font you are using.

- All Text anti-aliases all the text in the text block.
- Larger Than anti-aliases text larger than the point size entered in the points field.
- None--no text is anti-aliased in the current text block.

Unload controls how Director removes the cast member from memory if memory is low.

- 3--Normal: The selected cast member will be removed from memory as necessary.
- 2--Next: The selected cast member will be among the next to be removed from memory.
- 1--Last: The selected cast member will be the last to be removed from memory.
- 0--Never: The selected cast member remains in memory and is never purged.

Script opens a script window for the cast member. The script remains attached to the cast member if the cast member is cut or copied and pasted.

- **Cast Member Properties: Transition (Modify menu)**

The following information about the current cast member appears on the left side of the dialog box:

- The cast member number
- The cast member name
- The cast name
- The size in kilobytes.

Click a dialog box option for more information:



Name displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

Unload controls how Director removes the cast member from memory if memory is low. Choose one of these options from the pop-up menu:

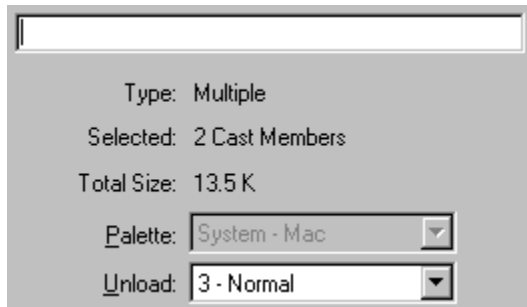
- 3--Normal: The selected cast member will be removed from memory as necessary.
- 2--Next: The selected cast member will be among the next to be removed from memory.
- 1--Last: The selected cast member will be the last to be removed from memory.
- 0--Never: The selected cast member remains in memory and is never purged.

Options button is only available if you have installed Xtra transitions in the Xtras folder (within the Director application folder). The contents of the Options dialog box is determined by the developer of the Xtra. Refer to any documentation supplied with the Xtra.

- **Cast Member Properties: Multiple Items (Modify menu)**

Use Cast Member Properties to view and change settings for several selected cast members at once.

Click a dialog box option for more information:



Since multiple cast members are selected, no cast member name is shown.

Type displays "Multiple" when several cast members are selected, unless all the selected cast members are the same type.

Selected displays the number of cast members in the selection.

Total Size displays the total size, in kilobytes, of the selected cast members.

Palette lets you choose the palette used by the selected cast members.

Unload controls how Director removes the cast members from memory if memory is low.

Choose one of these options from the pop-up menu:

- 3--Normal: The selected cast members will be removed from memory after all purge priority 3 cast members have been purged.
- 2--Next: The selected cast members will be among the next to be removed from memory.
- 1--Last: The selected cast members will be the last to be removed from memory.
- 0--Never: The selected cast members remain in memory and is never purged.

- **Cast Member Properties: Xtras (Modify menu)**

Xtras are cast members created as plug-in extensions to Director. They can be new media types or add-on transitions. Xtra cast members may have additional settings accessible through an Options button.

Click a dialog box option for more information:



- **Installing Xtras.**

Name displays the name of the current cast member for you to view or change. The name remains attached to the cast member if it is moved to a new position in the cast.

Director does not prevent you from creating duplicate cast member names, but you should avoid using them. If more than one cast member has the same name, Lingo uses the cast member with the lowest number in the cast. Use cast member names instead of numbers to address cast members in a Lingo script, so that you don't have to update your scripts if your cast members are renumbered or sorted.

Unload controls how Director removes the cast member from memory if memory is low. Choose one of these options from the pop-up menu:

- 3--Normal: The selected cast member will be removed from memory as necessary.
- 2--Next: The selected cast member will be among the next to be removed from memory.
- 1--Last: The selected cast member will be the last to be removed from memory.
- 0--Never: The selected cast member remains in memory and is never purged.

Options opens a dialog box containing special controls and options for the Xtra cast member. Not all Xtra cast member have options available. The contents of the Options dialog box is determined by the developer of the Xtra. Refer to any documentation supplied with the Xtra.

- **Cast Member Script command (Modify menu)**

Choose this command to open the script associated with the selected cast member. This is the same as clicking the Script button in the Cast Member Properties dialog box or in the cast window.

Shortcuts: To open the cast member script:

- Press Control-' (Control-hyphen).
- In the cast window, select a cast member and click the Script button.
- In the media editor window, click the window's Script button.

- **Sprite Properties command (Modify menu)**

Use the Sprite Properties command to change the size, location, and blend percentage of a selected sprite. If you select more than one sprite, this command lets you edit them as a group.

Click a dialog box option for more information:



The image shows a dialog box for editing sprite properties. It has a light gray background and contains several input fields and a checkbox. The fields are arranged as follows:

- Size:** A row of three input fields. The first is labeled "Width" and contains the value "26". The second is labeled "Height" and contains the value "19". The third is labeled "Scale" and contains the value "100". There is a small "X" symbol between the width and height fields, and a "%" symbol after the scale field.
- Maintain Proportions:** A checkbox with a checkmark inside, followed by the text "Maintain Proportions".
- Location:** A row of two input fields. The first is labeled "Left" and contains the value "533". The second is labeled "Top" and contains the value "212".
- Blend:** An input field labeled "Blend" containing the value "100", followed by a "%" symbol.

- The size of a sprite on the stage is controlled by its parent cast member unless you change the sprite's size using these options, or by manually resizing the sprite on the stage. Thereafter, any changes in the size of the parent cast member do not affect the size of the sprite on the stage.

- **In-Between Special**

- Size** changes the sprite's size. Enter an exact size or a scaling percentage.
- Height, Width: Enter a specific width and height for the sprite, in pixels.
 - Scale: Enter a percentage in the Scale field. The sprite is scaled relative to its current size, not to the size of its parent cast member.
 - Maintain Proportions: Check this box to maintain the same proportions of width and height when the object is resized.

Location changes the top left corner position of the sprite on the stage.

- Left: Enter the number of pixels you wish to offset the sprite from the left edge of the stage.
- Top: Enter the number of pixels you wish to offset the sprite from the top edge of the stage.

Blend specifies the blend percentage for selected sprites in the score. You can apply a blend effect to sprites that use the blend, background transparent, mask, or matte ink. Each sprite in the same frame can store its own blend value.

Use the Blend option in the In Between Special dialog box to fade sprites in or out of the stage as they are animating.

- **Sprite Script command (Modify menu)**

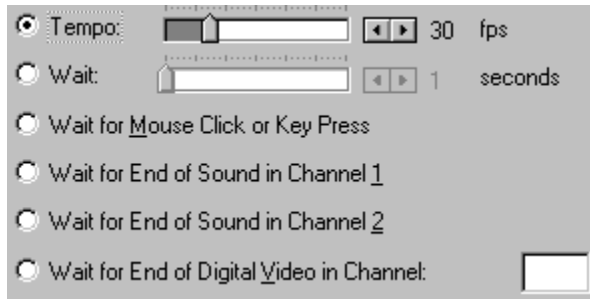
Sprite Script opens a script window for the currently selected sprite.

- **Script window**

- **Frame Tempo command (Modify menu)**

Use this command to set a tempo or pause in your movie. The tempo setting determines how the playback head moves from frame to frame.

Click a dialog box option for more information:



When you set a tempo, it also applies to all frames to the right of the tempo setting, until another tempo is encountered in the tempo channel.

Shortcuts: To open the Frame Tempo dialog box:

- Double-click a cell in the tempo channel.
- Right-click a cell in the tempo channel and choose Tempo.

- If you place tempo settings in the same frame as a transition, some tempo channel settings such as wait and sound playing become disabled. To avoid this, don't place a transition in the same frame as your tempo settings. Instead, place the tempo settings in the frame immediately before or after the transition.

Tempo sets a new tempo for the movie. Use the arrows or slide the box to change the settings. This is the same as changing the tempo in the **control panel**.

Wait stops the movie at the current frame for the time specified. Use the arrows or slide the box to change the settings.

Wait for Mouse Click or Key Press pauses the playback head until the user clicks the mouse or presses a key. The cursor changes to a blinking mouse to indicate that the movie is paused.

You might want to include some text on the screen telling users to click the mouse or press a key to continue.

Wait For End of Sound in Channel 1 pauses the playback head until a sound in sound channel 1 finishes playing.

Wait For End of Sound in Channel 2 pauses the playback head until a sound in sound channel 2 finishes playing.

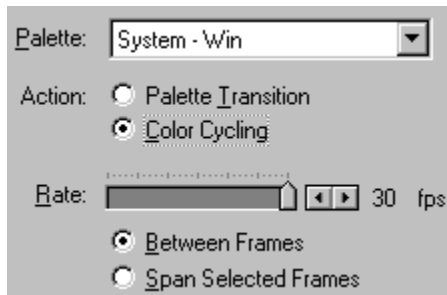
Wait for End of Digital Video in Channel pauses the playback head until a digital video finishes playing. If you have several digital videos, make sure you specify the channel of the longest playing digital video when using this option.

- **Frame Palette command (Modify menu)**

Use the Frame Palette command to change palettes in a selected frame of the palette channel. When you set a palette in the palette channel of the score, the color of the cast members in the movie is determined by that palette until another palette is encountered in the palette channel.

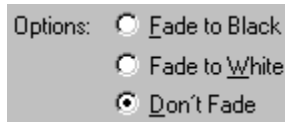
When you specify a new palette in the palette channel, cast members change color depending on the position of their colors in the palette. For example, if one of your cast members is yellow, and yellow occupies the fifth color in your palette, the cast member will become whatever color is in the fifth position in the new palette.

Click a dialog box option for more information:

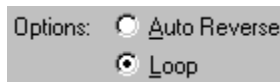


The options in the dialog box change depending on whether you choose Palette Transition or Color Cycling.

When Palette Transition is selected, these options appear in the dialog box (click for more information):



When Color Cycling is selected, these options appear in the dialog box (click for more information):



Shortcuts: To open the Frame Palette dialog box:

- Double-click a frame in the palette channel.
- Right-click a frame in the palette channel and choose Palette.

- **Understanding color depth**
Understanding color palettes

Palette lets you choose the palette used for the selected cells in the palette channel.

Palette Transition changes the palette at the current frame, or the beginning of the current selected range. Choose Palette Transition when you want a smooth transition from one palette to another. Instead of your cast member abruptly changing colors when you switch palettes, this option gradually blends from one palette to the next.

Color Cycling changes the palette by rotating the colors in a selected range of the palette. For example, if a cast member's color is the fifth color in the palette, and you select a range of colors from four to six, the cast member changes colors when the movie is played, cycling through colors four, five, and six.

To select a range to cycle, drag across the colors to be cycled in the dialog box. You can also click a color, and Shift-click another color to select those colors and all the colors in between.

Rate displays the rate at which the palette changes between frames. Use the arrow keys or slide the box to change the setting. This setting does not apply if you select Span Selected Frames.

- Between Frames changes the palette between frames. The Rate setting determines the length of the transition. All animated movement will halt as the transition takes place. This option appears only if you first select a range of frames in the palette channel.
- Span Selected Frames changes the palette while the selected frames are playing. The number of frames selected determines the length of the transition. Any movement in those frames will take place as usual. This option appears only if you first select a range of frames in the palette channel.

Palette Transition options

Fade to Black fades the entire screen to black. Like other palette transitions, this can occur over time, or in between frames. A nice way to end a movie is to make the final frame a black cast member that covers the whole stage, and then fade to black over several frames before the last frame.

Fade to White fades the entire screen to white. Like other palette transitions, this can occur over time, or in between frames. A nice way to end a movie is to make the final frame a white cast member that covers the whole stage, and then fade to white over several frames before the last frame.

Don't Fade changes the palette without fading the screen during the palette transition.

Note: Palette transitions including Fade to Black and Fade to White do not work in 16-, 24-, and 32-bit environments. These features require either that palettes be in use or that a video card set to 256 colors is present.

Color Cycling options

Cycles specifies the number of cycles per frame.

Auto Reverse reverses the direction of color cycling when the cycle completes.

Loop returns the colors cycle to the beginning when it reaches the end.

● **Frame Transition command (Modify menu)**

Use this command to define and select a transition. To set a transition, select a cell in the transition channel of the score window, and choose Frame Transition from the Modify menu. The transition occurs when the playback head reaches that cell.

Different options are available for different transitions.

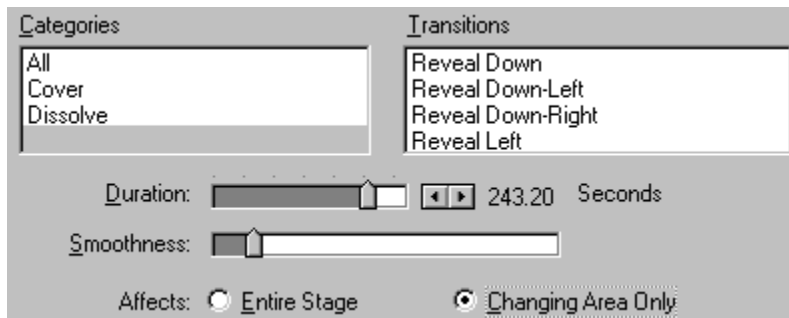
Transition Xtras

You can add new transitions to Director as Xtras. Xtra transitions appear with special icons in the Frame Properties: Transitions dialog box. Install Xtra transitions by placing them in the Xtras folder in the Director application. For more information, see [Installing Xtras](#).

Once they are defined, transitions appear in the cast window as cast members. You can place them in the transition channel by dragging them from the cast to the score.

Xtra transitions may offer extra options provided by the developer. If the Options button is available when you choose an Xtra transition, click it to view and change the transition options.

Click a dialog box option for more information:



Shortcuts: To open the Frame Transition dialog box:

- Double-click a cell in the transition channel.
- Right-click a cell in the transition channel and choose Transition.

●

● To play a sound while a transition takes place, place the sound in the frame immediately before the transition.

● If you are developing a movie for Macintosh and Windows computers, don't use the Dissolve Pixels, Dissolve Pixels Fast, or Dissolve Patterns transitions. These transitions produce entirely different results on a Windows computer.

● If you export a movie that contains transitions as a digital video or PICS file, the transitions will not be included.

Categories lists the categories of transitions. If you select a category, such as dissolve, then only the dissolve transitions are listed in the transitions list. If you select the All category, then all available transitions are listed.

Transitions lists the available transitions.

Duration indicates the approximate amount of time (in seconds) of the entire transition. Adjust the slider to change the setting.

Smoothness selects the smoothness of the transition. Adjust the slider to change the degree of smoothness

Affects indicates where the transition place on stage.

- Entire Stage makes the transition take place over the entire stage area.
- Changing Area Only makes the transition takes place over the changing area of the frame.

- **Frame Sound command (Modify menu)**

Use this command to select and preview sounds. Select one or more cells in one of the sound channels, and then choose this command. If you have not made a selection in one of the sound channels in the score, this command is not available.

The dialog box lists all sounds in the currently opened casts. Select a sound from the list and click OK to place the sound in the selected frames.

To play a sound, select one from the list and click Play.

Shortcuts: To open the Frame Sound dialog box:

- Double-click a frame in the sound channel.
- Right-click a frame in the sound channel and choose Sound.

- **Frame Script command (Modify menu)**

Choose this command to open the script for the current frame.

Shortcut: To open the script for the current frame:

- Double-click a frame in the script channel.
- Right-click a frame in any channel and choose Frame Script from the shortcut menu.

- **Movie Properties command (Modify menu)**

Control-Shift-D

The Movie Properties command lets you specify options such as stage size and color for the currently open movie.

Click a dialog box option for more information:

The screenshot shows the 'Movie Properties' dialog box with the following settings:

- Stage Size:** Current screen (dropdown), Width: 640, Height: 480
- Stage Location:** Upper Left (dropdown), Left: 0, Top: 0
- Default Palette:** System - Win (dropdown), Stage Color: Yellow (color swatch)
- Options:**
 - Lock Frame Durations
 - Pause When Window Inactive
 - Remap Palettes When Needed
 - Allow Outdated Lingo
- Buttons:** Save Font Map..., Load Font Map...

- **Projector Options dialog box**
Understanding Font Mapping

Stage Size defines the size of the stage. Changing the stage size is useful if you want to display movies on a smaller or larger stage, or if you want to change the stage size to match the size of a digital video movie. Change the size of the stage by choosing a setting from the menu, or by entering the width and height of the stage.

If you choose a setting from the pop-up menu, the values in the Width and Height fields automatically update.

Stage Location changes the location of the stage.

- Centered places the stage window in the center of your monitor. This option is useful if you play a movie that was created for a 13-inch screen on a larger screen. You can also use this option if you are creating a movie on a larger screen that will be seen on smaller screens.
- Upper Left places the stage in the top left corner. Alternatively, the values you type in the Left and Top boxes represent the number of pixels the stage is moved from the top left corner of the screen. These values apply only if the stage is smaller than the current monitor's screen size.

Default Palette defines the palette Director uses for the movie until it encounters a different palette setting in the palette channel.

Stage color determines the color of the stage. Click to select a new background color from the current palette.

Lock Frame Durations locks the current playback rate so that Director plays the movie at the same speed on all types of computers. For frames without recorded durations, Director uses the current tempo.

To unlock the playback rate, remove the check from this option or make an editing change to your movie, such as adjusting the tempo, or adding or deleting a frame in your movie. Unlocking the playback rate lets you record frame durations as you play the movie.

Locked movies will not play faster when played on a faster computer, but may play slower on a slower computer.

Pause When Window Inactive specifies when movies in windows play. When this option is set, a movie in a window only plays when the main movie is playing or it is the frontmost window; otherwise the movie in a window continues to animate.

Remap Palettes When Needed remaps the current palette when cast members with different palettes appear on the stage. If this box is checked, Director automatically creates a common palette and remaps all images on the stage that have a different palette to the common palette. The cast members themselves are not modified. The common palette determines how the cast member is remapped. For example, if a cast member uses a grayscale palette, it will be drawn on the stage using whatever grays are available in the common palette.

Allow Outdated Lingo lets you include Lingo commands used by Director 4.0 that are no longer acceptable.

Save Font Map saves the current font map settings in a text file. You can then edit this file to change the mapping. The font mapping table tells Director which Macintosh fonts to use in place of which Windows fonts--and vice versa--when a text cast member created on one platform is displayed on the other. The font map table consists of a text file that contains the font mapping information.

When you create a new movie, Director looks for a file called FONTMAP.TXT in the same folder as the Director application. This file specifies how Director maps fonts between the Macintosh and Windows platforms. If Director finds this file, it uses it to create an internal font map for the movie. If no FONTMAP.TXT file exists, the new movie uses no font map.

When you open the movie on the PC, Director uses the movie's internal font map to determine the appropriate substitute Windows fonts for text cast members that were created on a Macintosh. If the movie has no font map, Director substitutes other available fonts.

Load Font Map loads the font mapping assignments specified in the chosen font map file.

- **Movie Casts command (Modify menu)**

Control-Shift-C

Use Movie Casts to view the casts in the current movie, create new casts, and link external casts to the movie. The Movie Casts dialog box displays a list of all the casts in the current movie, including internal casts and linked external casts.

Dialog box options

New opens the **New Cast** dialog box so that you can create a new cast.

Link attaches existing external casts to the movie. Use the dialog box that appears to select an external cast file to link to the movie.

Remove unlinks or deletes a cast from the movie. Select a cast from the list and click Remove. If the cast is internal, it is deleted. If it is external, it is unlinked from the movie. You cannot delete the first internal cast in the movie.

Properties opens the **Cast Properties** dialog box for the cast selected on the list.

- **Understanding internal and external casts**

- **Font command (Modify menu)**

Ctrl+Shift+T

Use the Font dialog box to specify all the formatting options for characters and lines of text. Not all options are available for fields.

To change character formatting, first select the text you want to change, and then choose Font from the Modify menu. If you select the cast member, all the text changes.

Dialog box options

Font lists all the fonts installed in your system. Select a new font by clicking one on the list. Director may not be able to anti-alias certain fonts in your system (such as bitmap fonts and some variations of TrueType fonts). When you select a font that can't be anti-aliased, the message "This font cannot be anti-aliased" appears above the font preview.

Style: Bold, Italic, and Underline each apply the character attribute to the selected text.

Size controls the font size, in points.

Color determines the color of the selected text. Click to choose a new color from the current color palette.

Line Spacing sets the total height for all lines in the paragraph in points. This option is not available for fields.

Kerning increases or decreases the amount of space (in points) between selected characters. This option is not available for fields.

- **Understanding text and fields**

Cast Member Properties: Text

Text Inspector

Properties: Field

- **Paragraph command (Modify menu)**

Alt+Control+ShiftT

Use the Paragraph dialog box to view and change paragraph formatting. The Paragraph command is only available for text cast members.

Dialog box options

Alignment determines how the selected paragraph is aligned with the text box. The choices are Left, Right, Center, and Justify. (The Justify setting aligns text to both the left and right margins.)

Margin:

- Left and Right defines the left and right margin settings of the text box. Click the arrows or enter a number to change the amount the paragraph is indented.
- First Indent controls the indent setting of the first line of text in the paragraph. Changing the setting is the same as moving the margin markers on the text ruler.

Spacing increases or decreases the amount of space before or after the current paragraph. Click the arrows or enter a value in points.

- **Text ruler**
Understanding text and fields

• **Borders submenu (Modify menu)**

The commands on the Borders submenu only apply to fields. You cannot use them to change text cast members.

Line adds a box around the field on the stage. Choose the line thickness from the submenu.

Margin changes the distance between the edges of the field and the characters inside. Choose the margin width from the submenu.

Box Shadow adds a drop shadow to the text box. Choose the drop shadow width from the submenu.

Text Shadow adds a drop shadow to paint window text or field text. Choose the shadow width from the submenu. Drop shadow on text is a good way to ensure that your text will remain legible in color if you are planning to overlay text to videotape.

- **In-Between command (Modify menu)**

Control-B

Use In-Between when you want a sprite to move in a straight line across the stage, grow or shrink smoothly, or remain stationary for a number of frames. Director fills in the selected cells in the score with the sprite's incremental motion between frames.

In-Between interpolates the incremental positions of a sprite between key frames. Key frames are the starting and ending point for the sprite's motion. If the starting and ending points are the same, the sprite remains stationary. If the starting and ending points are different, In-Between generates new sprites in the right locations between the key frames. In-Between also works with sprites that are stretched or squeezed.

In-Between is also a quick way to fill in several frames with the same sprites. You can use In-Between on several channels at once.

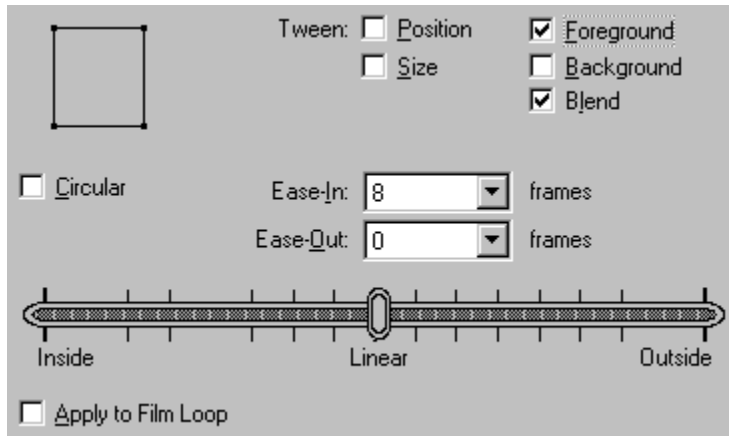
- **In-Between Special**

• **In-Between Special command (Modify menu)**

Control-Shift-B

Use In-Between Special when you want a sprite to move in a curved path, to accelerate or decelerate across the stage, or to change color or blend.

Click a dialog box option for more information:



To make your sprite follow a curved path, you must set the sprite in at least three positions on the stage in different frames. As with In-Between, select all the cells between the first and last position of the sprite, and then choose this command.

Tween specifies the parameters to be modified. These include Location, Size, Foreground Color, Background Color, and Blend. Check the parameters that you want to in-between.

Ease-In and Ease-Out add realism to your animations by gradually accelerating sprites to full speed or decreasing their speed as they come to a halt.

- Ease-In selects the number of frames over which you want to accelerate the sprite. If you choose to accelerate a sprite over the first eight frames, for example, a stationary sprite will ramp up to full speed over that many frames. To enter a number other than the choices in the pop-up menu, choose Other.
- Ease-Out selects the number of frames over which you want to decelerate the sprite. For example, decelerating a sprite over eight frames will gradually bring the sprite to a halt by the eighth frame. To enter a number other than the choices in the pop-up menu, choose Other.

Circular only affects sprites that begin and end in the same point. If checked, the sprite will circle around the stage, but it won't pass through the starting point. The effect is to make a rounder circle. If this option is not checked, the sprite passes through the starting point during the animation.

The **Inside/Outside slider** controls the degree to which the sprite's curved path follows the inside or outside boundaries of the path. If you drag the slider to the left, the sprite follows a curved path that is inside the sprite positions you set before choosing In-Between Special. If you move the slider to Linear, the sprite will travel in straight lines between the points you set earlier. Dragging the slider all the way to the right causes the sprite to pass through the points you set as it travels a curved path.

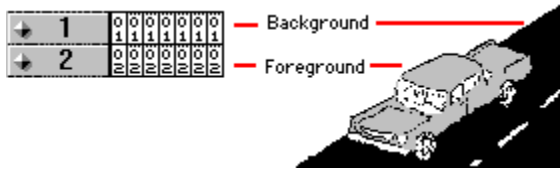
Inside, Linear, and Outside define the path the sprite follows when it is in-betweened. If the beginning and ending points of the sprite are the same, the diagram in the dialog box will be circular, indicating that the sprite will travel in a circle when in-betweened. If the beginning and end points are not the same, the diagram describes a curved path, indicating that the sprite ends in a position different than the starting point. This diagram does not show the actual path of the sprite, just the type of curve it will follow.

Apply to Film Loop applies the current in-between settings to a film loop. For example, if you have a sequence of six sprites that make a walking person, you can store all six as a single sprite, called a film loop. Then, the film loop can be in-betweened as if it were a single sprite. Be sure that a film loop is selected in the cast window before choosing In-Between Special.

- **Cast Member Properties: Film Loop**

● **Arrange submenu (Modify menu)**

The Arrange submenu contains four commands that move sprites up or down in the score, changing their order on the stage. Sprites appear on the stage in order, starting with the first channel. A sprite in channel two appears on top of a sprite in channel one.



Bring to Front (Control+Shift+Up arrow) moves the selected cells to the last channels in the score. The sprites in those cells move in front of all other sprites.

Move Forward (Control+Up arrow) switches the selected cells with the cells immediately below. This command is the same as clicking the Move Forward button at the bottom of the score window.

Move Backward (Control+Alt+Down arrow) switches the selected cells with the cells immediately above. This command is the same as clicking the Move backward command at the bottom the score window.

Send to Back (Control+Alt+Shift+Down arrow) moves the selected cells to the first channels in the score. The sprites in those cells move behind all other sprites.

- **Align command (Modify menu)Control-K**

Use the align palette to align sprites on the stage. You can align sprites in multiple channels and frames. Align is especially useful for making sure that a sprite does not move from frame to frame.

To align sprites:

1. Select the sprites you want to align on the stage or in the score.
You can select sprites in as many different frames or channels as you need. All of the sprites will be aligned to the last sprite you select.
2. Choose Align from the Modify menu.
The vertical and horizontal alignment options appear on two pop-up menus.
3. Select the options you want and click Align to align the selected sprites.
The align palette stays open until you close it.

Dialog box options

Horizontal alignment options include No Change, Align Lefts, Align Centers, Align Rights, Align Reg. Points.

Vertical alignment options include No Change, Align Tops, Align Centers, Align Bottoms, Align Reg. Points.

You can also click the preview to experiment with sprite alignment.

- You can also click the preview to experiment with sprite alignment.
- **Using registration points**

● **Tweak (Modify menu)**

Control-Shift-K

Use the tweak window to move one or more selected sprites in any direction with precision. Drag the point on the left side of the window, or enter the number of pixels in the fields for horizontal and vertical change and click the tweak button.

Continue clicking the Tweak button to repeatedly move the selected sprites the same distance.

- **Reverse Sequence command (Modify menu)**

Reverses the order of selected cells in the score.

● **Sort (Modify menu)**

Use Sort to rearrange selected cast members in the cast and eliminate empty cast member positions. To rearrange an entire cast, first choose Select All from the Edit menu before choosing this command.

Director automatically updates the score with the new number for each repositioned cast member.

Note: Because cast member numbers may change when you use this command, cast member number references in scripts may become invalid. If you use Sort Cast Members, you may have to go through your scripts to update them with the new numbers. Use cast member names instead of numbers to address cast members in a Lingo script so that you don't have to worry if your cast members get re-numbered.

Dialog box options

Usage in Score sorts selected cast members in the order in which they appear in the score. If a cast member does not appear in the score it is placed after all the cast members that are referenced from the score.

Media Type sorts selected cast members by type (bitmap, palette, button, text, sound, shape, PICT, digital video, film loop, movie, script, field, Xtra, transition, OLE).

Name sorts selected cast members alphabetically by name.

Size sorts selected cast members by file size, in decreasing size order.

Empty at End places empty cast members at the end.

- **Cast to Time command (Modify menu)**

Use Cast to Time to speed the creation of a cast member sequence in your movie. This command places selected cast members sequentially into separate frames in the score.

If you select a single cell in the score before choosing this command, the selected cast members are added to the score beginning at the selected cell. Any existing score data is replaced by the Cast to Time sequence. If you set an insertion point in the score before choosing this command, the Cast to Time sequence is inserted in channel 1, beginning at the insertion point. If you select a range of score cells before choosing this command, the Cast to Time sequence that is inserted will only be as long as the number of selected cells in the score.

Shortcut: You can also place selected cast members across time in the score by Alt dragging from the cast.

- **Recording with Cast to Time**

- **Space to Time command (Modify menu)**

Space to Time moves selected sprites in one frame to a single channel in the score so they play in a sequence of frames.

The dialog box lets you specify the number of frames apart to spread sprites. Consecutive cells (1 frame apart) is the default.

For example, if you are animating a bouncing ball, it's difficult to lay out each position of the ball without comparing it to previous positions. With Space to Time, you can drag the ball from the cast window repeatedly to lay out your sequence, select the cells in the score that contain the sprites you just positioned on the stage, then choose Space to Time, and all the sprites shift from their vertical positions in one frame to horizontal positions in one channel. You might also find this a useful way to lay out sprites before using **In-Between Special**.

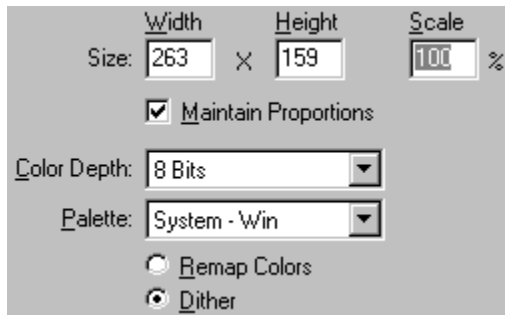
- **Recording with Space to Time**

- **Transform Bitmap command (Modify menu)**

Transform Bitmap changes the size, color depth, and palette of selected cast members. Any change you make to a cast member's color depth or palette affects the cast member itself—not just its appearance on the stage. As a result, color depth and palette changes can't be undone. If you want to keep a cast member's original bitmap unchanged but temporarily apply a different palette, use **Cast Member Properties** instead. To change the size of only the sprite on the stage, use **Sprite Properties**.

The Transform Bitmap dialog box displays values for the current selection. If more than one cast member is selected, a blank value indicates that cast members in the selection have different values. To maintain a cast member's original value, leave that value blank in the dialog box.

Click a dialog box option for more information:



- **Understanding color palettes**
Understanding color depth
Image Options

Size determines the dimensions of the selected cast member. If multiple cast members are selected, you can resize all the cast members to the dimensions you enter. You can either enter new measurements (in pixels) in the Width and Height fields, or enter a scaling percentage in the Scale box.

Check the Maintain Proportions box to keep the width and height of the selected cast member in proportion. If you change the width, the proportional height is automatically entered in the Height field.

Color Depth sets the color depth of the selected cast member. A cast member's color depth is determined at import by the selection set in the import dialog. A movie's color depth is determined by the cast member with the highest color depth.

You can change color depth to save memory and disk space when you are creating a color movie. For more information about color depth settings, see **Understanding color depth**.

Palette selects the palette for the selected cast member. The palettes listed in the pop-up menu are the default Director palettes, plus any additional ones found in the casts. You can create a common palette that contains most of the colors your cast member needs.

When you use the Transform Bitmap command to remap the color in a cast member, Director matches the colors of the cast member with similar colors in the new palette. For example, if the original artwork is red and the closest red available in the new palette is pink, the red is changed to pink.

If the movie is playing, the active palette is the one that is currently in use at any given time, as specified in the score. The palette active when you use Transform Bitmap may be different from the palette used by the movie.

Remap Colors replaces the image's colors with the most similar solid colors in palette you select from the pop-up menu.

Dither blends the colors in the new palette to approximate the original colors in the graphic.

- **Convert to Bitmap command (Modify menu)**

This command converts fields to bitmapped cast members. The converted graphic can then be edited in the paint window. Once you convert a cast member to a bitmapped graphic, you cannot undo the change.

You can't convert a shape to a bitmap.

• **Control menu**

Click the name of a menu command for more information:

Play

Stop

Rewind

Step Forward

Step Backward

Loop Playback

Selected Frames Only

Volume

Disable Scripts

Toggle Breakpoint

Watch Expression

Remove All Breakpoints

Ignore Breakpoints

Step Script

Step Into Script

Run Script

Recompile All Scripts

- **Play command (Control menu)**

Control+Alt+P

The Play command starts the movie. If you press the Shift key while choosing Play, the menu bar is hidden and the stage is cleared of all open windows as the movie plays.

Shortcut: The keypad + key toggles between Play and Stop.

- **Control Panel**

- **Stop command (Control menu)**

Control+period (.)

The Stop command halts the movie.

Shortcut: The keypad + key toggles between Play and Stop.

- **Control Panel**

- **Rewind command (Control menu)**

Control+Alt+R

Rewind moves the playback head back to frame 1. If the animation is playing, it also stops.

Shortcut: Keypad 0 rewinds the movie.

- **Control Panel**

- **Step Backward command (Control menu)**

Step Backward steps the movie backward one frame at a time.

Shortcut: Keypad 1 or Control+left arrow steps the movie backward.

- **Control Panel**

- **Step Forward command (Control menu)**

The Step Forward command advances the movie forward one frame. When using the step recording technique it can be used to record cast members to the next frame of animation.

Shortcut: Keypad 3 or Control+right arrow steps the movie forward.

- **Control Panel**

- **Volume submenu (Control menu)**

The Volume submenu specifies the sound level of the movie. Click the menu and choose a level from 0 (mute) to 7 (loud).

Shortcut: Keypad 7 toggles between sound on and sound off.

- **Control Panel**

- **Loop Playback command (Control menu)**

Control+Alt+L

If selected, the Loop Playback command causes the movie to repeat continuously when played. When the movie reaches the last frame, it automatically starts again from frame 1. By default, this option is on.

Shortcut: Keypad 8 causes the movie to loop.

- **Control Panel**

- **Selected Frames Only command (Control menu)**

If selected, Selected Frames Only designates a range of frames that can be played. This is convenient if you are working on just one part of a movie.

To play a portion of a movie, open the score and select the frames to be played. Choose Selected Frames Only, make sure loop is turned on, and play the movie.

When a portion of the score has been marked as selected frames, a green bar appears at the top of the score over the selected frames.

Turn off Selected Frames Only when you want to return to normal play mode.

This command is dimmed if no frames are selected in the score.

- **Control Panel**

- **Disable Scripts command (Control menu)**

This command lets you ignore scripts during playback. If this command is not selected, Director executes all scripts during playback. This is the default. If this command is selected, Director ignores all scripts in the movie during playback.

This command is useful when you want to control whether interactivity is on or off during playback, or if you want to preview exporting a range of frames as a digital video movie.

- **Toggle Breakpoint command (Control menu)** **Control+Shift+Alt+K**

This command inserts and removes breakpoints for the line of Lingo that the script window cursor is in. Director opens the debugger window whenever it encounters a breakpoint in Lingo.

When the line of Lingo has a breakpoint, the Toggle Breakpoint command removes it and when there is no breakpoint it inserts one.

- **Debugger window**

- **Watch Expression command (Control menu) Control+Shift+Alt+W**

The Watch Expression command adds any expressions and variables to the watcher window that are in the line of Lingo that the script window's text cursor is currently in. This has the same effect as clicking the Watch Variable button in the script window.

- **Remove All Breakpoints command (Control menu)**

This command removes all breakpoints from the movie's scripts.

- **Debugger window**

- ◆ **Ignore Breakpoints command (Control menu)Control+Shift+Alt+I**

This command has Lingo ignore any breakpoints in the movie's scripts as the movie plays.

- ◆ **Debugger window**

- ◆ **Step Script command (Control menu) Control+Shift+Alt+down arrow**

The Step Script command runs the current line of Lingo but doesn't run any handlers that the line calls.

- ◆ **Debugger window**

- ◆ **Step Into Script command (Control menu)Control+Shift+Alt+right arrow**

The Step Into Script command runs the current line of Lingo and follows Lingo's normal flow through any handlers called by that line.

- ◆ **Debugger window**

- ◆ **Run Script command (Control menu)** **Control+Shift+Alt+Up arrow**

The Run Script command is only available when Lingo has stopped at a breakpoint. Use this command to restart Lingo.

- ◆ **Debugger window**

- **Recompile All Scripts command (Control menu)Control+Shift+Alt+C**

This command recompiles all Lingo scripts and checks them for errors. If a script error is found, the appropriate script window opens and the error is selected.

If a script window is the active window, Director first saves any unsaved changes in the current script window and compiles its script before continuing.

●
Xtras menu

Click a command or submenu for more information:

Update Movies

Filter Bitmap

Auto Filter

Auto Distort

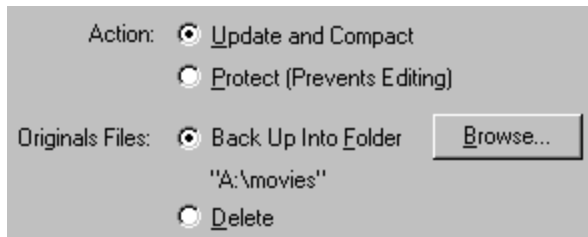
- **Update Movies command (Xtras menu)**

Use the Update Movies command on the Xtras menu to:

- Update movies from Director 4.x movies to the latest file format
- Remove redundant and fragmented data in movie and cast files
- Prevent users from opening movie and cast files.
- Batch-process movie and cast files in large projects.

- **Protecting and compacting with Update movies**
Converting Director 4 movies

Click a dialog box option for more information:



Update and Compact

Update and Compact updates movies from Director 4 or later. As it updates movies, Director consolidates and removes fragmented data. You can also use this option to compact files from the current version of Director. (To update movies from older versions, you must first convert them to the Director 4 file format.)

Protect

Protect makes a movie or cast uneditable. It prevents users from opening the movie or cast and making changes. Protect compacts the movie in the same way as Update and Compact, but it makes the movie even smaller by removing lingo script text and thumbnails. Once a movie is protected, there is no way to "unprotect" it, so be sure to keep an unprotected copy.

Back Up Into Folder

Back Up Into Folder specifies that the original files should be placed in a selected folder. Click Browse to select the folder for the original files. To avoid overwriting old backups, you should choose a new folder each time you run Update Movies.

Delete

Delete specifies that the original files should be overwritten by the newly updated files. Be very careful using this option, especially if you are protecting files. Once a file is protected, you cannot open it in Director.

- **Filter Bitmap command (Xtras menu)**

The Filter Bitmap command displays all the filters you have installed as Xtras. Filters are plug-in image editors that apply effects to bitmapped images. You can also install Adobe Photoshop and Premiere.

Filter Bitmap Dialog options

Categories displays the categories of available filters. These categories are defined by the filters themselves. When you select a category, the filters in that category appear in the Filters list to the right. Choose All to view filters in all categories.

Filters displays all the filters in the current category.

Filter button activates the current filter and opens the filter controls.

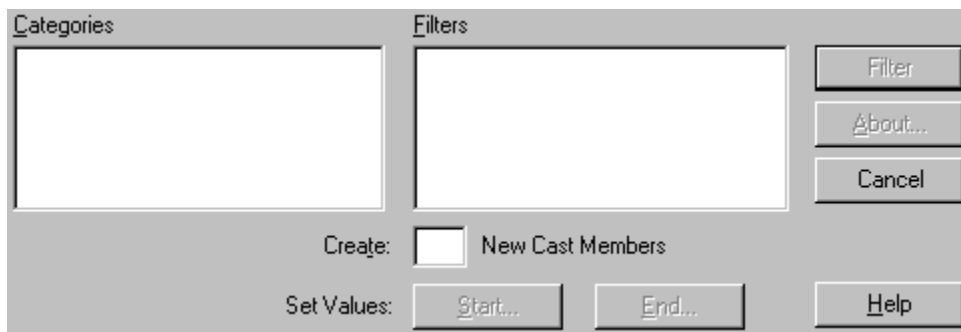
- **Applying a filter to a bitmapped cast member**
Installing Xtras
Auto Filter command

- **Auto Filter command (Xtras menu)**

Use Auto Filter to create dramatic animated effects with filters. Auto Filter applies a filter incrementally to a series of cast members. You can use it either to change a range of selected cast members, or to generate a series of new filtered cast members based on a single cast member. You define a beginning and ending setting for the filter, and then Auto Filter applies an intermediate filter value to each cast member.

For example, if a filter converts an image to look as if it is breaking apart like glass, you can apply it to a cast member with Auto Filter and create a series of ten cast members. The first would show the pieces just coming apart and the last would show the pieces completely fragmented. You could then show the image breaking apart in an animation.

Click a dialog box option for more information:



Auto Filter generates new cast members and places them in empty positions following the cast member you selected. If you selected a range of cast members, no new cast members appear, but the cast members in the range you selected are changed incrementally.

- **Installing Xtras.**
Using Auto Filter

Categories displays the categories of available filters. These categories are defined by the filters themselves. When you select a category, the filters in that category appear in the Filters list to the right. Choose All to view filters in all categories.

Filters displays all the filters in the current category.

Set Values:

- **Start** defines the filters settings of the first cast member to be filtered.
- **End** defines the filter settings for the last cast member to be filtered.

Create _ New Cast Members defines the number of new cast members that will be created. This option is not available if you select a range of cast members.

Filter activates the current filter and opens the filter controls.

- **Auto Distort command (Xtras menu)**

The Auto Distort command automatically generates in-between positions for any cast member that is free rotated, made into a perspective, slanted, distorted, or stretched. After artwork has been altered with one of these five effects, and before you deselect the artwork, choose Auto Distort, and enter the number of in-between cast members in the Create New Cast Members field in the Auto Distort dialog box. The new cast members are placed in the next available cast member positions.

For a rotated bitmap image, Auto Distort uses the center of the image as the rotation point. Consequently, you will have to use the paint window's registration tool to reset the registration point of each in-betweened cast member created by the Auto Distort operation.

- **Using registration points**

- Use the Auto Distort command in conjunction with Rotate Left, Rotate Right, Free Rotate, Perspective, Slant, or Distort (buttons in the Effect toolbar) to quickly create a number of cast members in between the artwork you selected and the artwork that has been changed. Auto Distort also works with artwork that is stretched or squeezed in the paint window.

Director 5.0 includes many new features in response to user requests:

Anti-aliased text support

Font dialog box

Paragraph dialog box

Text Inspector

Importing text

Multiple casts

Cast Properties dialog box

Grid on stage

Alignment

Photoshop filter support

Importing at different color depths

Xtras

New Lingo elements

Lingo that has changed in 5.0

Lingo that is outdated

Debugger window

Watcher window

Script window enhancements



Director 5.0 includes many new features in response to user requests:

Anti-aliased text support

Font dialog box

Paragraph dialog box

Text Inspector

Importing text

Multiple casts

Cast Properties dialog box

Grid on stage

Alignment

Photoshop filter support

Importing at different color depths

Xtras

New Lingo elements

Lingo that has changed in 5.0

Lingo that is outdated

Debugger window

Watcher window

Script window enhancements

Click a category for more information:

Shortcut menus

File menu shortcuts

Edit menu shortcuts

View menu shortcuts

Insert menu shortcuts

Modify menu shortcuts

Control menu shortcuts

Window menu shortcuts

Stage shortcuts

Cast window & cast editor shortcuts

Paint window shortcuts



Click a category for more information:

Shortcut menus

File menu shortcuts

Edit menu shortcuts

View menu shortcuts

Insert menu shortcuts

Modify menu shortcuts

Control menu shortcuts

Window menu shortcuts

Stage shortcuts

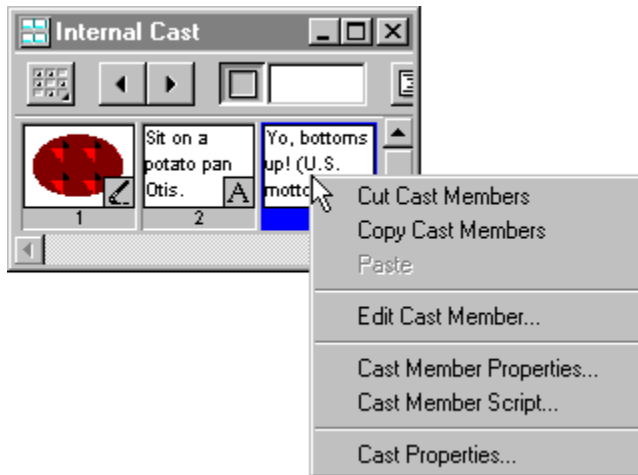
Cast window & cast editor shortcuts

Paint window shortcuts

Shortcut menus

Director supports shortcut menus throughout the user interface.

To display a shortcut menu, right-click a sprite or cast member, or in any window. A menu of commonly used commands is displayed.





File menu shortcuts

Command	Shortcut
New Movie	Control+N
New Cast	Control+Alt+N
Open	Control+O
Close	Control+F4
Save	Control+S
Import	Control+R
Export	Control+Shift+R
Page Setup	Control+Shift+P
Print	Control+P
General Preferences	Control+U
Exit	Alt+F4



Edit menu shortcuts

Command	Shortcut
Undo	Control+Z
Repeat	Control+Y
Cut	Control+X
Copy	Control+C
Paste	Control+V
Clear	Delete
Duplicate	Control+D
Select All	Control+A
Find Text	Control+F
Find Handler	Control+Shift+;
Find Cast Member	Control+;
Find Selection	Control+H
Find Again	Control+Alt+F
Replace Again	Control+Alt+E
Exchange Cast Members	Control+E



View menu shortcuts

Command	Shortcut
Next Marker	Control+right arrow
Previous Marker	Control+left arrow
Zoom In	Control++
Zoom Out	Control+-
Show Grid	Command+Shift+Alt+G
Snap to Grid	Control+Alt-G



Insert menu shortcuts

Command	Shortcut
Insert Frame	Control+]
Remove Frame	Control+[



Modify menu shortcuts

Command	Shortcut
Cast Member Properties	Control+I
Cast Member Script	Control+' (apostrophe)
Sprite Properties	Control+Shift+I
Sprite Script	Control+Shift+' (apostrophe)
Movie Properties	Control+Shift-D
Movie Casts	Control+Shift+C
Font	Ctrl+Shift+T
Paragraph	Control+Shift+Alt+T
In-Between	Control-B
In-Between Special	Control-Shift-B
Bring to Front	Control+Shift+Up arrow
Move Forward	Control+Up arrow
Move Backward	Control+Alt+Down arrow
Send to Back	Control+Alt+Shift+Down arrow
Align	Control-K
Tweak	Control-Shift-K



Control menu shortcuts

Command	Shortcut
Play	Control+Alt+P
Stop	Control+period (.)
Rewind	Control+Alt+R
Step Backward	Control+Option+left arrow
Step Forward	Control+Option+right arrow
Loop Playback	Control+Alt+L
Volume: Mute	Control+Alt+M
Toggle Breakpoint	F9
Watch Expression	Shift+F9
Ignore Breakpoints	Alt+F9
Step Script	F10
Step Into Script	F8
Run Script	F5
Recompile All Scripts	Control+Shift+Alt+C



Window menu shortcuts

Command	Shortcut
Toolbar	Control+Shift+Alt-B
Tool Palette	Control+7
Text Inspector	Control+T
Stage	Control+1
Control Panel	Control+2
Markers	Control+Shift+M
Score	Control+4
Cast	Control+3
Paint	Control+5
Text	Control+6
Field	Control+8
Color Palettes	Control+Alt+7
Video	Control+9
Script	Control+0
Message	Control+M
Debugger	Control+` (back single quote)
Watcher	Control+Shift+` (back single quote)



Stage shortcuts

Action	Shortcut
Open cast member editor	Double-click sprite
Open paint window	Control+5
Inks pop-up	Control+click
Toggle record light on and off	Alt-click
Real-time record	Control-Spacebar-drag sprite on the stage
Display shortcut menu for selection	right-click



Score window shortcuts

Action	Shortcut
Duplicate selected cells	Alt+drag
Open cast editor for selected sprite	Double-click cast thumbnail
Select entire range of a cast member	Double-click cell with a sprite in it
Select channel	Double-click channel number
Select multiple channels	Double-click channel number and drag up or down
Toggle record light	Alt+click channel number
Move playback head to end of movie	Tab
Move playback head to beginning of movie	Shift+Tab
Move playback head to beg/end	Control+Shift+left/right arrow
Open settings dialog box	Double-click tempo, palette, or transition channel
Go to next marker comment (or jump 10 frames)	Control+right arrow
Previous marker comment (or back 10 frames)	Control+left arrow
Shuffle backward	Control+up arrow
Shuffle forward	Control+down arrow



Cast window & cast editor window shortcuts

Action	Shortcut
Open cast member editor	Double-click a paint, text, palette or script cast member or select the cast member and press Return
Cast member script	Control+' (apostrophe)
Switch selected cast member with score selection	Alt+double-click thumbnail
Display cast member info	Control-click cast thumbnail
Open script in new window	Alt+Script button
Place button	Control+Shift+L (places selected cast member in center of stage)
Cast to Time (Option-Place button)	Control+Shift+Alt+L
*Add button (new cast member)	Control+Shift+A
*Left arrow (previous cast member)	Control+left arrow
*Right arrow (next cast member)	Control+right arrow
* same function, in a new window	Alt+left/right arrow
Scroll up/down one window	Page up, Page down
Scroll to top left of cast window	Home
Scroll to show last occupied cast member	End
Type-select by cast member	Type number.

Paint window shortcuts

Action	Shortcut
Undo	~ (tilde)
Next/previous cast member	Keypad left/right arrow keys
Turn selected tool into foreground eyedropper	D key, while pressed
Turn selected tool into background eyedropper	Shift-D key
Turn selected tool into destination eyedropper	Alt+D key
Turn selected tool into hand tool	Spacebar, while pressed
Nudge selection rect. or lasso selection	Keypad arrows with selection rectangle or lasso
Change airbrush size (while painting)	Keypad up/down arrows with airbrush selected
Change airbrush flow (while painting)	Keypad left/right arrows with airbrush selected
Change foreground color (not painting)	Keypad up/down arrows, all tools
Change background color (not painting)	Shift-keypad up/down arrows, all tools
Change destination color (not painting)	Alt+keypad up/down arrows, all tools
Draw border w/current pattern	Alt+shape or line tools
Select background color	Shift-eyedropper
Select destination color	Alt+eyedropper
Toggle between custom and grayscale patterns	Alt+click pattern
Polygon lasso	Alt+lasso
Duplicate selection	Alt+drag
Stretch	Control+drag
Draw with background color	Alt+pencil tool
Open Gradient Settings dialog box and set ink to gradient	Double-click paintbrush, rectangle, paint bucket, or polygon tool
Open Air Brush Settings dialog box	Double-click airbrush
Clear visible part of window	Double-click eraser
Open color palettes window	Double-click foreground, background, or destination color chip
Open Pattern Settings dialog box	Double-click pattern chip
Open Brush Settings dialog box	Double-click paintbrush
Open Paint Window Preferences	Double-click line width selector
Open Transform Bitmap dialog box	Double-click color resolution indicator
Toggle Zoom in/Zoom out	Control+click in window or double-click pencil tool

• **Window menu**

The commands in the Window menu open and close Director's authoring windows. Open windows have checkmarks next to their names.

Click the name of a menu command or submenu for more information:

New Window
Toolbar
Tool Palette
Text Inspector
Memory Inspector
Stage
Control Panel
Markers
Score
Cast
Paint
Text
Field
Color Palettes
Video
Script
Message
Debugger
Watcher

Director automatically hides all open windows if you choose Stage from the Window menu.

- **New Window command (Window menu)**

The New Window command duplicates the front-most window and its contents, creating another view of it. This command only works if a text, cast, digital video, or script window is the front-most window.

Duplicating a window is useful if the window's contents are large and you want to look at or edit different sections of the window simultaneously. It is especially useful for viewing several casts at once. Changes you make in the window are automatically reflected in all other views of the same window.

Shortcut: Press Alt while choosing a text, digital video, script, cast, or field window from the Window.

● **Toolbar (Window menu)** **Control+Shift+Alt+B**

The buttons on the toolbar provide shortcuts for common commands and functions.

The Toolbar command on the Window menu shows or hides the toolbar underneath the menu bar. Click a tool for more information:



New Movie

New Cast

Open

Save

Print

Import

Undo

Cut

Copy

Paste

Find Cast Member

Exchange Cast Members

In-Between

Align

Rewind

Stop

Play

Cast window

Score window

Paint window

Text window

Script window

Message window

- **Tool palette (Window menu)**

Control+7



Click a tool shown at the left for more information.

- Text, shapes, and buttons you create with tools appear as cast members in the cast window and the score window.
- Shapes are QuickDraw graphics, not bitmaps.
- Shapes print better than bitmaps, but they animate more slowly.
- **Understanding shapes and bitmaps**
Creating shapes

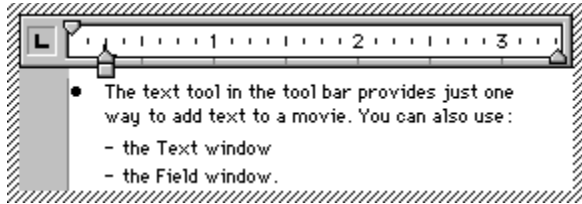
- **Selection (arrow) tool**

The selection tool is a standard selection arrow.

- **Text tool**

The text tool creates text cast members directly on the stage. Click the text tool and then drag to define the area on the stage where you want text. When you release the mouse button, a text insertion point appears in the area you just defined and you can begin entering text. The new text cast member is placed in the first available position in the current cast. The sprite is placed in the first open score cell in the current frame.

Click the arrow to select text that is already on the stage. You can change the color of selected text using the foreground and background color chips in the tools window.



To edit the text cast member on stage, click once to select and move the sprite or double-click to edit the text. Click and drag the handles to change the size of the text box. Add the onstage ruler and define tab settings in the text box by choosing **Ruler** from the View menu. When you make a change, Director updates all instances of the text cast member.

The text tool is an alternative to the text window for creating text. The **Text window** is faster and more convenient for working on substantial amounts of text.

You can use **Sprite Properties** in the Modify menu to change or determine the size, location, or blend of a selected sprite. Any changes to a sprite's properties only affects the sprite's appearance on the stage and does not alter the actual properties of the cast member.

- **Understanding text and fields**
Text Inspector
Font
Paragraph

- **Line tool**

Click the line tool and drag it across the stage to draw. You can choose the color for the line with the foreground and background pop-up that appears when you click the color chips. The width of the line tool is controlled by the line width selector at the bottom. The line tool is constrained to horizontal, vertical, or 45-degree lines with the Shift key.

- **Understanding shapes and bitmaps**

- ◆ **Shape tools**

Click the tools on the right to draw an outline of the selected shape or on the left to draw shapes with a solid color or pattern.

Choose the color for the shape with the foreground and background palettes that appear when you press on the color chips.

Use the pattern chip to select the current pattern. Use the line width selector at the bottom of the tools window to control the thickness of the borders of the shape tools.

- ◆ **Understanding shapes and bitmaps**

- Creating shapes**

- Shape Cast Member Properties**

- **Field tool**

The field tool creates field cast members directly on the stage. Click the field tool and then drag to define the area on the stage where you want the field.

Use fields for creating user-editable text in movies or text that you want to format with Lingo. Use the text tool to create all other types of text.

- **Understanding text versus fields**
Field window
Text window

- **Button tools**

Director provides three tools for creating buttons, checkboxes, and radio buttons. Click the checkbox tool, button tool, or radio button tool and drag a rectangle on the stage to create the button. Then type the text that you want to appear on or next to the button. Set the font, style, and size. The button is placed in the cast as a button cast member. You can edit the button's text on the stage or in a text window.

Buttons do not perform any special function until you write a Lingo script for them.

- **Button Cast Member Properties**

● **Foreground and background color chips**

The foreground and background color chips in the tool palette set the color of text, shapes, and sprites.

To set the color for a sprite, select the sprite in the score or on the stage and choose a new foreground color using the foreground color chip, or a new background color using the background color chip.

To set the color of text, select the text you want to change and then choose a text color using the foreground color chip. To set the text's background color, choose a color using the background color chip.

If you apply color to a 1-bit cast member, Director changes the color of the sprite on the stage but does not change the color of the actual cast member, which remains black and white.

- **Pattern Settings**

The pattern selector lets you select the current pattern for a shape tool. It also provides access to the **Tile Settings** and pattern settings dialog box.

- **Line width selector**

Lets you select the current width of the line tool, or the border for a shape tool.

• **Text inspector (Window menu)**

Control+T

The text inspector is a floating window that provides tools to edit text cast members directly on the stage. The tools are shortcuts for the formatting options in the **Paragraph** and **Font** dialog boxes.



Font pop-up is used to choose any font in your system.

Bold, italic, underscore buttons apply character formatting to selected text.

Size specifies the font size. Choose a size from the pop-up menu, or enter a size in the field. You may need to adjust the line spacing.

Line spacing displays the line spacing in points. Click the up and down arrows to change the setting, or enter a value in points.

Alignment aligns selected paragraphs, left aligned, centered, right aligned, or fully justified.

● **Memory inspector (Window menu)**

The memory inspector displays information about how much memory is available to Director for your movie. It also indicates how much memory different parts of the current movie use and the total disk space the movie occupies.

Total Memory shows you the total memory available in your system. This number depends on the amount of RAM installed on your computer and any virtual memory that's available.

Physical Memory indicates the memory the amount of RAM available on your computer.

Free Memory indicates how much more memory is currently available in your system.

Other Memory indicates the amount of memory taken up by Windows, by DIRECTOR.EXE, and by other applications.

Used by Program indicates the amount of memory used by Director(excluding the amount of memory taken up by DIRECTOR.EXE).

Mattes & Thumbs shows how much memory is used by cast members that use the Matte ink in the score and by thumbnail images in the cast window.

Cast & Score indicates the amount of memory used by the cast members in the cast window and the notation in the score window. Cast members include all the artwork in the paint window, all the text in the text windows, and any sounds, palettes, buttons, digital video movies, or linked files imported into the cast and currently loaded into memory.

Screen Buffer shows how much memory Director reserves for a "working area" while animating on the stage.

Memory Limit indicates the memory limit assigned to Director in the Limit Memory Size to box of the General Preferences dialog.

Total Used indicates how much RAM is being used for a movie.

Purge button removes all purgeable items from RAM, including all thumbnail images in the cast window. All cast members that have Unload (purge priority) set to a priority other than "0-Never" (as specified in the Cast Member Properties dialog box) are removed from memory. This is useful for gaining as much free memory as possible before importing a large file. Edited cast members don't get purged.

- **Stage (Window menu) Control+1**

The stage is the backdrop for all Director movies. Set the size of the stage with the Movie Properties command on the Modify menu. In most cases, the edges of the stage window extend to the edges of your screen, so you can use all of the monitor for your movie. Movies continue to play on the stage when other windows are open and active. This permits you to study the movie in the score, for example, while keeping an eye on the stage.

The Stage command in the Window menu brings the stage to the front of the screen. It temporarily hides all open windows. To open a menu just click and hold.

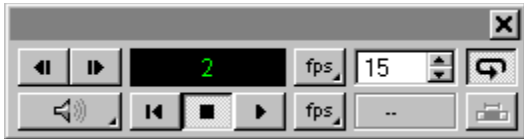
- **Resizing sprites on the stage**
Movie Properties command (Modify menu)

• **Control Panel (Window menu)** **Control+2**

The control panel has controls similar to a VCR. Use it to Play, Stop, Step Forward or Backward, or Rewind your movie. The control panel is also used to loop animation, set tempo, and turn sound on and off. The control panel indicates the current frame number, the current tempo, and the actual duration of the current frame.

Choose Control Panel from the Window menu to show or hide the control panel. The control panel buttons have corresponding commands on the Control menu.

Click part of the control panel for more information:



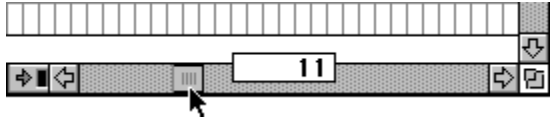
Click the name of a control panel indicator for more information:

Frame counter

Tempo display

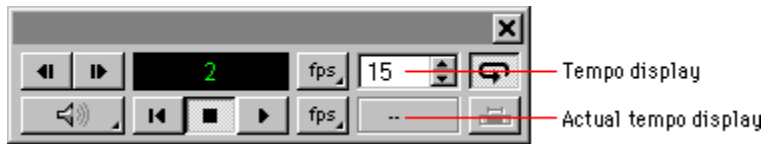
Actual tempo display

- **Frame counter (control panel)**



The frame counter displays the number of the frames currently on the stage. To go to a specific frame number, click the field, type a frame number, and press Enter.

- **Tempo display (control panel)**



The tempo display shows the tempo of the current frame. You can view the tempo in either FPS (Frames Per Second), or SPF (Seconds Per Frame). Click the tempo mode button and choose the mode you want from the pop-up menu. Seconds per frame measures the duration of a frame in milliseconds.

To change the tempo, click or press the up or down arrow next to the tempo indicator. To use a specific tempo, click into the tempo display area, type a tempo, and press Enter. If you are entering a tempo in frames per second (FPS), you must enter a whole number. If you are entering a tempo in seconds per frame (SPF), you must type a decimal (.) before entering a value.

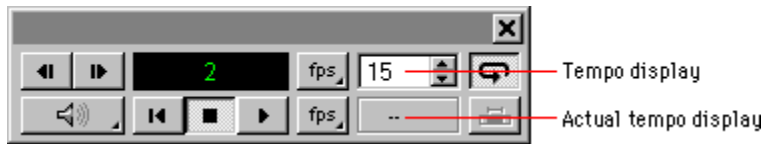
The tempo displayed is the tempo defined for the current frame, but the actual tempo may be different. Director attempts to play the movie at the specified tempo setting, but the speed of the computer and the complexity of the movie also affect the tempo. Use the actual tempo display to determine the real tempo of the movie.

If there are no tempo settings in the tempo channel, the control panel displays the default tempo. (If there is a tempo setting in the tempo channel, the control panel displays the tempo of the current frame.)

- You should always enter a tempo setting in the first frame in the tempo channel.

Related topic: **Actual tempo display**

- **Actual tempo display (control panel)**



While a movie is playing, the actual tempo display shows how fast the movie is going. Click the actual tempo mode button and choose a mode for viewing the tempo from the pop-up. There are four different modes:

- FPS (Frames Per Second) shows the actual duration of the previous frame in frames per second (FPS), including the time necessary to execute any Lingo scripts except Exit Frame scripts. If the movie is stopped, the display shows the duration of the current frame. Frames that don't have a recorded tempo value display "--" instead of a value for the actual tempo.
- SPF (Seconds Per Frame) shows the duration of the current frame in milliseconds.
- Running Total provides a quick summary of elapsed seconds from the beginning of the movie to the current frame.
- Estimated Total provides a more accurate but slower calculation of elapsed time. It is useful if you want to include transitions and palette changes in determining frame durations.

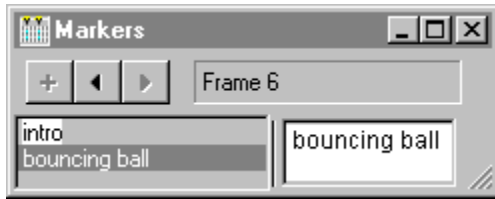
Computing estimated frame durations can reduce playback speed, so don't leave the Actual display in Estimated Total mode.

If the movie is locked (using the Lock Frame Durations option in the Movie Properties dialog box), the Actual display shows the previously recorded frame durations.

-
- Clear all recorded frame durations from the movie if you want to record frame durations for a section of the movie and lock them. To clear all recorded frame durations, press Alt while clicking the Lock option in the Movie properties dialog. During playback, frames that don't have a recorded duration instead display "--" .
- Since not all computers are equally fast, you can step through the movie frame-by-frame and compare the actual frame durations to the tempos you've set for the movie. To find frames that have longer durations than the current tempo, set both the Tempo and Actual displays to show seconds per frame (SPF); then step through the range of frames, and look for any frame whose actual duration is longer than the current tempo.

- **Tempo display**

- **Markers window (Window menu)** **Control+Shift+M**



The markers window lets you write comments associated with markers you set in the score. For example, a Director animation can have staging or acting directions, storyboard scripts, or speaker's notes written in the markers window and tied to specific frames in the score. A storyboard, transparencies, or handouts can be printed that include pictures of selected frames of your movie along with the comments written in the markers window. Double-clicking a marker in the score opens the markers window to the comment associated with that frame.

Once you've labeled a frame in the score, use the marker name in your scripts. This is important because references to frame numbers may become invalid if you insert or delete frames in the score. Marker names remain constant no matter how much you edit the score.

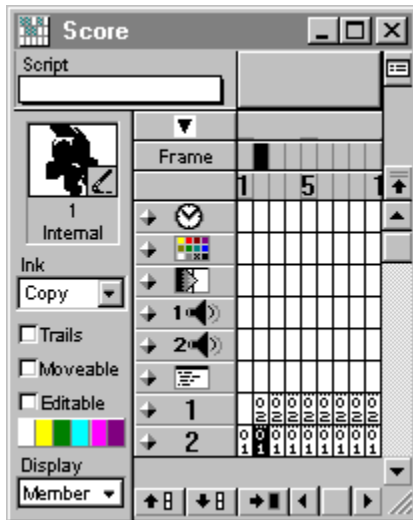
The left column of the markers window displays the marker names from the score window. Clicking a marker name or clicking the left or right arrow in the markers window moves the playback head to the selected marker location in the score window, and displays the comments associated with the marker in the markers window.

To enter a comment, click a marker name to select it. You can then enter your comments beginning at the insertion point that appears in the right column of the markers window. By default, the marker name appears as the first line of text in the right column. If you edit the marker name, your changes are also reflected in the score window. If you don't want to edit the marker name, press Enter to start a new line, and enter your comments on the new line.

- **Next Marker (View menu)** and **Previous Marker (View menu)**

● **Score window (Window menu) Control+4**

Click an element for more information or click a name of part of the score window:



Script preview button

Trails checkbox

Script pop-up

Moveable checkbox

Ink pop-up

Editable checkbox

Display pop-up

Selecting cells

Moving selections within the score

Moving around the score

Applying color to cells

Step-recording in a channel

Real-time recording in a channel

Turning a channel on and off

The score window contains the notation that describes your movie and is the primary tool for creating and editing animation. The score window contains a record of everything that happens on the stage.

- **Cells (score)**

The smallest unit in the score is a cell. Each cell contains information about one cast member at one point in time, called a sprite. When you select a cell, a small image of the sprite that occupies that cell appears in the upper left corner of the score window. Double-click the sprite's image to open a window in which you can edit the cast member.

- **Frames (score)**

A frame contains information about everything you see on the stage when you stop a movie at the moment that corresponds to the frame. Like a frame in a movie, a Director frame shows what each sprite is doing on the stage at one point in time. Each frame is numbered.

- **Special effects channels (score)**

A channel is a row of frames. The first five rows, called channels, keep track of special effects for each frame:

- Tempo
- Palette
- Transition
- Sounds (two channels)
- Script.

● **Sprite channels (score)**

A channel is a row of frames. Each sprite channel (located below the tempo, palette, transition, sound, and script channels) describes the state of one cast member through all the frames in the movie.

The order of sprites in the 48 animation channels determines which sprites appear in the foreground and which appear in the background. Think of the stage as a pile of transparent sheets, 48 sheets thick. The sprite that occupies a channel closer to the top of the score is like the object drawn on the last sheet of acetate. It appears behind any sprite in a channel closer to the bottom of the score.

Sprites in channels closer to the bottom of the score appear in front of, or take priority over, those in channels closer to the top of the score.

● **Markers (score)**

Markers help you coordinate the comments in the markers window with specific frames of your movie or to identify frames of your movie for printing. You can drag any number of markers from the left side of the score window and position them in any frame. Use this feature to add comments, speaker notes, or storyboard notes to specific frames of animation.

When you position a marker in the score, an insertion point appears to the right of the marker so you can type a short comment for that marker. Use the **Markers window** to review and edit marker comments.

- **Special effects channel toggle (score)**

Displays and hides the special effects channels (tempo, palette, transition, sound, and script) of the score.

- **Cast member preview (score)**

Displays a small preview of the cast member for the selected sprite in the score.

- **Shuffle forward/backward buttons (score)**

Moves the selected cell or cells up or down a channel.

- **Jump button (score)**

Moves the view of the score to the position of the playback head.

- **Channel playback toggle (score)**

Click this diamond-shaped toggle to turn a channel on or off.

Turning a channel off tells Director to ignore the channel during playback. By default, no channels are ignored (i.e., all channels are active). Clicking the button next to a channel turns the channel off, causing Director to ignore that channel when you play the movie.

If you turn the script channel off, Director ignores all scripts during playback. (This is the same as checking the Disable Scripts command in the Control menu.)

- **Display pop-up menu (score)**

The Display pop-up lets you change the type of information displayed in each cell of the score. It is located in the lower left corner of the score window. Use it to view different types of notation in the score. By default, the score displays cast member notation.

- **Display submenu (View menu)**

Cast member display notation

The cast display depends on how you have chosen to view cast members in the selected cast window using the **Cast Preferences** command on the File menu. If the cast window displays cast members by decimal number, or by Number:Name, the cast display shows the last two digits of the cast member's position number. If the cast window displays cast members by name, the cast display shows the first two letters of the cast member's name, if a name exists for that cast member.



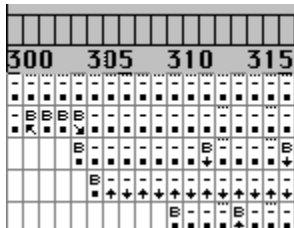
Script display notation

The script display shows the number of the script associated with each sprite. Each script has an identifying number associated with it. Sprites without scripts display 00. Sprites with cast member scripts display a plus (+) sign in the cell.



Motion display notation

The motion display shows the direction of the cast member's movement on stage and whether or not a new cast member appears in that cell.



Ink display notation

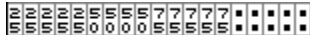
The ink display indicates which ink effect has been applied to each sprite.



- ◆ **Ink popup menu**

Blend display notation

The blend notation shows the blend percentage that's applied to the sprite. Choose **Sprite Properties** in the Modify menu to set the blend display notation.



Extended display notation

The extended display enlarges the score and displays all the sprite information at once.

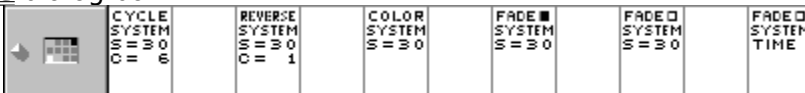
Select **Score Preferences** from the File menu to choose the information you want to appear in the extended display.

If you choose to view cast members by name in the cast, the extended display shows the first few letters of the cast member's name instead of the cast member's number.

- ◆ Notation in the tempo channel is determined by the settings specified in the **Frame Tempo** dialog box.



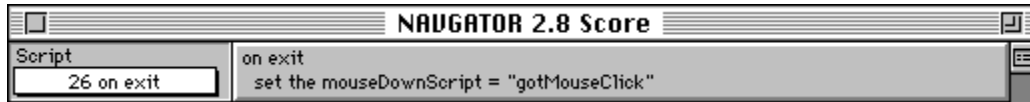
- ◆ Notation in the palette channel is determined by the settings specified in the **Frame Palette** dialog box.



- ◆ Notation in the transition channel is determined by the settings specified in the **Frame Transition** dialog box.



- **Script preview button (score window)**

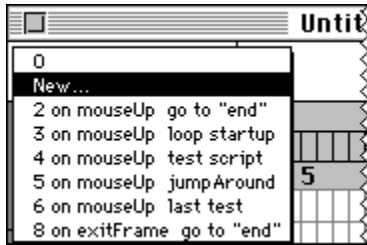


The script preview button displays the first two lines of the script associated with one or more selected cells in the Script channel. If the selection has no script associated with it, the Script preview button is blank. Clicking the Script preview button opens a new script window.

To hide the Script pop-up menu and the preview button, click the show/hide script button.



- **Script pop-up menu (score window)**



The script pop-up menu lists all the frame and sprite scripts used in the current movie. The script pop-up menu displays the script number associated with a selected frame in the script channel. If the selected cell has no script associated with it, the pop-up menu is blank. You can use this pop-up menu to assign existing scripts to areas of the score. Select the cells you want to apply the script to; then choose the script you want from the script pop-up menu.

Choose New Script to create a new score script. Choose Clear Script to remove the script from the selected cell or just the delete the cell.

- - Script window**
 - Script Properties**
 - Types of Scripts**
 - Writing Scripts**

● **Ink pop-up menu (score window)**

You apply ink to sprites to change the way they appear on the stage. The Ink pop-up menu also indicates the current ink applied to selected sprites in the score.

The best way to see how each of the following ink effects work is to play the sample movie Ink Effects on the Director CD.

Copy is the default ink and is useful for backgrounds or for sprites that do not appear in front of other artwork. If the cast member is not rectangular, a white box appears around the sprite when it passes in front of another sprite, or is displayed on a non-white background. Sprites with the Copy ink animate faster than sprites with any other ink.

Matte removes the bounding box (rectangular area) around a sprite. Artwork within the boundaries is opaque. Matte functions much like the lasso in the paint window, in that the artwork is outlined rather than enclosed in a rectangle. Matte, like Mask, uses more memory than the other inks, and sprites with this ink animate more slowly than other sprites.

Bkgnd Trans. makes the pixels in the background color of the selected sprite appear transparent and permits the background to be seen. This effect uses more memory and may make your sprite animate more slowly.

Transparent makes all colors transparent so you can see the artwork through it.

Reverse reverses overlapping colors. A pixel that was originally white becomes transparent and lets the background show through unchanged. Reverse is good for making custom masks.

Ghost is useful for reversing black and white. When it is applied to the foreground sprite, any black pixel turns the pixel beneath it white. Anything white becomes transparent.

Not Copy, Not Transparent, Not Reverse, and **Not Ghost** are variations of the above four effects. The foreground image is first reversed, then the Copy, Transparent, Reverse, or Ghost inks are applied. These are good for odd effects. Like Transparent, the Not Transparent ink is good for reversing black and white. Just choose Not Transparent, select a white fill, then draw a rectangle on stage on top of the artwork you want to reverse.

Mask ink allows you to define exactly what parts of a sprite are transparent and opaque. For mask ink to work, you must place a 1-bit "mask" cast member in the cast window position immediately following the cast member to be masked. The black areas of the mask make the sprite opaque and the white areas make the sprite transparent. This ink is especially useful for sprites in which you want some white areas to be transparent, and some opaque.

For example, to show a white car you would want the white body of the car to be opaque and the windows to be transparent. To create a mask, make a copy of the car in the next cast position, convert it to 1-bit color depth with the Transform Bitmap command, and then fill in the body of the car with black. In the score, apply Mask ink to the sprite of the car. The body of the car becomes opaque and the windows transparent.

Blend ensures that the sprite uses the blend percentage specified in the **Sprite Properties** dialog box.

Darkest compares pixel colors in the foreground and background, and uses whichever pixel color in the foreground or background is darkest.

Lightest compares pixel colors in the foreground and background, and uses whichever pixel color in the foreground or background is lightest.

Add creates a new color that is the result of adding the color value of the foreground sprite with the color value of the background sprite. If the value of the two colors exceeds the maximum color value, the addition wraps around the color scale.

Add Pin is similar to Add. The foreground sprite's color is added to the background sprite's color, but the value of the new color cannot exceed the maximum color value.

Subtract subtracts the value of the foreground sprite's color from the value of the background sprite's color to arrive at the new color. If the color value of the new color is less than the minimum color in the color scale, the new color is determined by wrapping around and starting at the top of the color scale.

Subtract Pin subtracts the color value of the foreground sprite from the value of the background sprite. The value of the new color does not wrap around the color scale.

- Mask and Matte use twice the memory of any other ink because Director has to internally create a duplicate of the artwork.

- **About score window inks**

- **Trails checkbox (score window)**



If Trails is checked, the selected sprite remains on the stage, leaving a trail of images along its path as the movie plays. If Trails is unchecked, the selected sprite is erased from previous frames as the movie plays. The checkbox also reflects the current selection.

- **Moveable checkbox (score window)**

This option is only available if you select one or more cells in the sprite channels in the score. If Moveable is checked, users can move the selected sprite(s) around on the stage during playback and in projectors. If checked, the moveable setting is in effect only when the playback head is executing those frames that contain the moveable sprites. The checkbox also reflects the current selection.

The Moveable checkbox displays a dash (-) if the current selection includes sprites that don't all have the same setting.

• **Editable checkbox (score window)**

This option is only available if you select one or more field sprites. If Editable is checked, users can edit the selected field sprites on the stage during playback. This option is convenient for making a field sprite editable in some frames, and noneditable in others. You can turn this setting off when it is no longer required. If checked, the editable setting is in effect only when the playback head is executing those frames that contain the editable sprites. The checkbox also reflects the current selection. If the current selection includes sprites that don't all have the same setting, the Editable checkbox displays a dash (-).

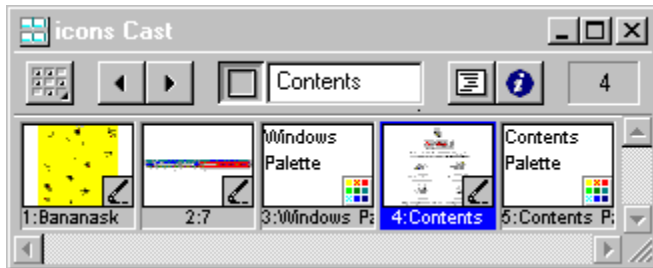
You can set a field cast member to always be editable using the **Field Cast Member Properties** dialog box. If you set a field cast member to be editable in the cast, it is always editable when used in the score. Director ignores the score's Editable checkbox setting for the cast member.

Using the Lingo statement `set the editable of sprite to TRUE` is the same as checking the Editable checkbox in the score.

● **Cast window (Window menu) Control+3**

The cast window displays the cast members in the current cast. A cast is a database of graphics, sounds, color palettes, Lingo scripts, buttons, transitions, digital video movies, and text used in a Director movie.

Click on part of the cast window for more information:



Click a topic for more information:

- [**Cast window positions and thumbnails**](#)
- [**Understanding internal and external casts**](#)
- [**Placing cast members on the stage**](#)
- [**Placing cast members over time**](#)
- [**Dragging cast members to the score**](#)
- [**Moving cast members between casts**](#)
- [**Creating a digital video cast member**](#)
- [**Cast Preferences**](#)
- [**Cast Properties**](#)

Viewing multiple casts

You can open as many cast windows as you need to display the different casts in your movie, or you can select a different cast for display in the current window.

- To open a new cast window for a particular cast in the current movie, choose Cast from the Window menu and then select the name of the cast from the submenu.
- To open a new cast window for the current cast, make a cast window active and then choose New Window from the Window menu.



To change the cast displayed in a cast window, click the cast selector and choose the cast you want to display from the pop-up.

The cast window title bar indicates whether a cast is internal, external linked, or external unlinked.

Cast window size

A movie's cast can contain up to 32,000 cast members. You control the row width and the number of visible rows using [**Cast Preferences**](#) in the File menu.

Selecting cast members

Clicking any cast member selects it. Select a range of cast members by Shift-clicking. Control-click selects multiple non-adjacent cast members. Individual cast members can be cut, copied, pasted, or cleared from the cast window.

• **Cast window positions and thumbnails**

Each cast member position is identified by a number and, optionally, a name. For every occupied position in the cast window, a thumbnail image is displayed that represents the cast member's type.



Note: Thumbnail images for Xtra castmembers vary.

Cast members with scripts display a script indicator icon in the lower left corner. To control whether or not Director displays a script indicator icon in the cast, use the Show Cast Member Script Icons checkbox in the **Cast Preferences** dialog box.

• Double-clicking a cast member is a shortcut for opening the paint window for a graphic cast member, the text window for a text cast member, the color palettes window for a palette cast member, the script window for a script cast member, and the digital video window for a digital video cast member. Alt-double-clicking is a shortcut for opening the cast member in a new window.

Cast selector

Use the cast selector to select which cast is displayed in the current cast window. You can choose between any internal cast or external cast linked to the current movie. You can also choose New Cast to create a new cast.

To open a new cast window, make sure a cast window is active and then choose New Window from the Window menu.

Place button

This button lets you move the selected cast members to the stage or score, or move it within the cast. When cast members are moved within the cast window, the score window is updated with their new position. When you press and hold down this button, the cursor changes to an open hand to let you drag one or more selected cast members. Using this button is the same as clicking and dragging a selected cast member in the cast window. Use this button to move selected cast members that may not currently be visible in the cast window, if you've scrolled to a new location.

To use the Place button to move a cast member to a new location within the cast window:

1. Select the cast members you want to move.
2. Scroll to the new location in the cast window where you want to insert the selected cast member.
3. Drag from the Place button to the new location in the cast window. As you drag within the cast window, a blinking insertion bar indicates the location where the cast member will be inserted.
4. Release the mouse to insert the selected cast member at the new location.

Previous, Next arrows

These arrows let you navigate to the previous or next cast member, skipping over empty cast members.

Shortcut

The keyboard equivalents are:

Previous - Control+Shift+left arrow

Next - Control+Shift+right arrow.

Cast Member Properties button

Cast Member displays the **Cast Member Properties** dialog box for the selected cast member. If the selection consists of more than one cast member, the dialog box displays the number of cast members selected, their total size, and purge priority.

Shortcut: Control+I also opens the Cast Member Properties dialog box.

Open Script button

Opens a new script window or makes an existing script window active for the selected cast member, or for the first selected cast member if more than one are selected. If the selected cast member has no script associated with it, clicking this button opens a new script window, creating a script for the cast member. This is the same as clicking the Script button in the Cast Member Properties dialog box.

Shortcut: Pressing Control+' (apostrophe) is the same as clicking this button.

Cast member number

Cast Member number displays the position of the selected cast member in the cast window or the position of the first selected cast member in a multiple selection.

- When the cast window is front-most, typing the number of an existing cast member automatically scrolls the cast window to the cast member's location and selects it. After you've stopped typing the cast window will scroll to show the new selection.

Cast member name

Cast Member name displays the name of the selected cast member, or the first selected cast member in a multiple selection. Click and type into the name area to enter or edit the name of a cast member. Press Enter to confirm your changes.

Tile Settings

Line width selector

Color resolution indicator

Paint toolbar

Creating a new bitmap cast member

Using rulers

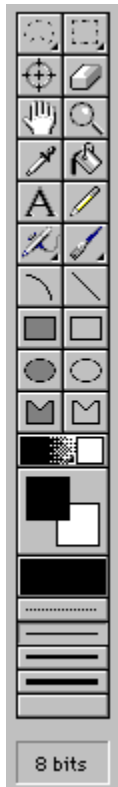
Zooming in and out

Selecting colors and patterns

Drawing with the shape tools

Shortcut: A shortcut for opening the paint window is to double-click a bitmapped cast member on the stage or in the score or cast windows. The paint window opens with that cast member showing.

The paint window tools



The list below shows whether clicking and/or dragging makes the tool work. Click the name of paint window tool for more information:

Lasso--(Drag) Selects irregular shapes

Selection marquee--(Drag) Selects rectangular areas

Hand--(Drag) Moves artwork within window

Paint bucket--(Click) Fills with foreground color or current pattern

Air brush--(Click or drag) Sprays foreground color or current pattern

Paintbrush--(Click or drag) Paints foreground color or current pattern

Text--(Click) Starts text entry

Pencil--(Click or drag) Toggles pixels between foreground and background color

Rectangle--(Drag) Draws hollow or filled rectangles and squares

Eraser--(Drag) Erases artwork

Ellipse--(Drag) Draws hollow or filled ellipses and circles

Polygon--(Click) Draws hollow or filled polygons

Line--(Drag) Draws straight lines

Arc--(Drag) Draws arcs (one quarter of an ellipse or circle)

Registration--(Click) Sets registration point

Eyedropper--(Click or drag) Picks foreground color

Zoom--(Click) Scales the view.

If you press the Control+click the image in the paint window, your view of the artwork will zoom in to a magnified view. In most cases, pressing the Shift key while dragging a tool constrains it to horizontal or vertical. The ellipse and rectangle tools are constrained to a perfect circle or square when Shift-dragging.

- **Lasso (paint window)**

The lasso can be used to select an area. Once selected, you can drag, cut, copy, or clear the artwork. You can also use the following buttons on the effects toolbar: Invert Colors, Trace Edges, Fill, Darken, Lighten, Smooth, and Switch Colors. (The last four commands are only available on a color Macintosh.)

-

- When artwork is selected with the lasso, hold down the Alt key while dragging the artwork to make a copy of it.

- If you press the Alt key while dragging the lasso, the lasso draws straight lines to select a polygon shape. Click the lasso to anchor a point and draw another straight line. Double-click when you reach the end of your selection.

- Pressing the Shift key while dragging the object constrains its movement to a horizontal or vertical line. To move the cast member in one-pixel increments, select it on the stage and use the arrow keys on your keyboard.

Note: Use the lasso to select everything but the pixels of a certain color. The color of the pixels not selected is determined by where you begin to drag the lasso. For example, if you're selecting an object that is red, white, and blue, and you only want to select the red and white pixels in the object, begin your drag on a blue pixel. Then, only the red and white pixels will be selected. If you want to avoid this effect, use the No Shrink option in the **Lasso pop-up**.

- **Lasso pop-up (paint window)**

Pressing and holding the mouse button while the pointer is on the lasso tool causes the lasso pop-up to appear.

Choosing a command from the lasso pop-up modifies how the lasso works.

- **Shrink** causes the lasso to tighten around the selected object so that only the object is selected.
- **No Shrink** permits you to select the entire area you drag around. The lasso selects whatever is inside the selected area.
- **See Thru** causes your selection to become transparent, as if the Transparent ink effect were applied.

- **Selection marquee (paint window)**

The marquee can be used to select artwork in the paint window. When selected with the marquee, artwork can be dragged, cut, copied, and cleared. It can also be modified with the commands on the paint toolbar.

Select the contents of the visible part of the current cast member by double-clicking the marquee.

-
- Stretch and compress art that is selected with the marquee by Control+drag.
- Make a copy of artwork that is selected with the selection marquee, by Alt+drag.

- **Selection marquee pop-up menu**

- **Selection marquee pop-up (paint window)**

If you press and hold the mouse button when the pointer is positioned on the selection rectangle tool, the selection rectangle pop-up menu appears with commands to modify the action of the tool.

- **Shrink** causes the rectangle to shrink around the selected artwork.
- **No Shrink** permits you to select everything within the selection rectangle.
- **Lasso** makes the selection rectangle tighten around your selection like the lasso tool. The selection tightens around the object and selectively selects the pixels according to the color of the pixel beneath the crosshair when you started your drag.
- **See Thru Lasso** makes the selection rectangle tighten around your selection like the lasso and applies the Transparent ink.

Any artwork selected with the lasso or selection rectangle can be further modified with ink effects or commands on the paint toolbar.

Double-clicking the selection marquee tool selects the entire cast member.

As with the lasso, you can reposition the selected area of the cast member. Move the crosshair into the selected area so that the crosshair turns into an arrow pointer and drag the selected area to reposition it. There are several key combinations that affect the selected area when you drag. They include:

- **Copy**--Alt+drag
- **Stretch**--Control+drag
- **Stretch proportionally**--Control+Shift+drag
- **Constrain to horizontal or vertical**--Shift-drag
- **Clear**--Backspace or Delete
- **Scale**--Control+Alt+drag

Use the keyboard arrow keys to horizontally or vertically nudge a selection.

- **Hand tool (paint window)**

The hand tool moves the view within the paint window, changing your position in the window relative to the artwork. Click the hand tool to select it, then drag the artwork to pan the view.

- A shortcut for using the hand is pressing the Spacebar. This turns any tool into the hand tool when the mouse is clicked.

- **Zoom tool (paint window)**

Click to zoom in on an area. Shift-click to zoom out.

- **Text tool (paint window)**

The text tool lets you type in any font, size, or style in the paint window. Use it to set the font, size, and style of the text.

The text you create in the paint window is bitmapped. It can be dragged around the paint window before you deselect the text. However, once you click outside the field after creating the text, you cannot edit its font, size, or style. To change the font, size, or style after clicking, you must erase the text and replace it with new bitmapped text.

You can modify selected text with ink effects from the Ink pop-up, patterns from the patterns pop-up, or foreground and background colors with the foreground and background color chips in the paint window.

- **Text window**
Field window
Understanding text and fields

- **Paint bucket (paint window)**

The paint bucket fills any enclosed area with the currently selected color and pattern. The fill can be further modified with the ink effects in the Ink pop-up menu in the paint window. If there is a break in the outlined area you are filling, the paint will leak out and fill the surrounding area. If this happens, immediately choose Undo Bitmap from the Edit menu. Then Zoom In from the View menu to get a magnified view and inspect the outline for breaks.

- Double-clicking the paint bucket tool opens the **Gradient Settings** dialog box.

● **Air Brush (paint window)**

The air brush sprays the currently selected color and pattern. The spray can be further modified by choosing the ink effects from the Ink pop-up menu in the paint window. The longer you hold the airbrush in one spot, the darker it fills in the area.

Air brush pop-up menu

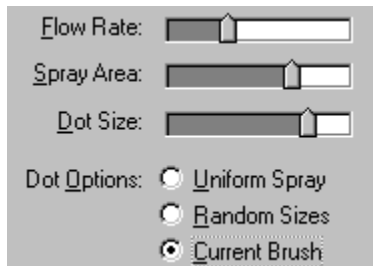
If you press and hold the mouse button when the pointer is positioned on the air brush, the air brush pop-up menu appears. Each of the five settings in the pop-up menu can be defined so you can have several types of spray available without opening the Air Brush Settings dialog box.

To define a setting:

1. Choose the menu item you want to define from the Air Brush pop-up menu.
2. Click the air brush again and choose Settings from the Air Brush pop-up menu.
3. Select the type of spray you want in the Air Brush Settings dialog box.
4. Click Set.

The choices you make in the Air Brush Settings dialog box are assigned to the menu item and remain until you change them.

Air Brush Settings dialog box



The Air Brushes dialog box defines the size of the area the air brush covers, the size of the dots in the air brush's spray, and the flow speed of the air brush's paint. The three radio buttons at the top of the Air Brushes dialog box control whether the air brush sprays uniformly sized dots, randomly sized dots, or dots shaped like the currently selected paintbrush.

● Double-clicking the air brush in the tool palette opens the Air Brush dialog box. Use this dialog box to set the size of the air brush's spray, the size of the dots of paint it sprays, and how fast it sprays paint.

Uniform Spray causes drops sprayed by the air brush to be uniformly sized.

Random Sizes sprays with randomly sized drops.

Current Brush sprays with drops shaped like the current paintbrush.

Spray Area sets the size of the air brush's spray area. To change the spray area, drag the Size scrollbar.

Dot Size sets the size of the dots sprayed by the air brush. To change the dot size, drag the Dot Size scrollbar.

Flow Rate controls how fast the air brush covers an area with paint. To change the flow, drag the Flow Speed scrollbar.

● **Paintbrush (paint window)**

The paintbrush draws with the currently selected colors, ink effect, or fill pattern. Double-click the paintbrush to change the size and shape of the brush. When the Brush Settings dialog box appears, click the brush shape you need, and then click Set.

Paintbrush pop-up menu

The paintbrush pop-up menu is similar to the air brush pop-up menu. Press and hold the mouse button while the pointer is positioned on the paintbrush tool to open the pop-up menu.

Each of the five settings in the pop-up menu can be defined so you can have several brush shapes available without opening the Brush Settings dialog box.

To define a setting:

1. Choose the menu item you want to define from the Paintbrush pop-up menu.
2. Click the paintbrush tool again and choose Settings from the Paintbrush pop-up menu.
3. Select the brush shape you want in the Brush Settings dialog box.
4. Click Set.

The choices you make in the Brush Settings dialog box are assigned to the menu item in the pop-up and remain until you change them.

Brush Settings dialog box

The Brush Settings dialog box lets you change the shape of the paintbrush. You can double-click the paintbrush in the paint window's tool palette.

Click a dialog box option for more information:



Custom/Standard selects standard default brush shapes or lets you create your own brush shapes from the set of custom brush shapes. Only the custom brush shapes are editable.

Change a custom brush shape by clicking one of the brush shapes on the right. You can edit the current brush shape by clicking the magnified image of the brush shape. Clicking a blank pixel fills it and clicking a filled pixel makes it blank.

Clicking outside the Brush Shapes dialog box picks up the shape on the screen at the point you click.

Right/Left arrows move the brush shape one pixel to the right or to the left.

Up/Down arrows move the brush shape up or down one pixel.

Black/White square reverses the colors of the brush shape (e.g., black becomes white and white becomes black).

Copy copies the brush shapes to the clipboard.

Paste pastes the brushes into the custom set of brush shapes.

● **Pencil (paint window)**

The pencil creates a one-pixel-wide line. On a black and white cast member, the pencil draws black pixels on a white background and white pixels on a black background. On a color cast member, the pencil draws with the currently selected foreground color unless you are drawing on pixels that are the foreground color. In that case, the pencil draws in the background color.

Double-clicking the pencil tool magnifies your view of the artwork in the paint window. It magnifies the current window at the point last clicked with any of the paint tools. You can also zoom in while using the pencil or any other tool by Control-clicking.

Once you are in a magnified view you can edit the cast member pixel by pixel. You can use any of the paint tools while in a magnified view. Clicking the reduced view in the upper right corner of the paint window returns you to a 100% view. Double-clicking the pencil in the tool palette while in magnified view also returns you to a 100% view.

Move around the magnified view using the paint window's scroll bars or by using the hand tool.

- **Rectangle tool (paint window)**

The rectangle tool draws rectangles of any size. When you click the hollow rectangle tool, it draws an outline in the current foreground color as you drag the crosshair. When you click the filled rectangle tool, the rectangle you draw is filled with the current foreground and background colors in the current ink and pattern. The thickness of the rectangle's border is controlled with the line width selector at the bottom of the tool palette.

- Double-clicking the filled rectangle tool opens the **Gradient Settings** dialog box.

To constrain the rectangle to a square, press the Shift key as you drag with the crosshair pointer. If you press the Alt key while drawing a rectangle, the border is drawn with the current pattern.

- **Eraser (paint window)**

The eraser clears the portion of the cast member you drag across. The eraser always clears to white. Double-clicking the eraser tool erases everything in the paint window's visible area.

- **Ellipse tool (paint window)**

The ellipse tool creates circles and ovals. Like the rectangle tool, the ellipse is an outline if you click the hollow tool. The ellipse tool is filled with the selected foreground and background colors, ink, and pattern when you click the filled tool. The thickness of the border is controlled by clicking the line width selector at the bottom of the tool palette.

- Double-clicking the filled ellipse tool opens the **Gradient Settings** dialog box.

When you hold down the Shift key as you draw, the circle tool draws perfect circles. If you use the circle tool while pressing the Alt key, the border of the circle is drawn with the currently selected pattern.

- **Polygon tool (paint window)**

The polygon tool draws polygons with as many sides as you want. As with the rest of the shape tools, the hollow tool creates an outline and the filled tool draws an area filled with the current foreground and background colors, ink, and pattern. When you click the tool, the pointer becomes a crosshair. Click in the paint window to start drawing the side of the polygon. Each time you click, a line is drawn from the spot you clicked previously. To connect the end-point of the shape to the beginning of the shape, double-click.

The thickness of the lines drawn with the polygon tool is controlled by the line width selector at the bottom of the tool palette. If you press the Alt key while drawing a polygon, the border is drawn with the currently selected pattern.

- Double-clicking the filled polygon tool opens the **Gradient Settings** dialog box.

- **Line tool (paint window)**

The line tool draws straight lines at any angle. When you hold down the Shift key, the line tool draws vertical, horizontal, or 45-degree lines, depending upon the direction you begin to drag. Change the line width by clicking the line width selector in the tool palette.

The line is drawn with the currently selected foreground color and ink effect.

- Pressing the Alt key while drawing a line causes the line to be drawn in the currently selected pattern.

- **Arc tool (paint window)**

The arc tool draws one quarter of an ellipse or circle. When the tool is active, the pointer becomes a crosshair. Drag the crosshair from the starting point of the line and move the pointer to see the curve. Experiment with dragging the tool until it produces the line you need. The thickness of the arc is controlled with the line width selector at the bottom of the tool palette.

The line is drawn with the currently selected foreground color and ink unless you press the Alt key while dragging, in which case the arc is drawn with the current pattern.

- **Registration tool (paint window)**

The default registration point for a bitmap image is the center of the cast member in the paint window. However, the registration point for shapes, buttons, and text cast members is always the upper left corner of the image. Clicking a point in the paint window sets the registration point at that location.

To set a registration point:

1. Display the cast member you want to change in the paint window.
2. Click the registration tool .

The dotted lines in the paint window intersect at the registration point. The default registration point is the center of the cast member.

The pointer changes to a crosshair when you move it to the paint window.

3. Click a location in the paint window to set the registration point.

You can also drag the dotted lines around the window to reposition the registration point.

- To reset the default registration point at the center of the cast member, double-click the registration tool.

- **Using registration points**

- **Eyedropper (paint window)**

The eyedropper is used to match colors. When you select the eyedropper, any color you click in the paint window becomes the foreground color. Use it to match colors without opening the color palette.

• Ink pop-up menu (paint window)

The result of the ink you choose depends on whether you are working in color or black and white. Also, some inks work better when painting with patterns and others work better when painting with solid colors.

Ink (click for info)	B&W	Color	Works best with
<u>Normal</u>	<input type="checkbox"/>	<input type="checkbox"/>	Solids and patterns
<u>Transparent</u>	<input type="checkbox"/>	<input type="checkbox"/>	Patterns
<u>Reverse</u>	<input type="checkbox"/>	<input type="checkbox"/>	Solids and patterns
<u>Ghost</u>	<input type="checkbox"/>	<input type="checkbox"/>	Solids (b&w) and patterns (color)
<u>Gradient</u>	<input type="checkbox"/>	<input type="checkbox"/>	Paintbrush, paint bucket, shape tools
<u>Reveal</u>	<input type="checkbox"/>	<input type="checkbox"/>	Paintbrush, shape tools
<u>Cycle</u>		<input type="checkbox"/>	Solids and patterns
<u>Switch</u>		<input type="checkbox"/>	Paintbrush
<u>Blend</u>		<input type="checkbox"/>	Solids and patterns
<u>Darkest</u>		<input type="checkbox"/>	Patterns
<u>Lightest</u>		<input type="checkbox"/>	Patterns
<u>Darken</u>		<input type="checkbox"/>	Paintbrush
<u>Lighten</u>		<input type="checkbox"/>	Paintbrush
<u>Smooth</u>		<input type="checkbox"/>	Paintbrush
<u>Smear</u>		<input type="checkbox"/>	Paintbrush
<u>Smudge</u>		<input type="checkbox"/>	Paintbrush
<u>Spread</u>	<input type="checkbox"/>	<input type="checkbox"/>	Paintbrush
<u>Clipboard</u>	<input type="checkbox"/>	<input type="checkbox"/>	Paintbrush

- **Normal (Ink pop-up of the paint window)**

Normal is the default ink. It is opaque and maintains the color of the current foreground color and pattern.

- **Transparent (Ink pop-up of the paint window)**

Transparent ink makes the background color of patterns transparent so you can see artwork drawn previously in the current cast member through the pattern.

- **Reverse (Ink pop-up of the paint window)**

Reverse ink makes overlapping colors reverse. Any pixel in the foreground art that was originally white becomes transparent. Any pixel that was black reverses the color of the background art.

- **Ghost (Ink pop-up of the paint window)**

Ghost in black and white creates an image than can only be seen when drawn over a black background. In color, Ghost draws with the current background color.

- **Gradient (Ink pop-up of the paint window)**

Gradient lets you paint with the gradient fill selected in the **Gradient Settings** dialog box. A gradient fill is one that progresses from one color, the foreground, to another color called the destination color. You can paint with Gradient ink with the paintbrush, paint bucket, or shape tools.

- **Reveal (Ink pop-up of the paint window)**

Reveal works indirectly with the art in the previous cast position. Imagine the previous cast member's artwork covered with a white area. Reveal erases the white area to show the artwork in the previous window. Reveal can be used to create specific shapes from shades created with the air brush. Since it is impossible to mask certain shapes for the air brush, spray an area with the air brush first; then in the next cast member, paint the shapes you need with a Reveal ink. As you paint your object, you will expose the air brush pattern in the previous window.

- **Cycle (Ink pop-up of the paint window)**

Cycle is a color ink. As you draw with a cycling ink, the colors change as the ink progresses through the palette. The beginning and ending points of the color cycle are determined by the foreground and destination colors. If you want to cycle through the whole palette, choose white as the foreground color and black as the destination color.

- **Switch (Ink pop-up of the paint window)**

Switch changes any pixel that is the current foreground color to the current gradient destination color as you paint over that color.

This ink only works when your computer is set to 256 colors.

- **Blend (Ink pop-up of the paint window)**

Blend creates a translucent color ink. You can see the background object, but its color is blended with the foreground object's color. You can choose the percentage of blend in the Paint Window Preferences dialog box.

- **Darkest (Ink pop-up of the paint window)**

Darkest is a useful ink for colorizing black and white artwork. For example, if you are painting yellow over black and white, black will remain black since it is darker than yellow, and white will become yellow because yellow is darker than white.

- **Lightest (Ink pop-up of the paint window)**

Lightest is another useful ink for colorizing black and white artwork. For example, if you are painting yellow over black and white, black objects become yellow when painted with the Lightest ink effect, and white remains white because it is lighter than yellow.

- **Darken (Ink pop-up of the paint window)**

Darken makes colors darker. The more the paintbrush passes over an area, the darker it becomes. The color of the foreground, background, or destination inks have no effect on Darken. Darken creates an effect that is the same as reducing a color's brightness with the controls in the color palettes window. You can vary the rate of this ink effect in the Paint Preferences dialog box.

- **Lighten (Ink pop-up of the paint window)**

Lighten makes existing artwork lighter. The more times you pass over the artwork with the paintbrush, the lighter it becomes. The color of the foreground, background, or destination inks have no effect on Lighten. Lighten creates an effect that is the same as increasing a color's brightness with the controls in the color palettes window. You can vary the lightness of this ink effect in the Paint Preferences dialog box.

- **Smooth (Ink pop-up of the paint window)**

Smooth blurs existing artwork when painted with the paintbrush. It is not directional like Smear and Smudge. The color of the foreground, background, or destination inks have no effect on Smooth. Smooth only works with art already in the paint window. Use it to smooth out jagged edges.

- **Smear (Ink pop-up of the paint window)**

Smear works with the paintbrush and is similar to mixing paint. Any area you drag across with a Smear ink is spread in the direction of the brush and fades as it gets farther from its source. The color of the foreground, background, or destination inks have no effect on Smear. Smear only works with art already in the paint window.

- **Smudge (Ink pop-up of the paint window)**

Smudge is a color ink for the paintbrush that is similar to Smear. It is also like mixing paint. The colors fade faster as they are spread. The color of the foreground, background, or destination inks have no effect on Smudge. Smudge only works with art already in the paint window.

- **Spread (Ink pop-up of the paint window)**

Spread works with the paintbrush in color. Whatever is under the paintbrush when you start to drag is picked up as the ink for the brush. Copies of what is beneath the brush are pushed across the window as you draw.

- **Clipboard (Ink pop-up of the paint window)**

Clipboard uses the current contents of the Clipboard as a pattern to paint with.

- **Gradient destination color chip (paint window)**

A gradient is a blend of a range of colors that can be used for shading, highlights, backgrounds, and special effects. On a black-and-white monitor, gradients are created with a pattern of black and white pixels that fade from black to white or vice versa. With a color monitor, the two colors that form the beginning and end of a gradient are the foreground color and the destination color. The range of colors between the foreground and destination colors is used with the Gradient, Cycle, and Switch inks. Select the destination color using the gradient destination color chip.

To set the current foreground color, click the left side of the selector and choose a color from the pop-up color palette. To set the gradient destination color, click the right side of the selector and choose a color from the pop-up palette.

- Hold down the Alt key while pressing the up or down arrow key to cycle through the colors in the Gradient color chip.

● **Gradient (paint window)**

The gradient effect creates a blend of colors that you can use for backgrounds, highlights, shading, and special effects. Limited gradient effects can be created with a black-and-white cast member.

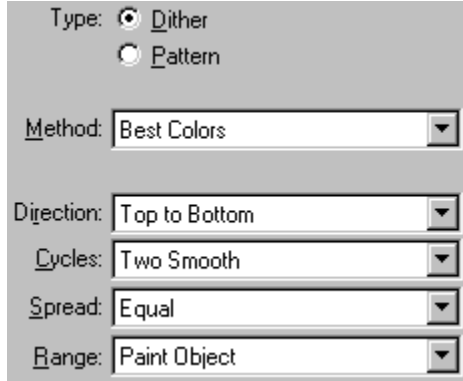
The foreground color and gradient destination color can be selected in the paint window with the destination color chip or with the controls in the Gradient Settings dialog box. When in the paint window, use the popup palette on the foreground color chip to select the foreground color. Pick the destination color with the pop-up palette at the right side of the gradient selection bar above the foreground and background color chips. The current foreground color is displayed to the left of the gradient destination color chip and the current destination color is displayed on the right side of the gradient destination color chip.

To change gradient settings, click the area between the foreground and destination color chips and choose Gradient Settings from the pop-up.

Gradient Settings dialog box

In the Gradients Settings dialog box, you can set the foreground and destination colors as well as the pattern to use with your gradient. The Gradients Settings dialog box also has several pop-up menus with drop-down lists to control the style of your gradient fill. Each choice you make is immediately previewed on the left.

Click a dialog box option for more information:



- **Type (Gradient Settings)**

This setting determines whether the gradient is made with the pattern you select with the pattern chip pop-up palette in the paint window, or with a dithered pattern. If you choose Dither, only dithering options appear on the Method pop-up below. If you choose Pattern, only pattern options appear.

● **Method (Gradient Settings)**

Method determines the way the gradient fills an area in the paint window. The options on the menu list determine where the dark and light colors of your blend are located.

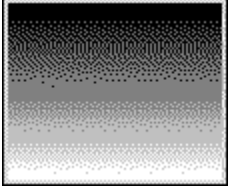
- **Pattern Best Colors** ignores the order of the colors in the palette and only uses colors that create a continuous blend of the foreground and destination colors.
- **Pattern Best Colors See Thru** ignores the order of the colors in the palette and only uses those colors that create a continuous blend of the foreground and destination colors. White pixels in patterns created with this method are transparent.
- **Pattern Adjacent Colors** uses all the colors in the palette between the foreground and destination for the gradient.
- **Pattern Adjacent Colors See Thru** uses all the colors in the palette between the foreground and destination for the gradient. White pixels in patterns created with this method are transparent.
- **Dither Best Colors** ignores the order of the colors in the palette and only uses colors that create a continuous blend from foreground to destination colors and blends them with a dithered pattern. Dithering is a technique of creating color with two or more colors of pixels interspersed together.
- **Dither Adjacent Colors** uses all colors between the foreground and destination colors and blends them with a dithered pattern.
- **Dither Two Colors** uses only the foreground and the destination colors and blends them with a dithered pattern.
- **Dither One Color** uses only the foreground color and fades it with a dithered pattern.
- **Standard Dither** ignores all colors between foreground and destination and adds several blended colors with a dithered pattern to create the gradient.
- **Multi Dither** ignores all the colors between foreground and destination and adds several blended colors with a randomized dithered pattern to create a smooth gradient. You can interrupt the drawing of this kind of dither by clicking anywhere in the dialog box.

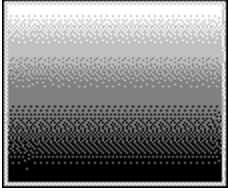
● **Direction (Gradient Settings)**

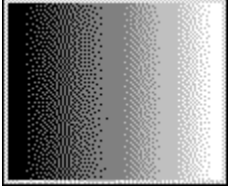
The Direction popup menu list determines the way the gradient fills an area in the paint window. The options on the menu list determine where the dark and light colors of your blend are located.

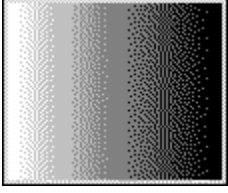
Click the name of a gradient direction to see an illustration:

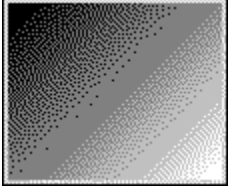
- **Top to Bottom** puts the foreground color at the top and the destination color at the bottom
- **Bottom to Top** puts the destination color at the top and the foreground color at the bottom.
- **Left to Right** puts the foreground color on the left and the destination color on the right.
- **Right to Left** puts the foreground color on the right and the destination color on the left.
- **Directional** you determine the direction of the gradient. You set the direction of the gradient in the paint window with the paint tool used to fill the area.
- **Sun Burst** starts filling at the edge of the artwork and moves in concentric circles to the center.

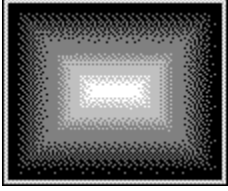


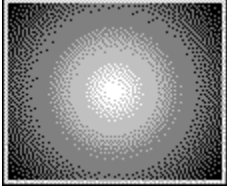










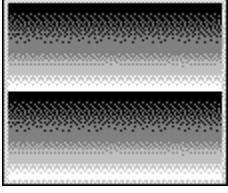


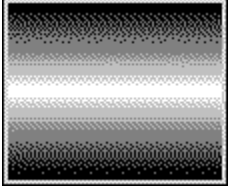
● **Cycles (Gradient Settings)**

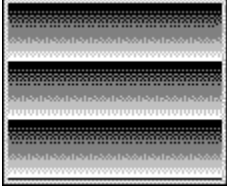
Cycles control the number of times the gradient is created within one filled area, and whether or not the colors cycle through the palette in one direction only, or auto reverse at the end of one pass through the palette. Sharp cycles have a banded appearance, while smooth cycles go from foreground to destination, then back to foreground.

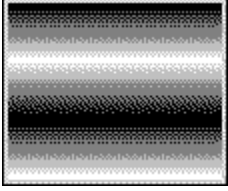
Click the name of a gradient cycle to see an illustration:

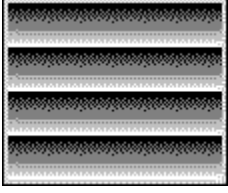
- **One Cycle** takes the gradient once through the range of colors you define.
- **Two Sharp** takes the gradient through the range of colors twice, from foreground to destination and from foreground to destination.
- **Two Smooth** takes the gradient from foreground to destination, then from destination to foreground.
- **Three Sharp** takes the gradient from foreground to destination three times.
- **Three Smooth** takes the gradient from foreground to destination, destination to foreground, foreground to destination.
- **Four Sharp** takes the gradient from foreground to destination four times.
- **Four Smooth** takes the gradient from foreground to destination, destination to foreground, foreground to destination, and destination to foreground.

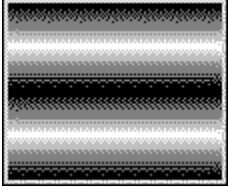












- **Spread (Gradient Settings)**

Spread options let you choose how to distribute colors between the foreground and the destination colors of the gradient.

- **Equal** provides an even spacing of colors between the foreground and the destination colors.
- **More Foreground** increases the amount of the foreground color in the gradient.
- **More Middle** increases the amount of the middle color in the gradient.
- **More Destination** increases the amount of the destination color in the gradient.

- **Range (Gradient Settings)**

Range options determine whether the full range of the gradient is created over the paint object, cast member, or the entire paint window. The options provide greater control over how the gradient is created relative to the cast member's position on the stage or in the paint window.

- **Paint Object** paints the full gradient as the fill or brush stroke of the object, regardless of the object's location in the paint window.
- **Cast Member** paints the full gradient with respect to the size of the cast member.
- **Window** paints a full gradient only if the object is the length or width of the entire window, otherwise it paints a partial gradient corresponding to the object's location in the window.

- **Foreground color chip (paint window)**

The foreground color is the color displayed in the foreground color chip. It's also displayed in the color chip on the left side of the gradient color selector. The foreground color is the color you work with when you're using the solid pattern solid and the Normal ink effect.

Press the up or down arrow key to cycle through the colors in the color palette associated with the Foreground color chip.

Double-click the foreground, background, or destination color chip to open the **color palettes window**.

- **Background color chip (paint window)**

The background color is the color displayed in the lower right color chip. The background color is the secondary color that appears in a pattern. When used with the Transparent ink, the background color in a pattern is transparent.

- Hold down the Shift key while pressing the up or down arrow keys to cycle through the colors in the color palette associated with the Background color chip.

- **Pattern chip (paint window)**

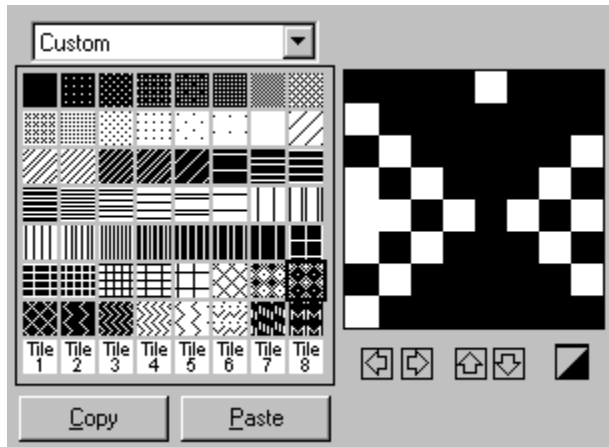
The current pattern is displayed in the pattern chip below the two color chips in the tool palette.

Click the pattern chip to select a new pattern from the pop-up palette.

-
- Pressing the Alt key before displaying the pattern palette permanently changes the patterns to shades ranging from the foreground color to the background color rather than the set of patterns you see without pressing the Alt key. Press the Alt key again to return to the default set of patterns.
- Double-click the pattern chip to open the **Pattern Settings** dialog box. Use the dialog box to edit or select new sets of patterns.

● Pattern Settings

Click part of the illustration for more information:



Custom/Standard selects from the standard default patterns and permits you to create your own patterns from the set of custom patterns. Only the custom patterns are editable.

Change a custom pattern by clicking one of the patterns on the left. You can edit the current pattern by clicking the magnified image of the pattern. Clicking a blank pixel fills it and clicking a filled pixel makes it blank.

Right/Left arrows move the pattern one pixel to the right or to the left.

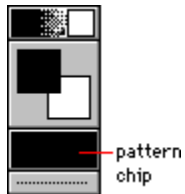
Up/Down arrows move the pattern up or down one pixel.

Black/White square reverses the colors of the pattern (e.g., black becomes white and white becomes black).

Copy/Paste copies or pastes the pattern you want to use.

• **Tile Settings (paint window)**

Tiles are a useful way to create patterns with more than two colors. Cast members in the paint window form the basis for creating a tile. When you choose a portion of a cast member to be made into a tile, the cast member becomes a building block for a pattern created with a field of tiles. You can use tiles in the paint window or with the shapes in the tools window.

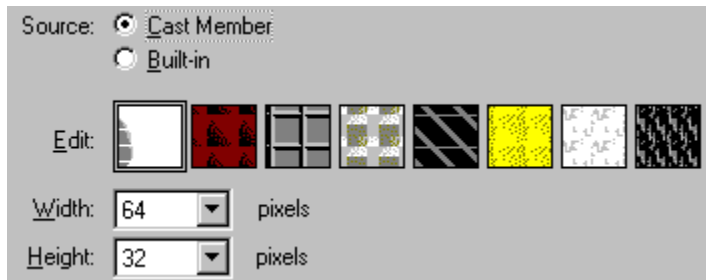


To open the Tile Settings dialog box, click the Pattern chip and choose Tile Settings from the pop-up.

Dialog box options

The Tiles dialog box lets you select the tile position, the cast member to make into a tile, and the tile size.

Click a dialog box option for more information:



The selection rectangle in the cast member box at the upper left of the Tiles dialog box determines which part of the cast member is used for the tile. Drag the rectangle to reposition it on a different part of the cast member or click a new spot in the cast member view to reposition the dotted rectangle.

Source controls which cast member is the basis for your tile. Click Built In to indicate that the selected tile is one of the default tiles. Click the Cast Member if you want to make a tile from a cast member in your movie. When you click the Cast Member radio button, left and right arrows appear that permit you to step through all the graphic cast members in your movie.

Edit lets you select the tile you want to edit. An enlarged version of the tile is displayed in the dialog box.

Width, Height lists the available tile sizes in pixels. You can make a tile as small as 16 by 16 pixels or as large as 128 by 128 pixels. As the size of the tile increases, more of the cast member is used for the tile.

- **Line width selector (paint window)**

The line width selector controls the thickness of the line drawn by the line or arc tool and the thickness of the borders drawn by the shape tools.

The width of the line drawn by the line, arc, rectangle, ellipse, and polygon tools can be changed with the line width palette. The line width palette has several settings for line width ranging from No Line (the dotted line in the line selector) to Other. Use the dotted line setting when you want to draw filled shapes without borders. If you choose Other, the line width is determined by the Other Line Width setting in the **Paint Preferences** dialog box.

- Double-click the line width selector to open the Paint Preferences dialog box. Use it to set the Other Line Width.

• **Color resolution indicator (paint window)**

The color resolution indicator displays the color resolution of the current cast member in the paint window.

Double-click the color resolution indicator to open the **Transform Bitmap** dialog box. Use the Transform Bitmap dialog box to change the color resolution of the current cast member in the paint window. Changing the color resolution from color to black and white saves disk space. You can still make a selected 1-bit sprite a color other than black by selecting colors with the foreground and background color chips in the tool palette after it has been reduced to 1-bit. (This colorizes the sprite on the stage, but does not affect the original cast member, which remains black and white.)

If you import black and white cast members, changing their color resolution to multiple colors permits you to colorize them with any color in the current palette.

● **Paint toolbar**

Paint toolbar buttons provide effects that modify bitmapped cast members.

Click toolbar buttons for a description:



Flip Horizontal

Flip Vertical

Rotate Left

Rotate Right

Rotate Free

Skew

Warp

Perspective

Smooth

Trace Edges

Invert Colors

Lighten

Darken

Fill

Replace

Repeat Effect

Flip Horizontal

Flip Horizontal mirrors the selected area in the paint window horizontally from right to left.

Flip Vertical

Flip Vertical mirrors the selected area in the paint window vertically from top to bottom.

Rotate Left

Rotate Left rotates the selected area in the paint window 90 degrees counterclockwise.

Rotate Right

Rotate Right rotates the selected area in the paint window 90 degrees clockwise.

Free Rotate

Free Rotate allows you to rotate the selected area in the paint window any number of degrees clockwise or counterclockwise. When you choose Free Rotate, handles appear at the corners of the selection rectangle. To rotate the artwork, drag any handle in the desired direction.

The Free Rotate command is one of four special effects you can apply to artwork selected with the selection rectangle in the paint window. The other special effects are Perspective, Slant, and Distort. When you choose one of the special effects after selecting the artwork, handles appear at each corner of the selection. Dragging the handle produces the desired effect.

Skew

The Slant command skews the selected artwork. When you choose Slant, handles appear at the corners of the selection rectangle. Dragging a handle in the desired direction moves the opposing corner an equal amount in the same direction, maintaining a parallelogram shape.

Warp

When you choose Distort, handles appear at the corners of the selection rectangle. Each corner can be dragged in any direction independent of the other corners. When you release the mouse button, the selected artwork assumes the shape that you have created.

Perspective

Perspective stretches the selected artwork to give it a perspective effect. When you choose Perspective, handles appear at the corners of the selection rectangle. Drag one or more handles to create the effect you want. For example, you can bring the two top handles closer together to create the illusion of linear perspective.

- To make artwork appear to be vanishing into the distance, choose Perspective and move the handles on one side of your selection together and the handles on the opposite side of your selection apart.

Smooth

Smooth softens the edges of the selected artwork by adding pixels of blended color to the artwork's edges.

Trace Edges

Trace Edges creates an outline around the edges of the selected artwork. The outline is the same color as the selected line, if the line is a solid color. If the original line is multicolored, an outline is created for each section of the line. You can add multiple outlines by clicking Trace Edges repeatedly.

Invert Colors

The Invert Colors command reverses the colors of the selected area in the paint window. For 2-, 4-, and 8-bit cast members, to see what the reverse colors are, open the color palettes window, select all the colors in the palette, and click Reverse Color Order in the Color Palettes window; you'll see that the effect is an upside-down mirror image of the palette. For 16- and 32-bit cast members, a true RGB-complement of each color is shown. If you are working in black and white, Invert Colors changes black to white and vice versa.

Lighten

Lighten increases the brightness of anything in the selection rectangle

Darken

Darken reduces the brightness of the selected artwork.

Fill

The Fill command fills a selected area with the current foreground color and pattern.

Replace Color

Replace Color changes each pixel that is the currently selected foreground color to the currently selected destination color.

Note: This command only works for images whose color depth is 8-bits or less.

Repeat Effect

Control+Y

Repeat Effect repeats the last color effect command chosen from the Modify menu for the selected area.

- **Text window (Window menu) **Control+6****

Use the text window to create and edit text cast members. Enter and edit text in the text window using standard word processing procedures. You select text in the text window or on the stage by dragging across the text, or by double-clicking to select a whole word. Triple-clicking selects all text in the cast member, which is the same as choosing Select All from the Edit menu.

The ruler and toolbar across the top of the text window provide several formatting shortcuts.

To open the text window:

- Choose Text from the Window menu.
- Click the text window tool on the toolbar.
- Double-click a text cast member in the cast or on the stage.

Click a topic for more information:

- - [Creating text cast members](#)
 - [Understanding text and fields](#)
 - [Editing text on stage](#)
 - [Importing text](#)
 - [Text ruler](#)
 - [Text Inspector](#)
 - [Applying color to text](#)
 - [Paragraph command \(Modify menu\)](#)
 - [Font command \(Modify menu\)](#)
 - [Editable text \(Field window\)](#)

- To create another view of the same text window, click New Window in the Windows menu. This is useful if you are editing a large text cast member, since you can display different sections of the text in each view and cut and paste between them.

• Text ruler

Use the text ruler to set tabs and indents for paragraphs. To show or hide the text ruler, choose Ruler from the View menu. To set units of measure for the ruler, choose **General Preferences** from the File menu.



Setting tabs

- To set tabs, click the tab well until the symbol for the type of tab you want is displayed and then click the ruler where you want to place the tab.
 - └ Left
 - └ Right
 - └ Center
 - └ Decimal
- To remove a tab, drag the tab up or down off of the ruler.
- To move a tab, drag the tab to the new location on the ruler.
- If you don't define your own tab stops, pressing the Tab key advances the cursor to the next preset tab.

Setting indents

To set an indent for selected paragraphs, drag the left and right indent markers on the ruler. To change the indent for only the first line of text, drag the marker that points down from the top of the ruler. Setting a special first line indent is useful for creating bulleted paragraphs and "hanging indents."

- **Field window (Window menu)** **Control+8**

Use the field window to enter and edit text for field cast members. It is identical to the text window except that the ruler is not available for setting tabs and indents. When you use the alignment and spacing tools on toolbar, the changes affect all the paragraphs in the cast member.

To open the field window:

- Choose Field from the Window menu.
- Double-click a field cast member.

- **Understanding text and fields**
Field Cast Member Properties

● **Color palettes window (Window menu)** **Control+Alt+7**

The color palettes window provides several ways of changing color palettes.

Click a topic name for more information:



● **Reserve selected colors**



● **Select reserved colors**



● **Select used colors**



● **Invert selection**



● **Sort colors**



● **Reverse sequence**



● **Cycle colors**



● **In-Between colors**



● **Color picker**

All the functions in the color palettes window involve changing the currently active color palette. You choose the active palette by selecting a palette from the pop-up.

Director has ten built-in palettes:

- System--Mac (the standard 256-color Macintosh system palette)
- System--Win (the standard 256-color Windows system palette)
- Rainbow palette
- Grayscale palette
- Pastels palette
- Vivid palette
- NTSC palette
- Metallic palette
- VGA palette, a special palette for VGA 4-bit displays. It provides consistent results when playing Director movies under Windows in 4-bit mode.
- System-Win (Dir4), a palette that is included for compatibility with movies created in Director 4.

If you add new palettes to your movie from other graphics applications, those palettes also appear in the pop-up and in the cast.

In Windows, if you use any palette other than the two System-Win palettes (or a palette that does not include the Windows colors in its top and bottom rows), Director switches to Classic user interface look, and switches Windows itself to black and white. This is to maintain legibility while editing your movies.

Note: Choosing a new palette in the Color Palettes window does not change the palette

for the movie, or any frame in the movie. Use **Movie Properties** on the Modify menu to choose the movie's default color palette, or **Frame Palette** on the Modify menu to change the color palette at a particular frame.



Use the hand tool to drag colors in the palette to reposition them.



Use the eyedropper to match the color of any pixel on the stage with the same color in the palette. Click the eyedropper tool and then move the pointer to the stage. When you click, the color matching the pixel you clicked is selected in the color palette.



Clicking the arrow changes your pointer back to the arrow pointer.

- **Understanding color palettes**
Editing colors

-
- **Reserve selected colors**

Reserve selected colors in the color palettes window isolates specific colors used in palette effects like color cycling. For example, if you are cycling colors and don't want to inadvertently use the cycling colors on a noncycling cast member, reserve the cycling colors to prevent them from being used.

To reserve colors, select them in the color palettes window and then click the reserve selected colors button.

When the Reserve Selected Colors dialog box appears, choose Reserve Currently Selected Colors and then click OK.

- Reserved colors appear striped in the color palettes window. The Select Reserved button (shown at the left) also appears in the window to help you see which colors are reserved.

Click the Select Reserved button to select all reserved colors.

To make all the reserved colors available again, click the reserve selected colors button and choose All Colors Available in the dialog box.

-
- **Select reserved colors**

Select reserved colors in the color palettes window highlights the colors in the current palette that have been reserved.

-
- **Select used colors**

Select used colors in the color palettes window highlights the colors in the current palette used in selected cast members.

-
- **Invert selection**

Invert selection in the color palettes window replaces the color or range of colors you selected with a new selection. The new selection consists of all the colors that were not part of your original selection.

-
- **Sort Colors**

The Sort button in the color palettes window reorders the selected colors in the palette by Hue, Saturation, or Brightness.

To use the sort button, select a range of colors in the color palettes window and then click the button. When the Sort Colors dialog box appears, choose either Hue, Saturation, or Brightness. When you click OK, the colors appear in the palette according to the sorting order selected.

If you sort colors after drawing cast members, the cast members that use the selected colors will also change color as the colors are sorted.

-
- **Reverse sequence**

The reverse sequence button in the color palettes window reverses the order of the selected colors: the first color of the palette becomes the last. The colors themselves remain unchanged.



Cycle colors

The Cycle colors button displaces all the selected colors in the color palettes window one square to the left. The leftmost color wraps around and appears at the last right square. Each time you click the cycle button, the selected colors shift by one more square. As the colors reach the left edge of the selection, they wrap around to the right edge and continue their journey. It is similar to the movement of color within a palette that you can see while colors cycle.

This is precisely what goes on when you use color cycling. If you are using a paint tool in the paint window with a cycling ink effect, the colors rotate through the palette as you draw. Another example of color cycling is in the **Frame Palette** dialog box in the Modify menu. You can select a range of colors to cycle as your movie plays. Any cast member that is the same color as one of the cycled colors will change as the colors rotate through the palette.

-
- **In-Between colors Control+B**

The In-Between colors button changes the current palette to create a blend of the first and last colors of a selected range in the color palettes window. Use it to create blends of color for color cycling or smooth gradients.

To use the In-Between color button, select a range of colors in the color palettes window and then click the button.

If you choose one of the nine built-in palettes, Director creates a new palette. A prompt appears for you to name the new palette.



Color picker

You can define a new color in the current color palette either with the controls at the bottom of the color palettes window, or with the Windows Color dialog box.

To use the Windows Color dialog box, select the color you want to change and then click the set color button. For information about using the Color dialog box, see the user's guide that came with Windows.

To edit selected colors in the color palettes window using the HSB (Hue, Saturation, Brightness) system, click the arrows at the bottom of the window to increase or decrease the value of hue, saturation, or brightness.

Hue is the primary or secondary color created by mixing two primaries. **Saturation** is a measure of how much white is mixed in with the color. A fully saturated color is vivid; a less saturated color is a washed out pastel or even a shade of gray.

Brightness controls how much black is mixed in with a color. Colors that are very bright have little or no black. As the brightness is reduced, the color gets darker as if more black were added. If brightness is reduced to 0, then no matter what the values for hue or saturation, the color will be black.

- **Video window (Window menu)** **Control+9**

The Video Window lets you play digital video movies. Use the controls at the bottom of the window to play, stop, advance, or rewind the movie. When the movie is stopped, you can cut, copy, and paste frames from the movie into another digital video window.



Choose Video from the Window menu, or double-click a digital video cast member in the cast window or on the stage to display the video window.

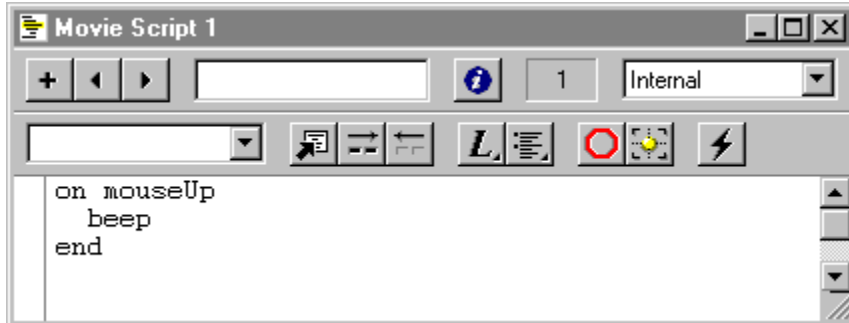
Note: On a Macintosh, the term "digital video" refers only to QuickTime movies. However, Director for Windows supports Microsoft's Video for Windows (.AVI) and QuickTime for Windows.

The **Place**, **Previous**, **Next**, **Info**, and **Script** buttons work the same as they do in the cast window.

- **Creating a digital video cast member**
Using multiple video
Digital Video Cast Member Properties

- **Script window (Window menu) **Control+0****

Use the script window to enter and edit Lingo scripts. A script can contain up to 32K of text.



For a description of the buttons at the top of the script window, see the **cast window** topic.

For descriptions of the tools on the script toolbar, see the **Script toolbar** topic.

- The help topics for **Lingo** commands include examples that you can paste into your scripts and use.

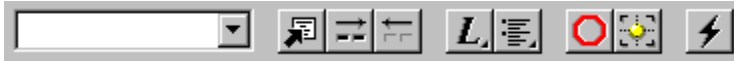
Director saves changes you make in the script window when you click anywhere outside of the window, close it, click the Previous or Next buttons to go to a different script, or if you choose Recompile All Scripts in the Control menu.

- Double-clicking a cell in the script channel opens the script window.

- **Script Cast Member Properties**
Debugger
Message
Watcher

- **Script window toolbar**

The script window toolbar has the following tools (click one for more information):



Handler name pop-up

Lists the name of each handler that is used in the current movie. Go to a handler by selecting its name from the pop-up.

Go to Handler

Goes to the handler that is in the current line of Lingo and inserts the cursor there.

Comment

If the current line of Lingo is not commented out, this command comments out the line by putting two dashes at the beginning of the line.

Uncomment

If the current line of Lingo is commented out, this command uncomments the line by removing the two dashes at the beginning of the line.

Alphabetical Lingo menu

Displays an alphabetical menu of all Lingo elements. Choosing one of the elements from the menu inserts it in the script at the cursor.

Categorized Lingo menu

Displays a menu of Lingo elements grouped according to the features that they can implement. Choosing one of the elements from the menu inserts it in the script at the cursor.

Toggle Breakpoint

Inserts and removes breakpoints in the current line of Lingo.

- When the current line of Lingo has a breakpoint, clicking Toggle breakpoint removes it.
- When the current line of Lingo has no breakpoint, clicking Toggle breakpoint inserts one.

Watch Expression

Adds the selected expression or variable to the list in the watcher window.

Recompile

Recompiles the movie's scripts without closing the script window.

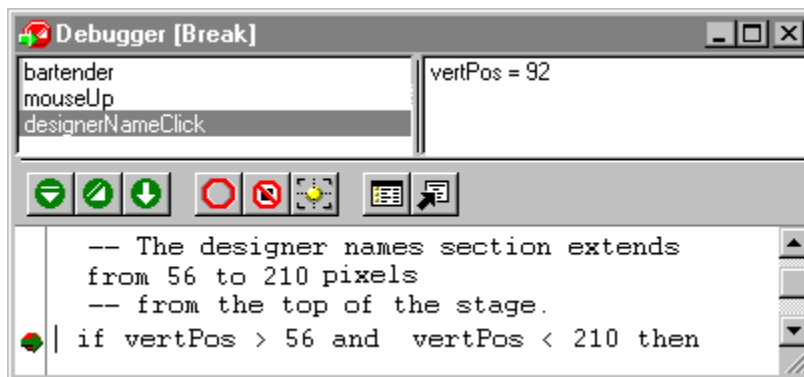
- **Debugger window (Window menu)**

Control+`

The debugger helps with troubleshooting. The window helps to locate and correct bugs in Lingo scripts. It includes several tools that let you:

- See the current line of Lingo
- Run the current handler line by line
- Track the sequence of handlers that were called as part of getting to the current handler
- Display the value of any local variable, global variable, or property related to the Lingo that you're investigating
- Open related windows such as the watcher window and script window.

Click part of the illustration for more information:



Any of the following actions opens the debugger window:

- Choosing Debugger from the Window menu
- Encountering a breakpoint in a script
- Clicking Debugger in an error message that appears when Director encounters a syntax error in a script.

You can't edit the script directly in the debugger; you must return to the script window.

- **Script
Message
Watcher**

Handlers history pane displays the order of handlers that ran to get to the current script. Clicking a handler name displays the script. Using the Step Into or Step Over button always focuses the debugger back to the bottom-most active handler in the pane.

Variable pane rapidly displays local variables, global variables, and property settings that Lingo encountered as it ran up to the current line. Only values are displayed; you can't change them directly in the debugger. To edit Lingo, return to the script window. To change any of these values while working with the debugger, use a `set` or `put` statement in the message window.

Script pane displays the current script and highlights the current line of Lingo. Breakpoints are indicated by a red dot to the left of the line of Lingo. The current line is indicated by a green arrow to the left of the line. The Lingo that appears in the script pane is only a display. You must return to the script window to edit the script.

Step script runs the current line of Lingo, runs any handlers that the line calls, and then stops at the next line in the current handler. This is useful when you are confident that handlers called from the current line are performing as expected and want to concentrate on Lingo in the current handler.

Step into script follows Lingo's normal flow, line by line from the current line from the current line through any nested handlers. Lingo advances one line each time you click Step into script. This is useful when you want to examine the handlers called from the current line of Lingo.

Run Script exits debugging and returns to the current handlers.

Toggle Breakpoint turns the breakpoint in the line of Lingo that is highlighted in the script display pane on and off. When the highlighted line has no breakpoint, clicking Toggle Breakpoint inserts one at that line. When the highlighted line has a breakpoint, clicking Toggle Breakpoint removes the breakpoint.

Ignore Breakpoints has Lingo pass over any breakpoints in the movie's scripts.

Watch Expression adds the selected variables or expressions to the watcher window.

Watcher window opens the watcher window, which displays the current value of variables that you select. For more information about the watcher, See [Watcher window](#).

Go to Handler goes to the handler in the script window whose name is selected in the debugger. After you are in the script window, you can edit the script normally.


• **Watcher window (Window menu)** **Control+Shift+`**

The Watcher window shows the values of simple expressions and variables in the movie's scripts.

The variable or expression appears at the left of the window followed by an equal sign (=) and the expression or variable's current value. If Director can't obtain the value of an expression or variable in the current context, the term "<void>" appears to the right of the equal sign. Director updates values in the watcher window when the user steps through lines of a script while in debug mode or continuously while the movie plays. Some variables, such as `the time` or `the mouseH` are updated even while the movie is not playing.

Note: Watching too many variables can decrease Director's response noticeably. For example, the cursor may blink slowly or windows may resize sluggishly. Reducing the amount of data the watcher window must continuously update will immediately improve Director's performance level.

To add variables or expressions to the list in the watcher window by selecting them in the script window:

1. Open a script window in which the variable or expression appears.
2. Select the variable or expression.
3. Click the Watch Expression button 

at the top of the script window.

Director adds the selected variables or expressions to the list in the watcher window and also displays those changes in the variable pane.

To change the value of a variable or expression:

1. Select the value or expression in the watcher window.
2. Type the new value in the field next to the Set button.
3. Click Set.

To add variables or expressions to the list directly from the watcher window:

1. Type the variable or expression in the field to the left of the Add button.
2. Click Add.

The variable or expression appears in the list.

To remove a variable or expression:

1. Select the variable or expression in the watcher window.
2. Click Remove.

• **Debugger** **Message** **Script**

- **Message window (Window menu)** **Control+M**



The message window is a convenient place to experiment with and test Lingo scripts. Actions occur immediately when you press the Enter key, so you can see the results before you insert your scripts into a movie. This allows you to see the results of any script, including whether it is a valid script.

For descriptions of the tools on the message toolbar, see the **Message toolbar** topic.

To move around the message window, use the arrow keys or scrollbar. Press Control+up arrow to move the insertion point to the top of the window. Press Control+down arrow to move the insertion point to the bottom of the window.

- **Debugger**
Script
Watcher

- **Message window toolbar**

The message window toolbar has the following tools (click one for more information):



Alphabetical Lingo menu

Displays an alphabetical menu of all Lingo elements. Choosing one of the elements from the menu inserts it in the script at the cursor.

Categorized Lingo menu

Displays a menu of Lingo elements grouped according to the features that they can implement. Choosing one of the elements from the menu inserts it in the script at the cursor.

Trace

Click the Trace button to have the message window display all the Lingo that runs as the movie plays. Using Trace slows down animation, so click Trace to turn it off.

Go to Handler

Goes to the handler that is in the current line of Lingo and inserts the cursor there.

Watch Expression

Adds the selected expression or variable to the list in the watcher window.

