22

CHAPTER

Menus and Dialog Boxes

In This Chapter

Learn to create and manipulate menus

Build simple dialog boxes with the MUI Xtra

Build not-so-simple dialog boxes with the MUI Xtra

By nature, Director encourages a free-form interface design: buttons that come alive when the mouse rolls over them, animations that dance across the screen, whole toolbars that appear and disappear effortlessly. However, even in the most dynamic environment, there is a place for menu bars and dialog boxes — the staples of more traditional programs on any platform. This chapter explores the creation of menus and dialog boxes by means of Lingo.

Making Menus

In the history of computing, the menu bar was a major step forward. It presented options to the user that previously had been available only by using terse (and frequently poorly documented) command-line arguments. The menu bar evolved into powerful and intuitive graphical user interfaces such as those of the Macintosh operating system (OS) and Microsoft Windows. Today's menu interface is vastly more complex than the first menus — some say too complex — but there is no denying the importance of the fundamental principles that make menus useful. Those principles are:

- Menus group commands by functionality, providing a quick and logical path to performing a specific action.
- Similar common menus make it easier to work with several programs running on the same operating system. The beginning user doesn't have to learn a completely different paradigm when exploring every new application. An application's interface is typically more similar to than dissimilar from those of other applications using the same operating system.

✦ Menus act as reminders for common keyboard equivalents. When the shortcut operation to save a file is to press Command+S (or Ctrl+S in Windows), that shortcut is usually displayed when you choose File ➪ Save. Until a user becomes familiar with the keyboard shortcuts, menus provide easy access to all features and list the shortcuts while remaining unobtrusive.

✦ Menus frequently serve as a gateway to dialog boxes in a manner that would be difficult to implement with buttons and other graphical elements. That's because each entry in a menu portrays an instant explanation of the action it is to perform (at least it should, although in poorly designed menus, this isn't always true).

✦ Menus are easy to localize, unlike bitmap graphics that need to be recreated every time the language changes. By switching a cast member, you can make a Director menu change the language it displays.

Tip

Menus are most useful to beginners—people who aren't familiar with the product. As users become more proficient, they tend to rely more on shortcut keys or, if they're available, button bar icons. Remember this as you work on your own design. When you're putting together a menu, if you aim it toward the novice, you can't go wrong.

Creating Your Own Pull-Down Menu System

You might not yet have given much thought to including a pull-down or drop-down menu in your Director movie, but you won't want to neglect this feature. Most off-the-shelf applications include some sort of pull-down menu system. In fact, menus are so commonplace that nobody even thinks about them — until they're not available.

Fortunately, menu systems are easy to implement in Director. You can build a menu system to handle tasks such as these:

✦ Executing commands and access features

✦ Navigating within the application

✦ Controlling system settings (such as audio volume)

Building a pull-down menu requires a few easy steps:

1. Create a field cast member that includes specific keywords that construct the menu and its options.
2. Install the menu in the movie by using the `installMenu` command.
3. Create a movie projector.
4. Play and test the projector.

Creating a menu by using a field cast member

Menus are created by using a field cast member that begins with the menu keyword followed by the name of the menu. To create a File menu like the one found in nearly every application, for example, you type the following text as the first line in a field:

```
menu: File
```

On subsequent lines in the field, you type the options that you want to appear on the menu. You add options to the menu by using the following syntax:

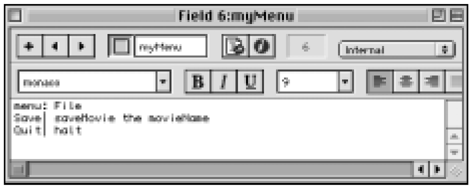
```
menuItem | LingoInstruction
```

The vertical bar (|) character (known as a *pipe* in most programming languages) separates the text of the menu option from the Lingo instruction that is executed when the option is selected by the user. On most keyboards, the pipe or vertical bar character is generated by pressing Shift+\.

Director does not support a direct method of executing multiple commands from a single menu item on a user-defined menu. However, you can create a workaround. To implement this arrangement on the user-defined menu, create a menu option to call a user-defined handler that you construct with the desired instructions.

On the CD-ROM The movie mars scanner 1.dir is in the EXERCISE:CH22 (EXERCISE\CH22) folder on this book's companion CD-ROM.

A typical File menu (on a Windows or Macintosh system) includes a Save option and a Quit (or Exit) option. Let's see how to add these options to the File menu. Open the movie mars scanner 1.dir on the CD-ROM. Create a new field member and the text shown in Figure 22-1.



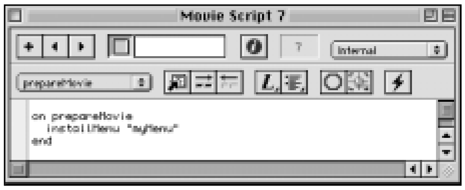
```
menu: File
Save| saveMovie the movieName
Quit| halt
```

Figure 22-1: Use field members to create menus, not text cast members.

Caution A space before and after the vertical bar character is optional. If you do add spaces, those spaces will be part of the name for that menu item. Also, adding spaces becomes an issue when you are referring to these menu items from Lingo, because you must use the exact name.

In this example, the `saveMovie` command saves the current movie, using the `movieName` property. When the Quit menu item is chosen, the Lingo command `halt` executes. In a projector, the `halt` command causes the program to quit, but when authoring, it just stops the movie. Lingo does have a `quit` command, but if you use that command in authoring mode, it doesn't just stop the movie, it causes Director to quit (which can be inconvenient).

For this menu to be used in the program, you need to use the `installMenu` command. Because we want this menu to appear as soon as the movie runs, a good place for the code to insert it is in a `prepareMovie` handler. Create a movie script and type the code shown in Figure 22-2.



```
on prepareMovie
  installMenu "myMenu"
end
```

Figure 22-2: Using the installMenu command

Figure 22-3 demonstrates how this menu appears when the movie is played back in authoring mode. The menu appears in the upper-left corner of the screen, not on the Stage area. Save this movie as **mars scanner 2.dir**.

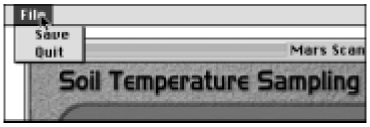


Figure 22-3: The custom File menu as it appears in authoring mode

Tip

When you add menus and menu options, be sure to use the capitalization that you want displayed. Capitalization for the Lingo instructions is not important, because Lingo is not case sensitive. For the menus and their contents, however, what you type is what is displayed on-screen.

In Windows, graphics and other objects on the Stage are shifted down because the menu bar occupies a 24-pixel band across the top of the Stage, as shown in Figure 22-4. On the Mac, the menu bar goes right over the Stage graphics. This functionality is irritating and did not exist in earlier versions of Director. It makes cross-platform development more difficult, because each platform displays menu bars in a different manner. We hope that Macromedia will change this in a future version.

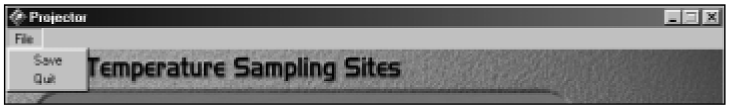


Figure 22-4: A Windows pull-down menu as it appears during playback from a projector

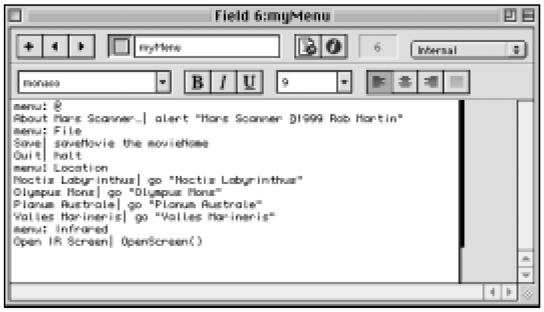
You need to keep this in mind when you add a menu to a movie. To avoid screen layout problems, you must:

- ✦ Shift everything on the Stage down 24 pixels from where you normally place the sprites.
- ✦ Leave the bottom 24-pixel band across the screen blank.

Note This downward shift of objects on the Stage does not increase the Stage size; it just pushes the bottom 24-pixel band into never-never land.

You can display more than one pull-down menu at a time on the Stage. To do so, you simply define the first menu (as just described), and then sequentially define each additional menu in the same field cast member. Each new menu and each new menu option within the field must start on a new line.

Open the mars scanner 2.dir movie from the EXERCISE:CH22 (EXERCISE\CH22) folder on the CD-ROM. Open member 6, which is the field "`myMenu`". Add the text shown in Figure 22-5. The field cast member creates a menu bar with four command menus (@, File, Location, Infrared). The File menu part is already there. If you are a Mac user, add the first two lines before the `menu: File` line (the `menu: @` and `About Mars Scanner` lines); Windows users should leave those off.



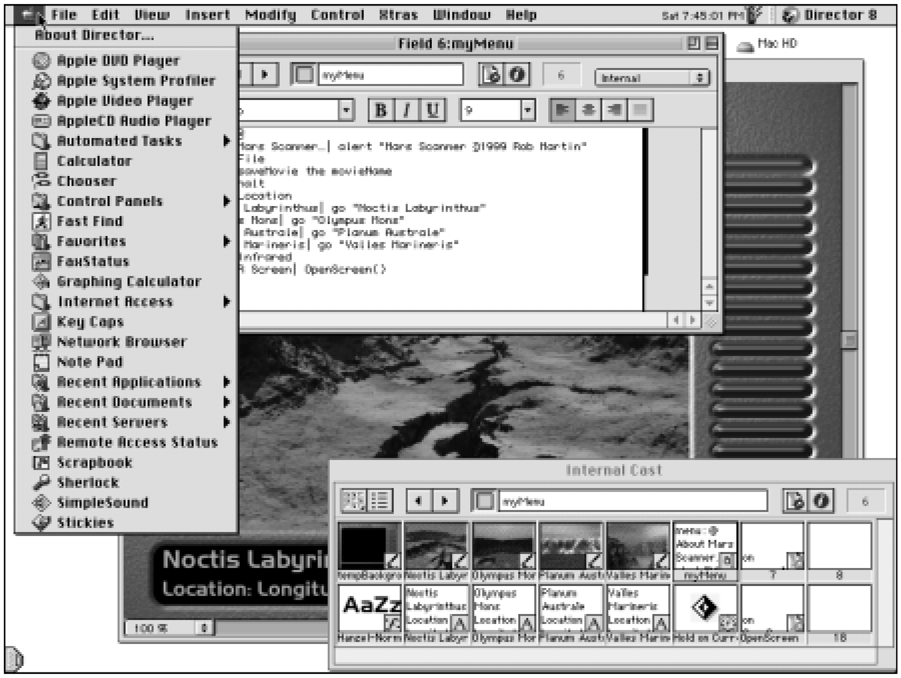
```
menu: @
About Mars Scanner…| alert "Mars Scanner ©1999 Rob Martin"
menu: File
Save| saveMovie the movieName
Quit| halt
menu: Location
Noctis Labyrinthus| go "Noctis Labyrinthus"
Olympus Mons| go "Olympus Mons"
Planum Australe| go "Planum Australe"
Valles Marineris| go "Valles Marineris"
menu: Infrared
Open IR Screen| OpenScreen()
```

Figure 22-5: Defining the menus for a program

This `menu: @` instruction only adds the Apple menu, not any options. You must create the desired options just as you would for any other menu that you want included in your movie. By convention, the Apple menu is always the first menu on the menu bar (starting at the upper left edge of the screen), so the `menu: @` instruction (followed by the options you want available) should always be listed as the first menu entry in the field cast member. If you leave this menu off, then the Apple and Application menus will not be available.

The instructions for the menus are stored in the same field, one after the other. If you put a RETURN character between menus, it will appear as a blank menu item.

The menu `Location` in Figure 22-5 has four menu items. When one is selected, the Lingo to the right of the bar is executed. The `Infrared` menu has one item. The Lingo it executes is a handler in a movie script. Calling a handler enables you to execute several lines of Lingo. Save this movie as **mars scanner 3.dir**. Figure 22-6 shows the Apple menu.



```
@
Mars Scanner…| alert "Mars Scanner ©1999 Rob Martin"
File
saveMovie the movieName
Quit
Location
 Labyrinthus| go "Noctis Labyrinthus"
s Mons| go "Olympus Mons"
 Australe| go "Planum Australe"
 Marineris| go "Valles Marineris"
Infrared
A Screen| OpenScreen()
```

Figure 22-6: When you add the @ menu, the Apple and Application menus become available on a Macintosh.

Installing a menu

After you have created the menu, you must install it, using the Lingo `installMenu` command as follows:

```
installMenu fieldNameorNumber
```

Replace the `fieldNameorNumber` parameter with the name or number of the field cast member that contains the menu instructions. In the following `on prepareMovie` handler, the menu is stored in a field that is cast member 6:

```
on prepareMovie
  installMenu 6
end
```

In this next example, the menu is stored in a field cast member named `"myField"`.

```
on prepareMovie
  installMenu "myField"
end
```

The initial installation of a menu is typically handled from within a movie script, but you can also install a menu from a cast, sprite, parent, or frame script. For instance, you can attach a cast member script to a button that, when clicked, switches from one menu to another. You simply issue the `installMenu` command and name a new field cast member that includes the new menu.

To remove the menu, use the following form of the `installMenu` command:

```
installMenu 0
```

Finishing the job

To finish the project, you need to save the movie and create a projector, play it, and test it. Be sure that the following are true:

- ✦ The menu options work as expected.
- ✦ The area used by the menu bar does not shift Stage objects down and hide important text or images.
- ✦ You have provided all necessary menu options.

Additionally, you might want to enhance your menu, by adding such features as the following:

- ✦ Keyboard shortcuts
- ✦ Checkmarks that indicate the currently selected options

✦ Grayed-out (disabled) options

✦ Formatting (boldface, underline, or shadow) for menu options

Adding keyboard shortcuts to a menu

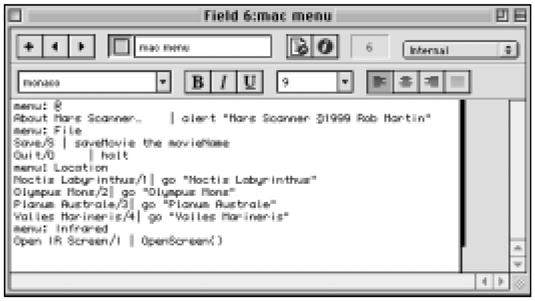
Most applications provide keyboard shortcuts, or quick keys, to access frequently used features. These keyboard shortcuts perform the same function as choosing a menu option. You can easily add keyboard shortcuts to each menu item in your Director movie.

On a Macintosh system, keyboard shortcuts are typically assigned to the Command key plus a letter; on a Windows system, a combination of the Ctrl key plus a letter is used. To add the shortcut, you simply edit the field that contains your menu's instructions and add the slash (/) character, which represents the Command (Ctrl) key, followed by the name of the key you want to use in this key combination to invoke the menu option.

On the CD-ROM

You can find the movie mars scanner 3.dir in the EXERCISE:CH22 (EXERCISE\CH22) folder on the CD-ROM that accompanies this book.

Open the movie mars scanner 3.dir on the CD-ROM. You can add shortcuts by changing the menu instructions to match those shown in Figure 22-7.



```
menu: @
About Mars Scanner…     | alert "Mars Scanner ©1999 Rob Martin"
menu: File
Save/S | saveMovie the movieName
Quit/Q      | halt
menu: Location
Noctis Labyrinthus/1| go "Noctis Labyrinthus"
Olympus Mons/2| go "Olympus Mons"
Planum Australe/3| go "Planum Australe"
Valles Marineris/4| go "Valles Marineris"
menu: Infrared
Open IR Screen/1 | OpenScreen()
```

Figure 22-7: Adding shortcuts to the menu

In this example, either press Command+S on a Macintosh or press Ctrl+S on a Windows system; this is the same as choosing the File ➪ Save command. Pressing Command+Q on a Macintosh or Ctrl+Q in Windows is the same as choosing the File ➪ Exit command. Figure 22-8 shows how the shortcuts appear in the menu.

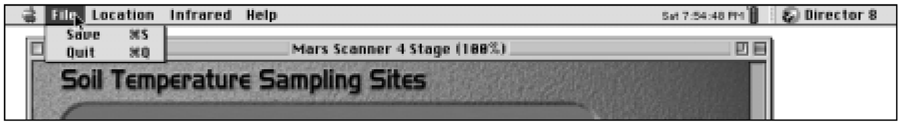


Figure 22-8: A menu with keyboard shortcuts

Make it intuitive!

When you select quick keys (keyboard shortcuts), use commonly accepted intuitive combinations. Users of Macintosh and Windows workstations *expect* consistency from application to application. The following quick-key conventions will be familiar to your users:

- ✦ Command+C (Ctrl+C) invokes the Copy command.
- ✦ Command+N (Ctrl+N) invokes the New Document command.
- ✦ Command+P (Ctrl+P) invokes the Print command.
- ✦ Command+S (Ctrl+S) invokes the Save command.
- ✦ Command+V (Ctrl+V) invokes the Paste command.
- ✦ Command+X (Ctrl+X) invokes the Cut command.

Adding check marks

To add a check mark to a menu item, use `the checkMark of menuItem` property. To display the check mark, you can issue an instruction that uses this syntax:

```
set the checkMark of menuItem itemName of menu menuNameorNumber
to True
```

In this instruction, `itemName` is replaced with the menu item's name or number that you want preceded by a check mark. If you use a number to identify the menu item, you must start counting with the first item on the menu. The `menuNameorNumber` parameter is replaced with the menu's name or number. The menu number is based on the menu's position, counting from left to right on the menu bar.

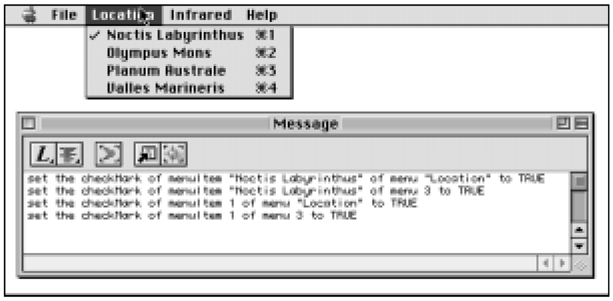
Note

On the Macintosh, the Apple Menu is menu 1. Be sure to include it when you determine the menu number.

Continuing with the mars scanner 3.dir, you could set the check mark to be in front of the first item in the Location menu to show that is where you are. In the Message window, type what is shown in Figure 22-9. You can freely use names or numbers for menus and menu items. Names are often better to use in your code. Numbers are convenient when you just want to test something quickly in the Message window.

To remove the check mark, issue this instruction:

```
set the checkMark of menuItem 1 of menu 3 to FALSE
```



```
set the checkMark of menuItem "Noctis Labyrinthus" of menu "Location" to TRUE
set the checkMark of menuItem "Noctis Labyrinthus" of menu 3 to TRUE
set the checkMark of menuItem 1 of menu "Location" to TRUE
set the checkMark of menuItem 1 of menu 3 to TRUE
```

Figure 22-9: Menus and menuItems can be referenced by name or number, or by a combination of the two.

Disabling menu items

By adding the left parenthesis character — (— after the menu item name, you can display a menu option as disabled. Disabled menu items appear grayed out and cannot be selected with a mouse click. This is a manual way to do it. Most of the time, however, you will be doing it through Lingo.

You use `the enabled of menuItem` property to enable or disable a menu item. In fact, using Lingo to reset `the enabled of menuItem` property while a movie is running is really a better solution than creating multiple field cast members to enable and disable menu options.

Note Even if you disable or enable a menu item by "hard coding" the instructions in the field cast member, you can use Lingo to change the property's setting.

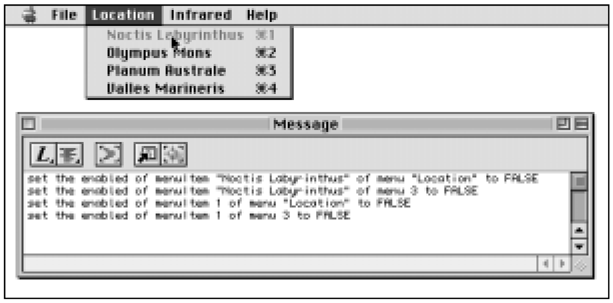
The full Lingo syntax to enable or disable a menu entry is as follows:

```
set the enabled of menuItem itemName of menu menuName to state
```

In this instruction, the `itemName` is replaced with the menu item's name or number. If you use a number to identify the menu item, you must start counting with the first item on the menu. The `menuName` parameter is replaced with the menu's name or number. The menu number is based on the menu's position, counting from left to right on the menu bar.

The last parameter, `state`, is either true or false, depending on whether you want the menu item enabled or disabled, respectively.

As Figure 22-10 shows, you can set `the enabled of menuItem` many different ways. After this property is set to FALSE, the user cannot select the menu item.



```
set the enabled of menuItem "Noctis Labyrinthus" of menu "Location" to FALSE
set the enabled of menuItem "Noctis Labyrinthus" of menu 3 to FALSE
set the enabled of menuItem 1 of menu "Location" to FALSE
set the enabled of menuItem 1 of menu 3 to FALSE
```

Figure 22-10: Just like the checkMark of menuItem, the enabled of menuItem can be set many different ways.

It is possible to use Lingo to disable a menu item with a visible check mark, but it doesn't make much sense to do so. In general, for any given menu item you should decide whether you want it to switch between the enabled and disabled state, or between the checked and unchecked state.

The best way of assigning the correct behavior is to question whether it makes sense to display the menu item's state as turned ON or OFF. If it does, use a check mark. On the other hand, if you want to indicate whether a menu item is currently available or not, use the enabled/disabled pair. For example, a menu item that saves work in progress makes no sense as a toggle — either you can save the file or you can't, and if you can't (because you have not made changes to the file), then the menu item should be disabled. On the other hand, a menu item that turns background music ON or OFF is a perfect toggle, and you should use a check mark to visibly indicate whether the music is audible or inaudible.

Adding text formatting to a menu item

You can add some formatting to your menu, but because the Macintosh and Windows operating systems have their own idiosyncrasies, the menus you create for a Macintosh system might not be supported under Windows (and vice versa). We recommend creating two versions of your menus, one for each platform. Before actually installing a menu, check which platform the user is running, and then install the Macintosh or Windows menu as appropriate.

Tables 22-1 and 22-2 show the options that are supported on the Macintosh and Windows platforms, respectively. Although more options are available for Macintosh programs, many of the options aren't often used these days. The Mac formatting commands can appear before or after the menu item's text, as in the following example:

```
<BQuit
Quit<B
```

An additional option on the Mac is the capability to combine formatting. For example, you can make a menu item bold and italic, as follows:

```
<B<IHelp
```

As mentioned, the formatting characters can come after the text of the menu item, as in the following:

```
Help<B<I
```

Note

In Tables 22-1 and 22-2, the letters in the menu formatting codes (column 1) must be entered in uppercase. The codes can precede or follow the text of the item to be formatted. Unless noted otherwise, the space character that separates the code from the item's text is optional.

Table 22-1
Macintosh Menu-Formatting Options

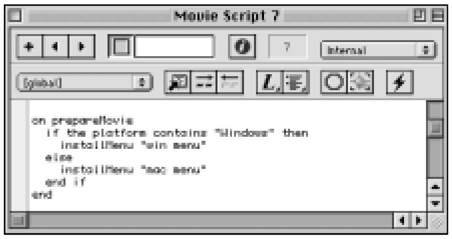
Formatting Code	*Example Usage*	*Results*
@	`Menu:@`	Creates an Apple menu and enables other Macintosh menu items, such as the Application menu.
|	`Quit|halt`	Used to associate Lingo commands with the menu item.
/	`Quit/Q|halt`	Used to create keyboard shortcuts for the menu item.
(	`(Save`	Disables the menu item.
(-	`(-`	Creates a line separator between menu items.
√	`√Music`	Puts a check mark in front of the menu item. Typing Option+v creates the check mark. It is ASCII character 195. Under Windows, it appears as an uppercase *A* with a tilde: Ã. It can be created with the key combination Alt-0-1-9-5. This feature is meant to be used only on Macintoshes.
<B	`<Bcontrols...`	Makes the menu item **bold**.
<I	`<Iselect`	Makes the menu item *italic*.
<U	`<Unavigation`	Underlines the menu item.
<O	`<Otrace`	Outlines the menu item.
<S	`<Sfloat`	Makes the menu item have a shadow.

Table 22-2
Windows Menu-Formatting Options

Formatting Code	*Example Usage*	*Results*
|	`Exit|halt`	Used to associate Lingo commands with the menu item.
/	`Exit/Q|halt`	Used to create keyboard shortcuts for the menu item.
(	`(Save`	Disables the menu item.
(-	`(-`	Creates a line separator between menu items.

Switching the menu based on platform

If you are developing for both Mac and Windows platforms, you will probably end up creating a menu for each platform. If you are on a Mac, rename the field menu **mac menu**; for Windows, name it **win menu**. Then create another menu for the other platform. Next, change the `prepareMovie` handler as shown in Figure 22-11.

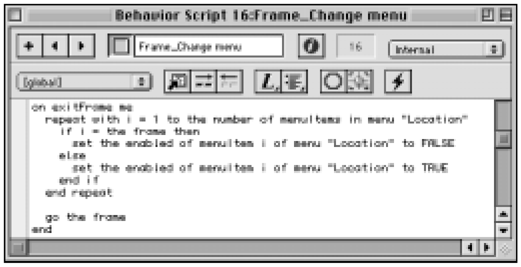


```
on prepareMovie
  if the platform contains "Windows" then
    installMenu "win menu"
  else
    installMenu "mac menu"
  end if
end
```

Figure 22-11: Changing the menu based on platform

Changing a menuItem from a frame behavior

Create a new frame behavior like the one shown in Figure 22-12.



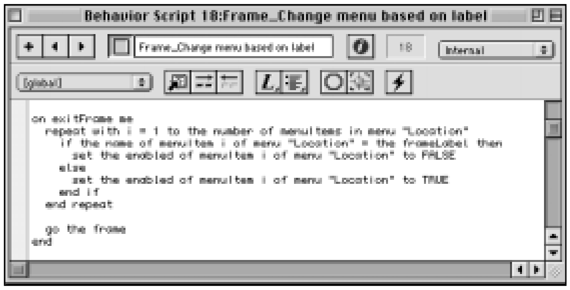
```
on exitFrame me
  repeat with i = 1 to the number of menuItems in menu "Location"
    if i = the frame then
      set the enabled of menuItem i of menu "Location" to FALSE
    else
      set the enabled of menuItem i of menu "Location" to TRUE
    end if
  end repeat

  go the frame
end
```

Figure 22-12: Changing menuItems in the Location menu

Now, as you navigate through the movie, the menu item that corresponds to the frame you are on is disabled. Each `exitFrame` the script iterates through all of the `menuItems` in menu "`Location`". It uses the Lingo expression `the number of menuItems` to determine how many items are in a particular menu. You can also get the number of menus in a movie at any time with — you guessed it — `the number of menus` property.

A more flexible way to do the same thing is based on the name of the marker. Figure 22-13 shows how to implement a script to do this. It uses `the name of menuItem` property and compares it with `the frameLabel` (which returns the name of a marker).


Behavior Script 18:Frame_Change menu based on label
Frame_Change menu based on label
18
Internal
[global]

```
on exitFrame me
  repeat with i = 1 to the number of menuItems in menu "Location"
    if the name of menuItem i of menu "Location" = the frameLabel then
      set the enabled of menuItem i of menu "Location" to FALSE
    else
      set the enabled of menuItem i of menu "Location" to TRUE
    end if
  end repeat

  go the frame
end
```

Figure 22-13: Changing the menuItems based on the frameLabel (the name of the marker)

Earlier, we mentioned that you could, optionally, put spaces before and after the bar character when defining the menu. You were cautioned, however, that the spaces become part of the menu item's name. Here is an instance in which the spaces' presence could create a subtle bug. Suppose that you have the first menu item defined with a space before the bar, as in the following:

```
Noctis Labyrinthus/1 | go "Noctis Labyrinthus"
```

The name returned by the name of `menuItem 1` of menu "`Location`" will be:

```
"Noctis Labyrinthus "
```

If your marker name does not have that space, the two will not be equal. Even though slightly less readable, the following is safer to use:

```
Noctis Labyrinthus/1| go "Noctis Labyrinthus"
```

Changing menu and menu item names

Two additional menu-related properties can be useful when you need to determine the name of a menu or the name of a menu item. In fact, you can even use one of these properties to modify an existing menu during the playback of your movie. The two properties, `the name of menu` property and `the name of menuItem` property, serve parallel functions.

Determining the name of a menu

With `the name of menu` property, you can determine the name of a specific menu. For instance, assume that you have three menus in your movie (File, Edit, and View). If you type the following instruction in the Message window (to determine the name of the second menu), you get the result shown:

```
put the name of menu 2
-- "Edit"
```

Note

You cannot change the name of a menu by using `the name of menu` property; you can only identify the name of an existing menu.

Setting and changing the name of a menu item

With `the name of menuItem` property, you can set or determine the name of a specific menu item. For example, assume that you have three entries under the File menu in your movie (Save, Export, Import). If you type the following instruction in the Message window (to determine the name of the second menu item), you get the result shown:

```
put the name of menuItem 2 of menu "File"
-- "Export"
```

Similarly, you can change the name of the third menu item from Import to Exit (in the Message window or from a handler) by using this instruction:

```
set the name of menuItem 3 of menu "File" to "Exit"
```

Note

You cannot create a menu item from scratch by using `the name of menuItem` property. A menu item in that position must already exist.

Caution

When using the `menuItem` number, always count the number of menu items, *including* disabled items and disabled lines (line breaks in your menu).

Establishing a Dialog Box

Back in the good old days of Director 3.0, Macromedia introduced an XObject that enabled users to create their own custom windows. This precursor of today's Xtras never really worked well, was for the most part undocumented, and was quietly dropped from the available sites when Director introduced the Windows Gaffer for converting Mac files to Windows.

Movies in a window (MIAWs), a part of Director since version 4.0, were intended to fill the need of developers for the occasional dialog box to get information from users. MIAWs (see Chapter 23), however, have several critical flaws:

- ✦ They are memory hungry because they are essentially separate instances of Director that run in their own windows.
- ✦ To create a simple dialog box that returns a value, you must arrange for another Director movie to already be available, and you spend a lot of time coding that movie from both the calling side and the dialog box side.
- ✦ They take a long time to instantiate, which gives the impression that Director applications are slow. They aren't, comparatively speaking, but it reflects badly on your application if it takes 21 seconds for a simple YES/NO dialog box to come up.

Macromedia again addressed the problem of dialog boxes in Director 6, and the solution, though still a little too complex, comes much closer to providing developers with the means to use dialog boxes. The latest incarnation of Director's dialog box handlers is the MUI (Macromedia User Interface) Dialog Xtra that you can use to create general -purpose, OS-compliant dialog boxes. The Xtra creates dialog boxes that look like Macromedia products. You get the grayscale controls that are expected by Macintosh users when a movie plays on the Mac, and the typical controls expected by Windows users when a movie plays on a Windows system.

The MUI Dialog Xtra makes direct calls to the same engine within Director that creates most standard dialog boxes. In other words, you can use the Xtra to draw dialog boxes that are appropriate for either Macintosh or Windows movies with the same set of code. Indeed, the dialog boxes that are created to set Director behavior properties are generated internally with the Xtra. It's not terribly friendly, but the Xtra offers the capability of integrating editable text, slider bars, list boxes, and other interface devices directly into your dialog boxes. Macromedia may even enable ActiveX and Java controls within these dialog boxes at some future time.

Getting basic information with MUI

The MUI Xtra is located in the Director XTRAS folder. The filename for the Macintosh version of the Xtra is mui dialog; the Windows filename is mui dialog .x32.

Note

If you plan to use the MUI Dialog Xtra in your own application, make sure that the Xtra is copied to your application's own XTRAS folder.

After the MUI Xtra is installed, create an instance of the Xtra by using an instruction similar to the following (with the `new` keyword):

```
set muiInstance = new(Xtra "mui")
```

Dialog boxes that open, save, and target

The MUI Xtra usually makes you work to obtain any meaningful information, but it does offer a few freebies. If you need to load or save a file, or if you want the user to enter an Internet address, the MUI Xtra provides access to three functions for these situations: `FileOpen`, `FileSave`, and `getURL`. The first two functions display standard system dialog boxes for loading and saving files, respectively, and they return the full pathname of the file that the user opens or saves. The `getURL` dialog box simply prompts the user to enter or accept a default URL.

Displaying the File Open dialog box

The syntax for the `FileOpen` function is:

```
FileOpen (muiInstance, filename)
-- or dot syntax
muiInstance.FileOpen(filename)
```

The `muiInstance` parameter refers to the name you use when an instance of the Xtra is created. The `filename` parameter stores and returns the filename (of the file to be opened) that is supplied by the user. The following example creates an instance of the Xtra named `messageBox` and displays the File Open dialog box using textfile.txt as the default filename, as shown in Figure 22-14. The second instruction returns the result of the entry in the File Open editable text field and stores that string in the `dialogResult` variable:

```
set messageBox = new(xtra "mui")
set dialogResult = FileOpen(messageBox,"textfile.txt")

-- or dot syntax
messageBox = (xtra "mui").new()
dialogResult = messageBox.FileOpen("textfile.txt")
```

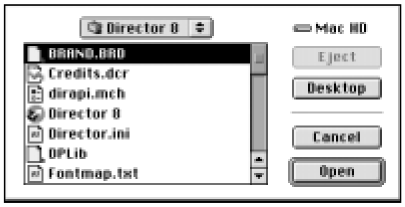


Figure 22-14: The File Open dialog box as drawn by the MUI Xtra

Displaying the File Save dialog box

The syntax for the `FileSave` function is the following:

```
FileSave (muiInstance, filename, prompt)

-- or dot syntax
muiInstance.FileSave (filename, prompt)
```

As in `FileOpen`, the `muiInstance` parameter refers to the name you use when an instance of the Xtra is created. The `filename` parameter is the default name supplied for the user. In other words, when using the `FileSave` function, the text string you substitute for the `filename` parameter appears as the default entry in the editable field in the File Open dialog box. The `prompt` parameter establishes the text that appears on the dialog box's title bar.

In the following example (shown in Figure 22-15), the instance of the Xtra is called `saveMessage`, the default filename to use when saving a file is textfile.txt, and the title bar on the File Save dialog box displays the text `Save the file as`.

```
set saveMessage = new(xtra "mui")
set result = FileSave (saveMessage,"textfile.txt", "Save the
file as" )

-- or dot syntax

saveMessage = (xtra "mui").new()

result = saveMessage.FileSave("textfile.txt", "Save the file
as")
```

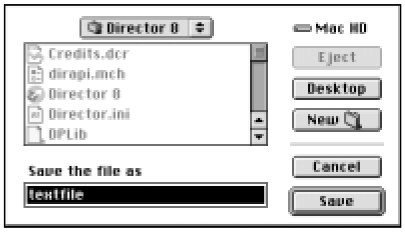


Figure 22-15: The File Save dialog box as drawn by the MUI Xtra

Note The `FileOpen` and the `FileSave` functions do not perform any actual file input/output (I/O). They simply return a filename after displaying the relevant dialog box—it's up to you to save or load the file itself.

Getting an Internet URL from the user

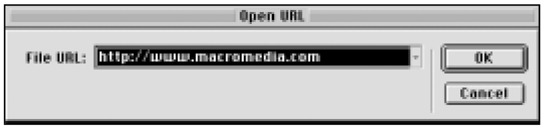
The MUI Xtra also enables you to prompt the user for an Internet address — a handy and useful function. The `getUrl` function pops up a dialog box that requests a URL and can pass a default URL as an argument. The syntax for the instruction is the following:

```
getURL(muiInstance, defaultURL, booleanMoveable)

-- or dot syntax
muiInstance.getURL(defaultURL, booleanMoveable)
```

The `defaultURL` is replaced with the URL that appears in the dialog box by default. The `booleanMoveable` parameter is used to determine whether the dialog box itself can be moved. A TRUE setting makes the dialog box movable and a FALSE setting makes it unmoveable. However, this parameter only works on a Macintosh system. The dialog box is always moveable in a Windows movie.

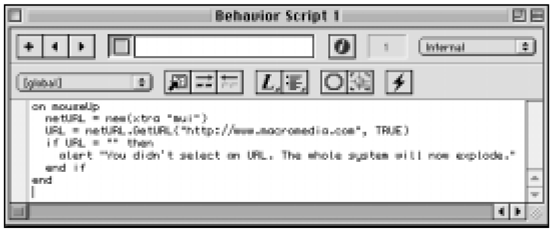
In the example in Figure 22-16, the Open URL dialog box appears (with the Macromedia site as the default entry) when the user clicks a button to which the script is attached.



File URL: http://www.macromedia.com

Figure 22-16: The URL dialog box as drawn by the MUI Xtra

The instance of the Xtra is named `getNetURL`. When the pop-up list appears, the user can click the OK button to use the preexisting URL, or type in a specific URL. If the end user decides to click the Cancel button, the `GetURL` function returns an empty string (" "), so you should always check to see whether the entry is a blank. The script for using this function is shown in Figure 22-17.



```
on mouseUp
  netURL = new(xtra "mui")
  URL = netURL.GetURL("http://www.macromedia.com", TRUE)
  if URL = "" then
    alert "You didn't select an URL. The whole system will now explode."
  end if
end
```

Figure 22-17: Using the MUI Xtra's GetURL function

Abort, Retry, or Ignore

The basic alert box available in Director is useful only for one thing — enabling the developer to track what is happening in the system in real time. From an interface standpoint, however, a dialog box with only one button typically annoys the user. It stops the flow of the program without offering the user any options for handling the situation. This is like telling your neighbor that her house is on fire, her car has just been stolen, and would she like to organize the upcoming block party — and offering no response for her to select other than OK.

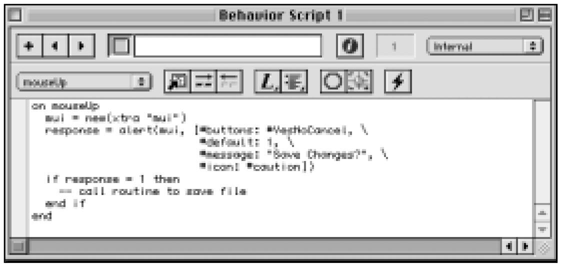
In the MUI Xtra, Director supplies a super-alert box that provides many more capabilities than the fundamental OK button. A typical situation for this usage is when you need to put up a dialog box before quitting to ask users whether they want to save a file before quitting. You can create an alert box with the buttons Yes, No, and Cancel (for aborting the quit process altogether). You can also display an Alert icon similar to system dialog boxes. Table 22-3 summarizes the various options for putting together a custom alert box.

Table 22-3
MUI Alert Box Options

Property	***Possible Values***	***Results***
`#buttons`	`#Ok`, `#OkCancel`, `#YesNoCancel`, `#YesNo`, `#RetryCancel`, `#AbortRetryIgnore`	Establishes which buttons are displayed in the alert box.
`#default`	0, 1, 2, or 3	If the default is greater than 0, then the indicated button (the first, second, or third) is set as the default, and is the button selected if the user presses the Return (Enter) key. A default of 0 means there is no default.
`#title`	String (for example, `File Save`)	Establishes the text that appears on the title bar of the alert box. Used in a Macintosh movie only when the alert box is movable. If not stipulated, a `<Null>` entry appears in the Windows title bar.
`#message`	String (for example, `Save Before Quitting`?)	Establishes the text that appears within the dialog box itself.

Property	***Possible Values***	***Results***
`#icon`	`#stop`, `#note`, `#caution`, `#question`, `#error`	Sets the icon that is displayed in the alert box. If you want a custom icon, you have to create the dialog box from scratch.
`#movable` or `#moveable`	True, False	Indicates whether the alert box is moveable or not. All alert boxes in a Windows movie are moveable, regardless of this setting.

Creating an alert dialog box is only slightly more complicated than making a `GetURL` dialog box. You need to pass both the MUI instance and a property list containing the appropriate values for the alert box. In other words, you must explicitly establish each of the properties for the alert box. The MUI Xtra does not assume any default values; you must set them. For example, to create a Save Before Quitting dialog box and attach the script to a Save button, you can use the instructions in Figure 22-18.



```
on mouseUp
  mui = new(xtra "mui")
  response = alert(mui, [#buttons: #YesNoCancel, \
                         #default: 1, \
                         #message: "Save Changes?", \
                         #icon: #caution])
  if response = 1 then
    -- call routine to save file
  end if
end
```

Figure 22-18: Using the MUI Alert function to get feedback from the user

The `alert` method returns the number of the button that was pressed. Here, for example, if the Yes button were pressed, the response would be 1. The dialog box created with this script is shown in Figure 22-19.

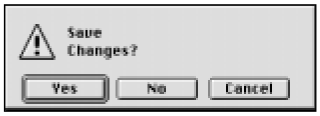


Figure 22-19: The MUI Alert dialog box

As you discover in the upcoming exercise, the MUI Xtra also returns a value corresponding to the position of the button clicked. For instance, in the preceding example (based on the use of the `#YesNoCancel` symbol), the Yes button is button 1, the No button is button 2, and the Cancel button is button 3. A click on button 3 returns a value of 3. The default button was set to the first button in Figure 22-18. Try changing it to one of the others or changing some of the other values.

Getting fancy

The MUI Xtra features discussed in this chapter represent only a fraction of the Xtra's capabilities. To implement anything of substance with the MUI Xtra requires some serious programming and is outside the scope of this book.

Summary

Among the things that you learned in this chapter are these:

- ✦ You can create a pull-down menu using a field cast member and the `installMenu` command.
- ✦ Menus provide an easy way for beginners to access features in your program.
- ✦ You might want to create unique menus for each platform (Macintosh and Windows), because some Macintosh formatting is not available in a Windows movie.
- ✦ You can modify menu items or swap menus on-the-fly by using Lingo.
- ✦ The MUI Xtra is built into Director, and can be used to add more powerful dialog boxes.
- ✦ You can use MUI Xtras to create sophisticated dialog boxes that are essentially identical to behavior property boxes.

The next chapter shows you how to use movies in a window.

✦ ✦ ✦