CHAPTER 25

Xtras and File Manipulation

In This Chapter

Understanding different types of Xtras

Xtra methods

Opening a file by using the FileIO Xtra

Saving a file by using the FileIO Xtra

Macromedia Director is an extremely powerful authoring system — but no matter how many bells and whistles a product has, someone somewhere will want it to do more. Using Xtras, you can extend the already considerable capabilities of Director.

Xtending the Power of Director

The capacity to extend the features of an application is called *extensibility*. Two familiar examples of extensibility are the plug-ins used by Web browsers (such as Netscape Navigator and Microsoft Internet Explorer) and the plug-in filters supported by Adobe Photoshop. In both cases, using small modules of code that enrich the product's feature set enhances the base product.

Director's extensibility comes from plug-in code modules called Xtras that are developed by Macromedia and by third parties. Xtras are written in a language (often C or C++) that produces machine-language code. Prior to Director 5, programmers could write XObjects using C or other programming languages to enhance the capabilities of Director. Just as XObjects are extensions to the Lingo programming language, Xtras are more powerful extensions to Macromedia programs. Most of the XObjects available before the advent of Director 5 are now available as scripting Xtras (see the discussion under "Scripting Xtras" later in this chapter).

What Is an Xtra?

The software routines in Xtras (and their predecessors, XObjects) are written in C. Director supports four kinds of Xtras that you can call, as you would call a Lingo command, or access from a Director menu. Either way, you extend the capabilities and functionality of your Director movie. As the X in its name implies, an Xtra's code is external to Director.

Because Xtras interact at a low level with the operating system, they are platform specific. So, an Xtra written for the Macintosh OS cannot be used on a Windows system, and vice versa. Most developers compile and provide Xtra code for both platforms as separate externals.

Why use Xtras?

When you first begin programming with Director, you may be overwhelmed with the application's native capabilities. After you gain a foothold on the learning curve and start building your own projects, however, you may wind up wanting "More! Now!" Director and Lingo cannot possibly include all the tricks and treats that each of 300,000 developers wants and requires — and Xtras are what fill these "feature gaps." Xtras provide additional functionality in specific areas where Director is lacking. More commonly, an Xtra may fit the bill when Director "twists," but the developer also wants it to "shout."

Xtras also add new tools to Director's tools set — similar to plug-in filters in Photoshop. Extending Director with Xtras is so powerful that many of the new features in Director are implemented as Xtras.

Types of Xtras

Director Xtras fall into five general categories: transition Xtras, cast member Xtras, scripting Xtras, tool Xtras, and importing Xtras.

Transition Xtras

Director has 52 predefined, built-in transitions. You can enlarge the number and type of transitions available even further by installing one or more transition Xtras.

Transition Xtras control and customize frame-to-frame Stage changes. A transition Xtra can include its own custom properties, a Properties dialog box, an animated thumbnail, a cast window icon, and an About box. It can use a combination of visual change, sounds, and other media to handle the changes between frames in your movie.

As soon as you use a transition Xtra in the Score's Transition channel, it appears in the Cast window, just like any other cast member.

Xtra resources

The number of Xtras available for use with Director seems to grow almost daily. Three Web sites are especially helpful for tracking down Xtras that meet your specific programming needs. The first place to check is Macromedia's Web site at:

```
http://www.macromedia.com/software/xtras/director/
```

Other useful sites are:

Gretchen MacDowell's Mile-High Table of Products for Director at:

```
http://www.updatestage.com/products.html
```

Tab Julius's Penworks Corporation sells Xtras and holds training seminars on how to develop Xtras:

```
http://www.penworks.com/xtras/
```

Bruce Epstein's Zeus Productions:

```
http://www.zeusproductions.com/xtras/
```

Like other Xtras, to be accessible during playback, a transition Xtra must be distributed with the movie. Typically, packaging a transition Xtra is handled automatically when you activate the Check Movies for Xtras option in the Projector Options dialog box. Third party Xtras need to be added to the Xtra list in order to be bundled into a projector.

Cast member Xtras

Cast member Xtras are similar to other objects that you include in the cast of your movie. They add functionality, such as database engines, text-processing engines, and graphics engines. After being installed, the cast member Xtra appears on the Insert menu. Depending on how it is designed, a single cast member Xtra can add one or more menu items to the Insert menu.

Cast member Xtras are sometimes referred to as *sprite* Xtras (because they operate as sprites on the Stage after assignment to the Score) or *asset* Xtras (because they increase the assets available to the movie). Flash, GIF, and QuickTime members are all asset Xtras. Not all asset Xtras can be sprites. Shockwave audio members, for example, appear in the Cast Window, but they do not have a Stage presence.

Just like other Director sprites, the cast member Xtra is displayed on the Stage, receives and reacts to user actions (mouse clicks, keypresses, and movement of the playback head), and responds to standard and custom Lingo commands. Similar to a transition Xtra, a cast member Xtra may include its own custom properties, a Properties dialog box, a media editor, an animated thumbnail, a cast window icon, and an About box.

Here's the general procedure for using a cast member Xtra:

1. Select a location in the Cast window.
2. Select the cast member Xtra from the Insert menu. The cast member Xtra appears in the Cast window and can be assigned to the Score.
3. To set the Xtra's properties, select the object in the Cast window, open the Cast Member Properties dialog box, and click the Options button.
4. You can open the Xtra's media editor by double-clicking the cast member's thumbnail in the Cast window.

To be accessible during playback, a cast member Xtra must be distributed with the movie. Typically, packaging a cast member Xtra is handled automatically when you activate the Check Movies for Xtras option in the Projector Options dialog box.

Scripting Xtras

Scripting Xtras enhance Director's scripting language by adding new commands and new capabilities to Lingo. To access the functions provided by a scripting Xtra, you must use the Xtra's built-in methods (which are discussed later in this chapter in "Using Xtras in Director"). Scripting Xtras can control movies, internal and external casts, cast members, Scores, media, and almost any Director feature. The FileIO Xtra, shipped with Director and available on Macromedia's Web site, is an example of a scripting Xtra. It enables you to open, close, save, and append data to an external file. The Beatnik Lite Xtra discussed in Chapter 18 is another example of a scripting Xtra.

Scripting Xtras on the Windows platform take the form of dynamic link libraries (DLLs). For Director to have access to the power of the Xtra, the file ending with the .DLL extension must be placed in the Xtras folder or included in the same folder as your projector.

For a scripting Xtra's methods to be accessible during playback, the Xtra must be distributed with your movie. This type of Xtra must be added manually to the list of Xtras that can be packaged in a projector. To add a scripting Xtra, choose Modify ➪ Movie ➪ Xtras and then click Add. After the scripting Xtra appears on the list of movie Xtras, it is automatically packaged with the projector if you activate the Check Movies for Xtras option in the Projector Options dialog box.

Tool Xtras

Tool Xtras add custom tools to create or modify cast members and sprites. Tool Xtras can take these forms:

- ✦ Bitmap filters (such as Photoshop filters)

When three's a welcome crowd

Ever wonder about the origins of the term *third party*? When you deal with the developer and publisher of a product such as Macromedia, the company is the "party of the first part," and you (the developer who pays a fee to use the product provided by the company) are the "party of the second part." Any person or company who creates and sells you add-on tools, utilities, or services to help you better use the original product is the "party of the third part."

- External casts that can be placed in the LIBS folder or in the XTRAS folder. Putting them in the XTRAS folder makes them accessible from the Xtras menu.
- Director movies (open as an MIAW in authoring mode). The ScoreSaver that you created in Chapter 22 is one such Xtra. On the CD-ROM is the Behavior Writer Xtra by Roy Pardi, which is also an MIAW (movie in a window).

The tool Xtra is only useful when you are creating and editing a movie. As such, this type of Xtra is only available in authoring mode and is not useful or available when a movie plays. Tool Xtras are not packaged in projectors for distribution to the end-user.

Tool Xtras are placed in the XTRAS folder (or in subfolders within the XTRAS folder), and are accessible from the Xtras menu.

Importing Xtras

An importing Xtra includes the code required to import various types of assets (media) into Director. Director uses an importing Xtra to access external media when the movie plays. If the Xtra is not available, Director cannot import the specified type of media. When you create a projector with the Check Movies for Xtras option enabled (in the Projector Options dialog box), Director automatically adds the importing Xtras required to access any media within your movie.

Using Xtras in Director

Using Xtras in Director can be a bit of a mystery to someone new to programming, in part because Xtras often do not have a visual component. To effectively use an Xtra, you must:

- Know where to install the Xtra so that Director can find it, if it is not already installed.
- Install the Xtra, if Director does not automatically install it.

- ✦ Know how to use specific Lingo commands to access and use Xtras (for scripting Xtras only).
- ✦ Identify and use commands that are specific to the Xtra (for scripting Xtras only).
- ✦ Know how to access Xtras by means of the Director menu system (for cast member, transition, and tool Xtras).
- ✦ Know how to package Xtras in projectors (for all Xtras except tool Xtras).

How does Director know an Xtra exists?

For Director to recognize and use an Xtra during authoring, the Xtra must be installed in one specific location. Director will always look in the XTRAS folder within the Director folder, or in a subfolder within the XTRAS folder.

When you create a projector that requires an Xtra to run, the Xtra should be included in the same folder as the projector. If the Xtra is stored elsewhere, you must provide the path to the file every time the Xtra is called within your movie.

Director enables you to place Xtras up to five levels deep within the XTRAS folder, and it will still locate and automatically open the Xtra when it's needed. This feature enables you to group Xtras by function or vendor in the XTRAS folder's subfolders.

If you run a movie or projector that references an Xtra that Director cannot find, the program issues you one of three types of warnings:

- ✦ If the missing Xtra is a transition Xtra, Director substitutes a simple Cut transition.
- ✦ For missing cast member Xtras, Director displays a red X character on the Stage. This serves as both a warning and a placeholder for the missing cast member Xtra. In the Cast window, the icon is changed to four arrows pointing toward the center, as shown in Figure 25-1.
- ✦ In all other cases when an Xtra is missing, Director displays an alert box, as shown in Figure 25-2.

Figure 25-1: The icon for the cast member is changed, indicating that the Xtra for this member is missing.

Figure 25-2: When a movie contains an asset whose Xtra is missing, an alert appears when the movie opens.

Installing new Xtras

If you are installing a new Xtra, make sure it is copied to the XTRAS folder or to a subfolder within the XTRAS folder on your hard drive. When properly placed, Xtras are automatically opened when Director is loaded and automatically closed when you exit the program. Here, "properly placed" means installed in the folders Director checks, as described in the preceding section.

If you install a new Xtra by copying it to the XTRAS folder, you must exit Director and then reopen the application before the new Xtra is recognized.

Although the actual implementation of a new Xtra depends on the type of Xtra installed, Director does the following things after it recognizes a new Xtra:

- Adds the Xtra to the Insert menu for asset Xtras.
- Adds the Xtra to the Xtras menu.
- Makes the Xtra accessible by means of Lingo.
- Adds the Xtra to the list of transitions available from the Frame Properties: Transition dialog box for transition Xtras.

Using the openXlib and closeXlib commands

Xtras in the Xtra folder are automatically opened when Director starts. If you choose to store new Xtras in a different folder, you must use the `openXlib` command to open the Xtra and make it accessible for use in your movie.

Xtras are opened by using the `openXlib` command. The syntax for the `openXlib` command is simple. You follow the command with the Xlibrary filename that you want to open, like this:

```
openXlib "FileIO"
```

After using the `openXlib` command to open an Xlibrary, you can manually close the Xlibrary by using the `closeXlib` command, like this:

```
closeXlib "FileIO"
```

When you're opening and closing the Xlibrary under Windows, the .DLL extension is optional. You do not need to include it in the command.

If you do not indicate a specific Xlibrary file name in the `closeXlib` command, all open Xlibrary files are closed. You'll do well to develop the habit of closing Xlibraries as soon as you have finished using them.

Before you proceed to use Xtras in the construction of your movies, you must know which ones are installed on your system. The easiest way to find out is to issue the `showXlib` command in the Message window.

Remember that you access some Xtras (the scripting Xtras) by using Lingo. Other types of Xtras appear as part of the Director user interface — menu choices or cast members or both.

Using Xtra-Related Lingo Commands

Lingo includes some commands that are specific to the use of Xtras. You have already read about three of these commands: `closeXlib`, `openXlib`, and `showXlib`. In addition, you can use Lingo to create a new instance of an Xtra (similar to creating an instance of a child object, as discussed in Chapter 13). Because Xtras include their own command set, you need a method to discover which commands are available for use with each Xtra.

We'll discuss the most commonly used Xtra — FileIO. It is a scripting Xtra, like the Beatnik Xtra discussed in Chapter 18. The Xtra is named after its function: file input and output. Being able to read information and write information is a task you do for almost every projector-based project. It might seem odd at first to use functions that do not appear in the Lingo menu. After you become comfortable working with Xtras, you'll wonder how you ever lived without them.

Creating a new instance of an Xtra

To use an installed Xtra in your movie, you still need to create an instance of the Xtra. The way in which you create an instance varies depending on the type of Xtra. For example, the method that you use to create a scripting Xtra is different from the method that you use to create a transition, tool, or cast member Xtra. Because the latter three types can be created using Director's user interface (menu choices and so on), we focus here on the procedure for creating an instance of a scripting Xtra.

To create the new instance of the scripting Xtra, you use the `new` function, with the keyword `xtra` as the first parameter and the name of the Xtra as the second parameter, as in the following:

```
new (xtra nameOfTheXtra)
```

To effectively use the newly created instance, you assign the instance to a variable. In the following example, an instance of the FileIO Xtra is assigned to the variable `aQuote`:

```
aQuote = new(xtra "FileIO")
```

Like a child object, when you have finished using an instance of an Xtra, you should remove the instance from memory. To do so, simply set the variable representing the instance to VOID. The following example removes from memory the `aQuote` instance of the FileIO Xtra:

```
aQuote = VOID
```

Listing the messages that you can send to an Xtra

Each Xtra can receive messages that cause a specific action to occur. Earlier, we loosely referred to these messages as commands. In reality, each Xtra includes specific handlers (just like the scripts you have created using this book), but you cannot see them. These invisible handlers are called *methods*. When you send a message to the Xtra, the method is executed.

Obviously, because they're invisible, you'll need a method to determine the methods available for an Xtra. That's where the `interface` command comes in. The following instruction displays the Xtra's command in the Message window:

```
put interface(xtra "FileIO")
```

See Figure 25-3 for the results.

Figure 25-3: The methods available for the FileIO Xtra

Note the last two methods listed for the FileIO Xtra: one begins with a plus sign, and the other begins with an asterisk. A plus sign indicates that you can call this method without creating an instance, but you *do* need to reference the Xtra:

```
put version(xtra "FileIO")
-- "FileIO for Director 7.0.0.r198 (Dec 2 1998)"
```

An asterisk indicates that you do not need to create an instance or use a reference to the Xtra to call this method:

```
put getOSDirectory()
-- "Mac HD:System Folder:"
```

All the other methods require creating an instance of the Xtra to use them.

Using Xtra-Specific Methods

After you know the specific methods available with an Xtra, you can incorporate them into your movie and access the features of the Xtra.

The FileIO Xtra provides file-management services that Director itself does not offer. With FileIO, you can open, close, save, create, and delete a disk file. You can also display an Open dialog box, save it, and filter (restrict) the files that appear in the dialog box. This Xtra also enables you to read and write a character, string, line, word, or the entire file.

The commands covered in the following sections are not Lingo commands; they are specific to the FileIO Xtra. If the Xtra is not available, the commands are not available. When you rely on methods available through an Xtra, it's easy to forget that these methods or commands are not native to Director and Lingo. For a project that relies on one or more scripting Xtras, you must distribute the Xtra(s) with the project. Because third-party developers typically develop Xtras, be sure that you are licensed to distribute the Xtra with your product.

Opening a file by using the FileIO Xtra

Before you can read or write to a file, the file must be opened by using the `openFile` method, as follows:

```
openFile (instance, nameOfFile, mode)
-- or dot syntax
instance.openFile(nameOfFile, mode)
```

The instance refers to the instance of the FileIO Xtra. In the following example, the instance is set to a variable named `aQuote`:

```
aQuote = new(xtra "FileIO")
```

A single instance of FileIO can reference a single open file. To open more than one file at a time, you must create multiple instances of FileIO.

If you use this instruction in a movie, you would substitute `aQuote` for `instance` in the syntax of the instruction.

The `nameOfFile` parameter is replaced with the filename of the disk file you want to open. If this parameter refers to an actual filename (rather than a variable name), the filename must be enclosed in quotation marks. The filename can be qualified by using the path to the file on the local computer system. If the file to open is in the same folder as the Director movie or the projector, you can use a relative pathname or list just the filename itself.

The last parameter for the `openFile()` command is `mode`. Files are opened in one of three modes. You replace the mode parameter with one of the following numeric codes. To open a file:

- ✦ In read-only mode, use the numeric code 1.
- ✦ In write-only mode, use the numeric code 2.
- ✦ In read and write mode, use the numeric code 0.

After you set the mode to write or read/write, you can write to a disk file. Everything after the current position in the file will be overwritten by the write operation.

In the following example, Director uses the FileIO Xtra to create an instance of the Xtra called `aQuote`, and to open a file named ReadThis.Txt in read-only mode:

```
openFile(aQuote, "ReadThis.Txt", 1)
-- or dot syntax
aQuote.openFile("ReadThis.Txt", 1)
```

Closing a file by using the FileIO Xtra

When you have finished working with the data in a file, you must close it. You cannot close a file unless it has been previously opened by means of the `openFile` method. Here is the syntax for the instruction to close a file:

```
closeFile(instance)
-- or dot syntax
instance.closeFile()
```

In the following instruction, the `aQuote` instance of the FileIO Xtra is closed:

```
closeFile(aQuote)
-- or dot syntax
aQuote.closeFile()
```

Displaying an Open dialog box

Users are accustomed to seeing an Open dialog box when it comes time to load a file. The FileIO Xtra includes a method that displays a platform-specific Open dialog box, with which the user can select a file to open.

This method does more than just display the dialog box. It returns a fully qualified path and filename, which can then be used in your Lingo scripts and handlers. To access this method, use the following syntax:

```
displayOpen(instance)
-- or dot syntax
instance.displayOpen()
```

To use the path and filename in your Lingo handlers, you can use an instruction similar to this:

```
fileName = displayOpen(instance)
-- or dot syntax
fileName = instance.displayOpen()
```

In this example, the `fileName` variable now stores the path and filename returned by the `displayOpen` method.

Using the FileIO Xtra filterMask

You can limit the files visible in an Open dialog box by using the FileIO Xtra's `setFilterMask` method. (This method is also used with the Save dialog box discussed later in the chapter in the section "Displaying a Save dialog box."):

```
setFilterMask(instance, filterMask)
-- or dot syntax
instance.setFilterMask(filterMask)
```

The `filterMask` parameter is replaced with a text string identifying the file type (Windows only) and an associated file extension (Windows and Macintosh). Together, these values restrict the type of files that appear in the dialog box. By default, when you create a new instance of the FileIO Xtra, `filterMask` is set to all files (`*.*` or `""`).

The `setFilterMask` method must be issued prior to using the `displayOpen()` or `displaySave()` method.

Setting filterMask under Windows

The `filterMask` parameter differs slightly, depending on whether you're using the Windows or Macintosh platform. When you create a filter mask for use on a Windows system, you must stipulate the type of files using a text string (such as `"Text Files"`) followed by the file's extension (such as `*.txt` or `*.doc`) The `filterMask` value is limited to 256 characters under Windows.

On a Windows system, the following instruction displays only text files in the Open dialog box:

```
setFilterMask(aQuote, "Text Files, *.txt")
-- or dot syntax
aQuote.setFilterMask("Text Files, *.txt")
```

On the Windows platform, you can offer users the capability to access more than one filter by listing multiple file types in the filter mask. For instance, the following instruction creates two filters:

```
setFilterMask(aQuote, "Document Files, *.doc, Text Files,
*.txt")
-- or dot syntax
aQuote.setFilterMask("Document Files, *.doc, Text Files,
*.txt")
```

Each specified type is not displayed simultaneously, however; instead, the user has an opportunity to select one of the listed filter masks from a drop-down list. The first filter you list in the instruction becomes the default selection when the Open or Save dialog box appears.

Setting filterMask on a Macintosh

When you create a filter mask on a Macintosh system, you do not use a text string to describe the type of file. You include only a string of file types, using four-character case-sensitive codes (such as `TEXT` or `PICT`) in the filter mask. On the Macintosh, you are limited to four sets of four-character filename extensions.

In the following example, only text files will be displayed:

```
setFilterMask(aQuote, "TEXT")
-- or dot syntax
aQuote.setFilterMask("TEXT")
```

You can include additional file types (up to a maximum of four) by stringing them together. For example, this command displays only text files and bitmap (PICT) files:

```
setFilterMask(aQuote, "TEXTPICT")
-- or dot syntax
aQuote.setFilterMask("TEXTPICT")
```

Table 25-1 lists a few of the four-character codes that you can use with the `setFilterMask` method on the Mac.

Table 25-1
File Type Codes for setFilterMask

File Type Codes	*File Type*
`MV07`	Director DIR
`FGDM`	Director DCR
`M!07`	Director DXR
`PICT`	PICT file
`TEXT`	Text file

An easy way to discover the file type of a file is to use the Find File application. Open the Find File application from the Finder (press Command+F) and set the pop-up to file type, as shown in Figure 25-4. Then drag a file of the kind whose type you want to find out onto the Find File window, and the type of that file appears in the field.

Figure 25-4: Using Sherlock's Find File application to discover file types on a Mac

Resetting filterMask to all files

You might need to reset the drop-down list to All Files. To reset the filter mask to display All Files, use this instruction:

```
setFilterMask(aQuote, "")
-- or dot syntax
aQuote.setFilterMask("")
```

The advantage of this form is that it works for both Windows and Macintosh platforms. The *.* form is Windows-only.

Accessing data in a file

The FileIO Xtra has several methods available for you to read data from a file. Each method reads a specific chunk of data, as identified in Table 25-2. To use any of these methods, the file must be open in read or read/write mode.

Table 25-2
Methods for Use with the FileIO Xtra to Read Chunks of Text

FileIO Xtra "Chunk" Methods	*Result*
`readChar(instance)`	Reads the character (either single- or double-byte) at the current position of the pointer. Next, the method increments the position and returns the character to Lingo as a string.
`readWord(instance)`	Reads the next word starting at the current pointer position. Next, the method returns the string to Lingo.
`readLine(instance)`	Reads from the current position, up to and including the next carriage-return in the file. Next, the method increments the pointer's position and returns the string to Lingo.
`readFile(instance)`	Reads from the current position of the pointer to the end-of-file (EOF) character. Next, the method returns the file to Lingo as a string.

After you access the data from a file, you need to assign it to a variable, so that the data can be used by your movie. In the following snippet of code, the contents of the entire file are assigned to the contents variable. In the second line of code, the data stored in contents is placed in the field cast member named Today's Quote:

```
contents = aQuote.readFile()
member("Today's Quote").text = contents
```

Alternatively, you can just read the file and place it into the member at the same time:

```
member("Today's Quote").text = aQuote.readFile()
```

In the next exercise, you use a text file (artquote.txt) and a partially complete Director movie (artquote.dir). In its current condition and as shown in Figure 25-5, artquote.dir includes a bitmap cast member called Art (cast member 1) and a frame script (to cycle the movie on frame 1).

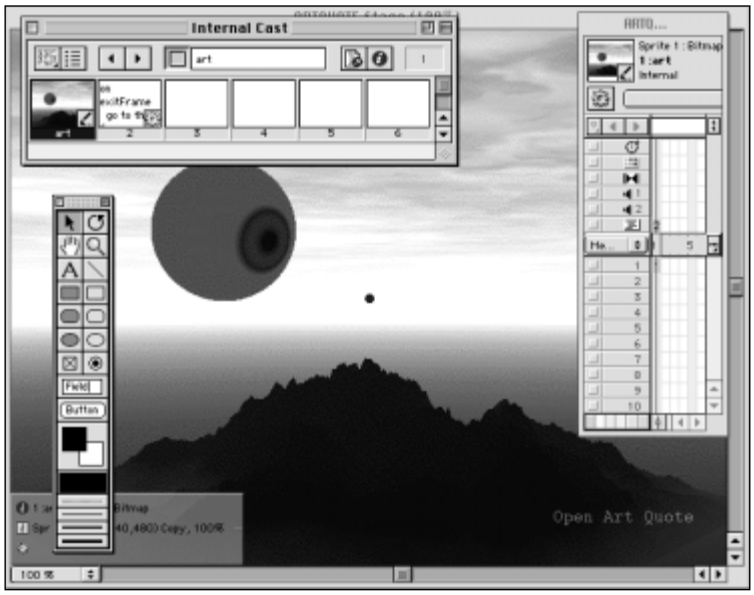

Figure 25-5: The artquote.dir movie on Stage with the floating Tool Palette visible

Note the text `Open Art Quote` in the bottom-right corner of the Stage. In the following steps, you create a transparent button (with a script attached) to appear over that text. When the user clicks this button, the attached cast member script executes. The script then opens an external text file and reads its contents into a field on the Stage.

Because of the platform-specific parameters required, the instruction that calls the `setFilterMask` method differs based on whether your movie runs on Macintosh or Windows. You select the instruction to execute by using the platform function:

```
if the platform contains "Windows" then
  --Instructions for Windows Systems
else
  --Instructions for Macintosh Systems
end if
```

In the script you add to the movie, you use the bits of example code from the preceding discussion of the FileIO Xtra. Examine the code, and you can see that the script creates an instance of the FileIO Xtra, sets a filter mask, displays the Open dialog box (using the filter mask) with only text files visible, opens the file selected by the user, reads the data into a variable named `contents`, places the text into a field called Today's Quote, and then closes the disk file. At the end of the handler, the instance of the FileIO Xtra is eliminated.

You can find the artquote.dir and artquote.txt files on the companion CD-ROM in the EXERCISE:CH25 (EXERCISE\CH25) folder.

Opening a File by Using the FileIO Xtra

1. Open the artquote.dir movie in Director.
2. If the Tool Palette is not displayed, open it (press Command+7 or Ctrl+7).
3. At the bottom of the Tool Palette, set the border to No Line.
4. Be sure that the foreground color chip is set to black and the background color chip is set to white.
5. Select the Rectangle tool (*not* the Filled Rectangle tool).
6. Drag to create a rectangle that surrounds the `Open Art Quote` text on the Stage, as shown in Figure 25-6.

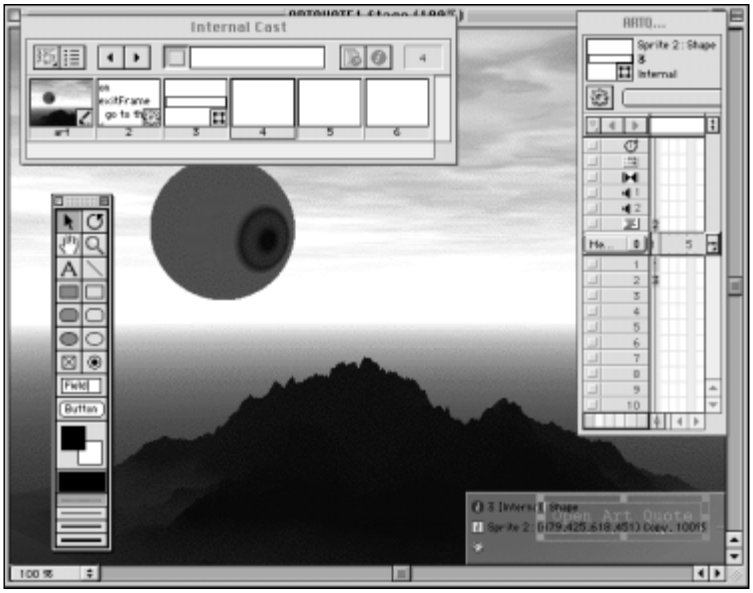

Figure 25-6: A rectangle area drawn for the transparent button

7. With the rectangular area still selected, shorten the sprite to span a single frame (frame 1) and set the ink effect to Matte.
8. Select the new shape cast member in Cast window, slot 3.
9. Create the behavior shown in Figure 25-7 and attach it to sprite 2 (the invisible rectangle). This script creates a FileIO instance, sets the filter mask based on platform, displays an open-file dialog box, checks to make sure the user didn't

cancel, then opens the file, reads it, and puts text into the member named Today's Quote, and then closes the file. Because `aQuote` is a local variable, it will no longer be in memory after the handler ends, so it does not need to be assigned VOID, but you do need to make sure you close the file.

Figure 25-7: A behavior to read a text file

10. Close the Script window.

11. Select the Text tool on the Tool Palette.

12. Drag in the white area of the Stage image, as shown in Figure 25-8, to build a new text member.

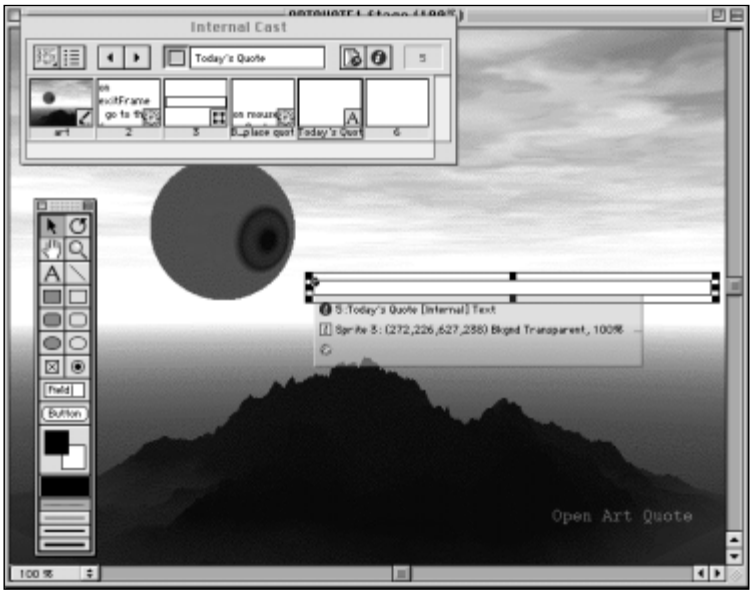

Figure 25-8: A text member created for the quotation

13. Set the ink to Background Transparent. Set the font to Times (Times New Roman for Windows). Set the point size to 12.
14. Select the text member (slot 4) in the Cast window.
15. Change the field cast member name to **Today's Quote**.
16. Save the movie as **artquot1.dir,** rewind it, and play it.
17. With the movie running, click the Open Art Quote (transparent) button.
18. When the Open File dialog box appears, navigate to the folder on your local hard drive where you store your practice files. Locate and select the artquote.txt file, and click the Open button.

    Your Stage now appears similar to Figure 25-9. If you stop the movie and check the Today's Quote field, it contains the text of the artquote.txt file.
19. Save the movie again as artquot1.dir and close it.

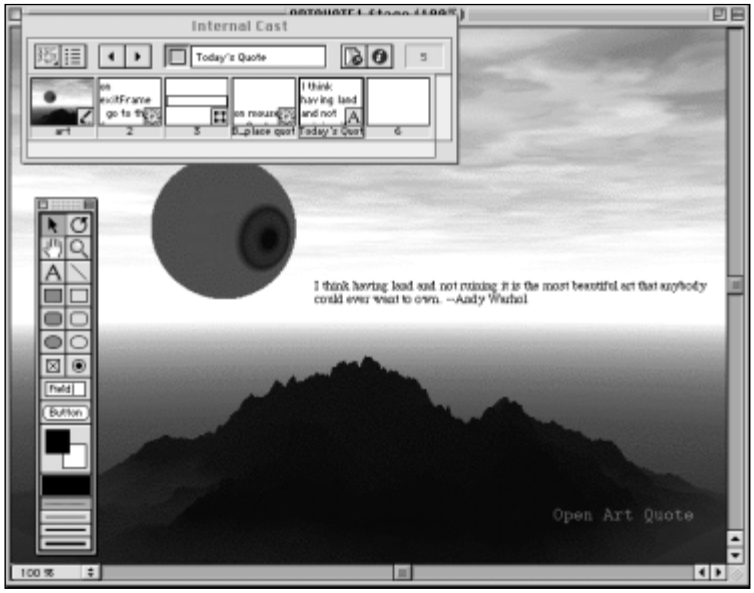

Figure 25-9: The completed artquote.dir movie

More FileIO methods

In addition to what you've used so far, there are several other methods available through the FileIO Xtra. You still need to learn to open a Save dialog box, create a new disk file, and save data to it from your movie.

Displaying a Save dialog box

Users are accustomed to seeing a Save dialog box when it comes time to save a file. The FileIO Xtra includes a method, displaySave, that displays a platform-specific Save dialog box that enables the user to type the path and filename of a file to save.

You use the setFilterMask method to determine which file types appear in the Save dialog box. For it to affect the files displayed, setFilterMask must be issued before using the displaySave method. The setFilterMask does more than just display the dialog box. It returns a fully qualified path and filename (indicating where the file is saved), which can then be used in your Lingo scripts and handlers.

To access the displaySave() method, use this syntax:

```
displaySave( instance, titleName, defaultFileName )
-- or dot syntax
instance.displaySave(titleName, defaultFileName)
```

The titleName parameter sets the text that appears on the title bar of the Save dialog box. If you want the data to be stored in a specific file by default, you can use the defaultFileName parameter to establish the path and filename. To save game scores, for example, in a file called scores.txt in the same folder as your movie's projector, you would use this instruction:

```
displaySave(saveScores,"Save Game Scores","scores.txt")
```

This instruction designates the string "Save Game Scores" as the text to appear on the Save dialog box's title bar.

To use the path and filename in your Lingo handlers, you can use the set command to store the information in a variable. The following instruction is an example of this technique:

```
fileName = displaySave (saveScores,"Save Game
Scores","scores.txt")
-- or dot syntax
fileName = saveScores.displaySave("Save Game Scores",
"scores.txt")
```

In this example, the fileName variable stores the path and filename returned by the displaySave method.

Creating a new disk file

Using the createFile method from the FileIO Xtra, you can create a new disk file in the current Directory, or in a different folder if you specify the path. The syntax for this method is as follows:

```
createFile( instance, newFileName )
```

If the `newFileName` parameter is an actual filename, it must be surrounded by quotation marks. If it is a variable name, no quotation marks are required.

When specifying a specific filename, you must use the Macintosh delimiter (:) or the Windows delimiter (\ ) between folders in the pathname.

Creating a new file does not automatically open it. You must open it before you can read or write data to the new file, just as you did in the "Opening a File by Using the FileIO Xtra" exercise.

For Macintosh users, after you create and open the new file, you can tell the Finder the type of file that you want it to be. This enables your file to have an icon of the right type, which, when double-clicked, will launch the appropriate application. You use another FileIO Xtra method called `setFinderInfo` to do this. The syntax for this method is:

```
setFinderInfo(instance, "fileType creator")
```

The `fileType` parameter is replaced with a four-character code representing the file type (such as `TEXT`, `APPL`, or `PICT`). The `creator` parameter is replaced with a four-character code representing the application that created the file (such as `XCEL` for Excel, `ttxt` for TeachText or SimpleText, and `MSWD` for Microsoft Word). Using these codes has no effect on a Windows system.

The four-character codes for the file type and the application that creates the file are case sensitive!

Table 25-3 lists a few of the codes that you can use with the `setFinderInfo` method.

Table 25-3
File Types and Creator Codes for the setFinderInfo Method

File Type Code	*Creator Code*	*Source Application*
`MV07`	MD00	Director DIR
`TEXT`	ttxt	SimpleText text document
`PICT`	ttxt	SimpleText PICT document
`ttro`	ttxt	SimpleText read-only (used for Read Me documents)
`W8BN`	MSWD	Microsoft Word 98 document
`XLS8`	XCEL	Microsoft Excel 98 document
`TEXT`	CARO	Acrobat document
`TEXT`	MOSS	Netscape document

The FileIO Xtra also includes a `getFinderInfo` method that returns file type and creator codes for an existing file. Both methods (`getFinderInfo` and `setFinderInfo`) only work after a file has been opened using the `openFile` method.

As mentioned earlier with file types, the Find File application can be used to find the creator as well.

Writing a disk file

After you have created and opened a new file, the next logical step is to write something to it. The FileIO Xtra includes two similar methods for accomplishing this task. After using the keyword (either `writeString` or `writeChar`), you identify the instance of the FileIO Xtra in use, and then specify the text string to write.

Here's the syntax for the `writeString` method:

```
writeString (instance, theTextString)
-- or dot syntax
instance.writeString(theTextString)
```

In this method, a text string is substituted for `theTextString` parameter and written to the open file. The string written to the file is a null-terminated string. For instance, to write a score of 12 to an instance of the FileIO Xtra called `diceThrow`, you can use the following instruction:

```
writeString(diceThrow, "12")
-- or dot syntax
diceThrow.writeString("12")
```

The other option is to write a single character to the current pointer position in the file by using the `writeChar` method. The syntax is:

```
writeChar(instance, theChar )
-- or dot syntax
instance.writeChar(theChar )
```

To use either of these write operations, the file must be open in write or read/write mode. All writes are to the current pointer position within the file.

Knowing the current position within a file

In the foregoing discussion of reading and writing to a disk file, we've used the concept of *current position*. If you are familiar with database management, you know that applications use pointers to identify positions within a file. A pointer points to a specific record, the beginning of a file, or the end of a file. The pointer indicates the current position within the file. That location is where you can read or write

data to the file. The same principle applies when you use the FileIO Xtra to read or write to a disk file. The Xtra includes several methods that enable you to determine or set the current position.

To establish (or set) the file pointer position for the current open file, use the following method syntax:

```
setPosition(instance, position)
-- or dot syntax
instance.setPosition(position)
```

To identify the current pointer position in an open file, use the `getPosition` method. It returns an integer representing the number of bytes from the beginning of the file to the current pointer position. It uses the following syntax:

```
getPosition(instance)
-- or dot syntax
instance.getPosition()
```

The `getLength` method returns an integer that is the length of the file in bytes. This method's syntax is as follows:

```
getLength(instance)
-- or dot syntax
instance.getLength()
```

Each of these position-related methods works only on an open file.

By using a combination of these methods, you can write to the end of an open file. This enables you to append data rather than overwrite existing data. When a file is initially opened, the current position is the beginning of the file. You can start reading or writing at the end of a file by using the following combination of the `setPosition` and `getLength` methods:

```
setPosition(instance, (getLength(instance)))
-- or dot syntax
instance.setPosition(instance.getLength())
```

In the following exercise, you use score.dir, a partially complete Director movie. In its current state, the score.dir movie includes a bitmap cast member called Column (cast member 1) and a frame script (to cycle the movie on frame 1). To complete the movie, you add a button that saves text (a name and score) to a disk file. The user types the name and score in an editable field.

You can find the score.dir file on the companion CD-ROM in the folder EXERCISE:CH25 (EXERCISE\CH25).

Saving a File by Using the FileIO Xtra

1. Open the score.dir movie in Director.
2. Select the Text tool on the Tool Palette.
3. Drag to build a Text member centered on the column (center Stage).
4. With the text still selected, type **Save**.
5. Set the `foreColor` of the text to white and the background to black (so you can read the text).
6. Set the ink for the sprite to Background Transparent. Center the type as well (see Figure 25-10).

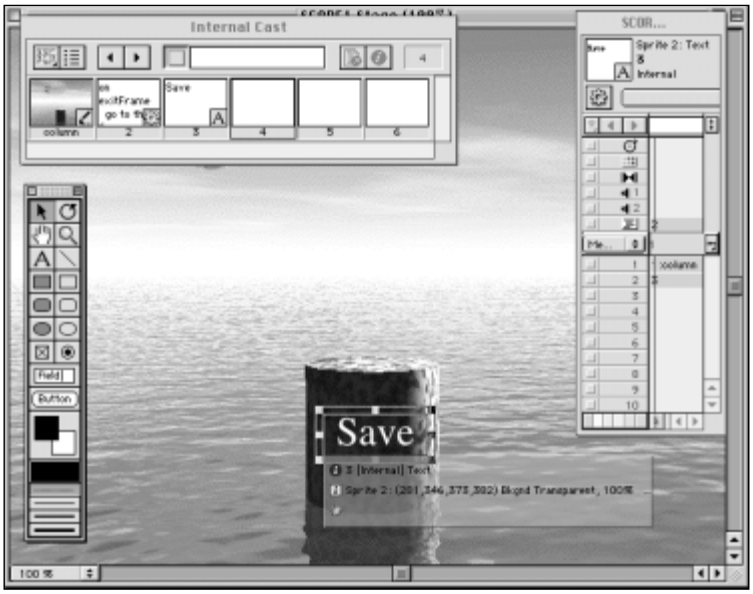

Figure 25-10: A button caption added to the column

7. Create the behavior shown in Figure 25-11 and attach it to sprite 2 (the text sprite).
8. The behavior creates a new instance of the FileIO Xtra, gets a filename from the user, tests it to make sure the user didn't cancel, opens the file, and writes the text from the member named Top Score; if it is running on a Mac, it sets the file type and creator information, and then closes the file.
9. Select the Text tool on the Tool Palette.

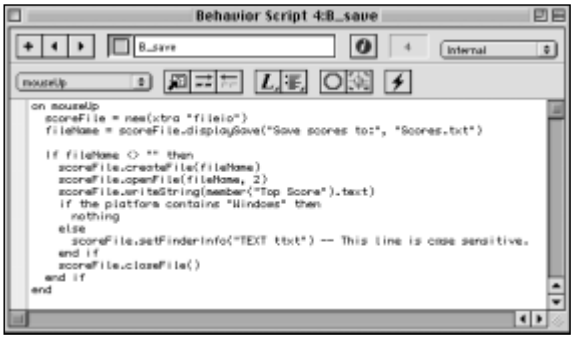

Figure 25-11: The save behavior

10. Drag in the white area at the horizon, to create a new field, as shown in Figure 25-12.

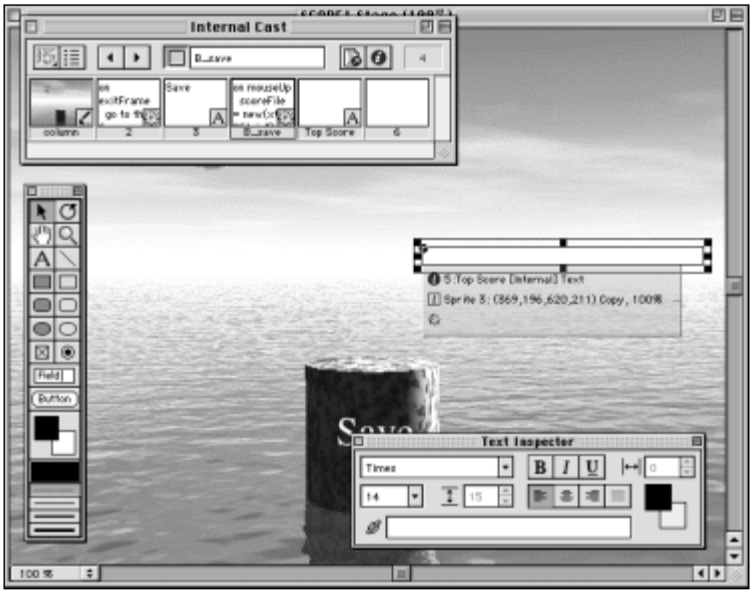

Figure 25-12: The text member for the scores

11. Make the text black on white and name the member **Top Scores**.

12. Select the newly created text member in the Cast window and open the Cast Member Properties dialog box. Click Options and then select Options to make this text member editable. Deselect Use Hypertext Styles

13. Save the movie as **score1.dir**, rewind it, and play it.
14. In the Top Scores text sprite, type your name followed by a number for your score.
15. Click Save on the column.
16. When the Save Scores To dialog box appears, it defaults to saving scores.txt in the same folder in which you have placed the scores1.dir movie. Although you can save the file elsewhere by typing in a different filename, in this case, leave the filename as is and click the Save button to save your name and score. (If a file with this name already exists in the target folder, you will be warned by the operating system, in which case you can either replace the file or cancel the Save operation.)
17. Stop the movie, go to the folder from which you played the score1.dir movie, and locate the scores.txt file. Double-click it to start SimpleText (Macintosh users) or NotePad (Windows users). When the file opens, you should see the text you typed on the Stage in the field.

Summary

Some of the things that you learned about Xtras in this chapter are as follows:

- Xtras are extensions to the functionality of Director. These code modules are developed by third-party vendors to enhance Director.
- The Macromedia Open Architecture (MOA) supports five types of Xtras: transition Xtras; cast member Xtras; scripting Xtras; tool Xtras; and importing Xtras.
- The `showXlib` command returns a list of available installed Xtras.
- Each Xtra has its own set of methods that you can access. You can display the list of methods by using the `interface` command.
- You use the FileIO Xtra to create, open, close, read, and write to an external disk file from within a Director movie.

In the last chapter of the book, we look at a feature that is more advanced than Xtras: Imaging Lingo!

✦ ✦ ✦