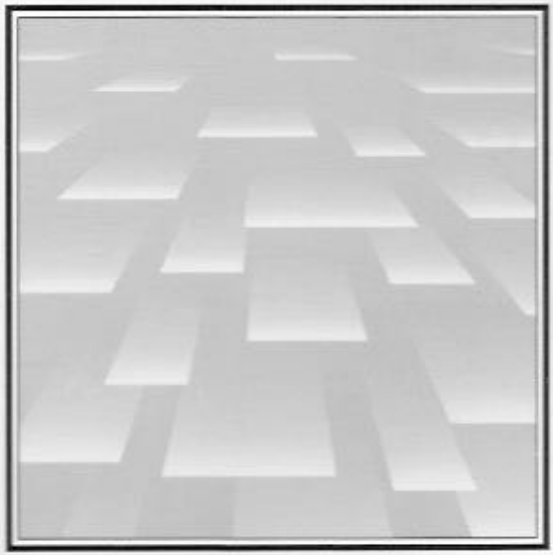
HEIZER SELECT SERIES

WindowScript™



The Interface Design Studio

Distributed by

Heizer Software

Copyright Notice

CREDITS

Project Team

Leonard Buck	Developer
Brian Molyneaux	Product Manager
Rob Terrell	Documentation
Alan Pabst	Testing
Ray Heizer	Publisher

Acknowledgements

Thanks to all our beta testers, especially: Bryan McCormick, Danny Goodman, Steve Michel, James Beldock, Dean Wette, Peter Cleaveland, Dan Shafer, George Pytlik, Peter Nagel, Jeff Winkler, Michael Holm.

Extra special thanks to: Ron Therrien, Robertson Reed Smith, Tom Hammer, Andrew Meit, Sanford Selznick.

Thanks also to all our Dialoger Professional customers. Your support and comments made WindowScript possible.

TABLE OF CONTENTS

Chapter I
INTRODUCTION

WindowScript is a powerful interface design environment for HyperCard that is as easy to use as HyperCard itself. WindowScript gives you the power to create dialogs, windows and floating palettes quickly and easily. And WindowScript lets you create scripts in HyperTalk to customize the behavior of these windows.

Before WindowScript, creating dialogs and floating windows required extensive programming in other computer languages. A programmer would have to create these windows with Pascal or C. The rich interface design features of HyperCard were lost.

This is where WindowScript comes in. WindowScript lets you create external windows and write the scripts that control them. WindowScript provides you with windows that have powerful features already built into them. WindowScript offers a powerful set of objects that can be created to fill these windows, and a rich set of properties for those objects, making the design of custom interfaces a snap. And all this takes place within HyperCard.

This manual assumes a certain familiarity with HyperCard, as well as a working knowledge of XCMDs and XFCNs. If you're new to HyperCard, some of the concepts presented here may be unfamiliar. Don't panic. Read what you can, and refer to your HyperCard manual. WindowScript is easy!

Requirements

WindowScript requires, firstly, HyperCard. Specifically, HyperCard version 2.0v2 or later. And it's sort of a given that you'll have a Macintosh capable of running HyperCard, too. This means at least a Mac Plus.

You'll also need about 800k free on your hard drive. Two MB of RAM are recommended and may be required depending on your specific system configuration.

Installation

Installation is a two-step process. First, copy WindowScript from the floppy disk to your hard drive; then, install it into your Home stack. The WindowScript stack takes care of the second part for you, but you'll have to do the first on your own.

Your WindowScript package contains:

- This manual
- The WindowScript diskette
- A product registration card

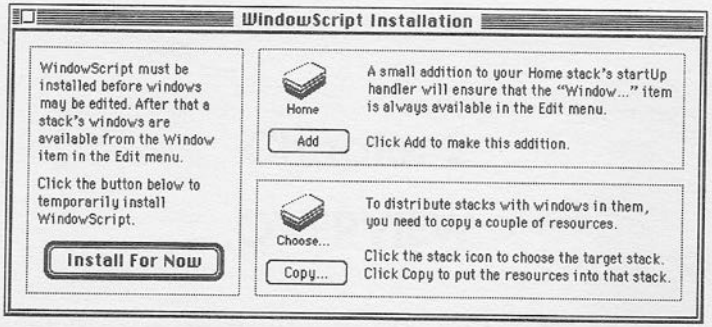
If any of these items are missing—and you can't be missing your manual if you're reading this—please call us at 510-943-7667. Be sure to fill out the registration card and mail it back to us as soon as possible (if you purchased WindowScript directly from Heizer Software you are already registered.) This way, we can always keep you informed of fixes and updates.

To install WindowScript:

1. Insert the disk labeled "WindowScript™" into your floppy disk drive.
2. Double click on the disk icon. Locate the stack named "WindowScript."
3. Drag the stack onto your hard disk.
4. Drag the disk icon into the trash to eject the disk.

WindowScript is now on your hard disk. You may put it into any folder you want.

Now double-click on the WindowScript stack itself. After a moment, the WindowScript stack's window will be visible. Click on the "WindowScript" title card. A card will be displayed describing your installation options. Underneath the help are three buttons. Click on the button "Installation..." and you will see:



Three carefully described choices are presented to you.

- If you plan on using WindowScript often, click on "Add." This will make your Home stack "start using" WindowScript every time you launch HyperCard. This way, WindowScript will always be available.
- If you just want to install WindowScript temporarily, click on "Install for now." WindowScript does use a significant amount of memory, so you may wish to install it only when you need it.
- If you are going to distribute a stack that uses WindowScript, you'll need to add certain WindowScript resources to that stack. The "Copy" button will do so. See the chapter "Distributing WindowScript" for details.

If you're unsure, the path of least resistance is "Install For Now."

While you can run WindowScript in 1000K, we recommend using a HyperCard memory partition of at least 1200k. You can change this by selecting the HyperCard application in the Finder and choosing the Get Info command from the File menu; type "1200" into the "Current Size" field.

Now you're ready to create windows. You're welcome to play around with the sample windows that come with WindowScript. If you want a tutorial to show you the ropes, head for Chapter 3. Be sure to read the next chapter for a quick overview of what WindowScript can do.

Chapter 2
OVERVIEW

WindowScript is an external function (XFCN) for HyperCard that creates windows, dialogs and floating palettes. Windows contain objects, such as push buttons, popup menus and pictures. WindowScript comes with fifteen standard objects; advanced users can create additional custom objects. The window and all of its objects have properties which describe and control their behavior.

A property has a name and a value. For example, the `visible` property has a value of `true` or `false` and governs whether an object is visible. Many properties, such as `visible`, are common to all objects. Some are specific to individual types of objects (e.g., a checkbox object has a `hilite` property which defines whether or not it is checked).

Each object in a window may have a script associated with it. These scripts are automatically invoked when appropriate. While in these scripts, or any time a window is displayed, any property of any object may be determined using the `wsGet` XFCN and changed with the `wsSet` XCMD. Certain objects can also be sent predefined commands with the `wsSend` XCMD (e.g., send `play` to a QuickTime object).

Making a Window

When WindowScript has been installed, a "Window..." item is added to HyperCard's Edit menu. Selecting this menu item brings up a dialog from which you may select an existing window or create a new one. Creating windows is done in the stack in which they'll be used.

Once a window has been opened for editing, a modified tool palette allows objects to be added directly to the window and a property picker windoid allows quick access to most of the properties of each object. After objects have been added to a window, arranged as desired, and had their properties set, the window is saved into the stack for later use.

Using a Window

To use a window, the WindowScript XFCN is called with the name of the window to be displayed. (Alternately, a full textual description of a window may be given.) For example, this command would a open a window previously saved under the name "DateDialog":

```
get WindowScript("DateDialog")
```

and afterwards, the variable `"it"` would contain the item number of the object used to close the dialog along with some other stuff, or empty if the user clicked cancel. (It is possible to configure WindowScript to return additional information; this will be discussed later when we cover the `Params` property.)

How Windows Are Stored

Windows are stored in a stack's resource fork as textual descriptions of type LENS. The LENS resources describe the windows, their objects and their properties. It's basically a complete inventory of all the objects and properties that make up the window. Each window you create will have exactly one LENS resource and, depending on what the window does, may require that other resources be in the stack as well. (A LENS resource is just a template from which a window is created; it isn't used while the window is open, so you can open many windows from one LENS.)

Changing Properties

You should already be familar with properties of HyperCard objects such as the `name` of a button or the `lockText` of a field. WindowScript properties perform similar functions and are used in much the same way. Windows and objects are manipulated through their properties. Properties are the individual features of objects that set them apart: location, name, font, color and so forth.

Properties are accessed through HyperTalk statements. For example, to learn if a window has a title bar, you could type:

```
put HasTitleBar of window "Untitled"
```

into the message box, and HyperCard would return either `true` or `false` as appropriate. All the properties of a window are accessible in this manner, except for modal dialogs, which cannot be accessed in this fashion. (See below for details.)

A slightly different syntax is required to get at an object's properties. To refer to a property of an individual object, the property name is preceded by the object's name and an underscore. Say that we have an editable text item named Bob. To get the text that has been typed into Bob, you could say:

```
get Bob_Text of window "Untitled"
```

and HyperCard would put the contents of the text item `Bob` into the container it. Likewise:

```
set Bob_Text of window "Untitled" to "Nothing Here!"
```

would set the text for Bob to "Nothing here!"

Changing Properties Using Item Numbers

While WindowScript will allow two or more word names for objects, HyperCard demands that object names be just one word. Also, the above won't work if you want to leave an object unnamed, or if you used the same name for several objects.

Instead of using the object's name, you can use an "i" and the object number and "_" as a reference. For example:

```
get i3_Disabled of window "Example"
```

would get True or False, depending on whether the item was disabled or not. If the name of your property is not a single word, you must refer to the object by number when using get or set.

Changing Properties Using wsSet and wsGet

Rather than trying to remember all the special cases, you can use WindowScript's built-in externals, wsSet and wsGet, to do the same thing as HyperCard's set and get. The nice thing about wsSet and wsGet is that they work all the time, for all window types, and for all objects, regardless of name.

You can also use expressions in the property name using these externals. Modal windows—those windows whose `style` is Dialog—can only be manipulated using wsSet and wsGet. You use them like this:

```
get wsGet("Example","OK","Disabled")
wsSet "Example",1,"Disabled",TRUE
```

which are the same as:

```
get OK_Disabled of window "Example"
set i1_Disabled of window "Example" to true
```

These externals, however, are slower than HyperCard's Get and Set commands. In general, you should use them when it can' t be avoided: when your object names have more than one word, or when you are manipulating the objects of a modal window.

Chapter 3
MAKING A DIALOG

Let's begin by using WindowScript to create a simple dialog box. WindowScript makes the creation of such dialogs a simple matter. Hypercard already provides a basic dialog box through the answer command. We'll create a dialog that goes a little further than that: we'll create one with a custom error message, and with two buttons.

To get started:

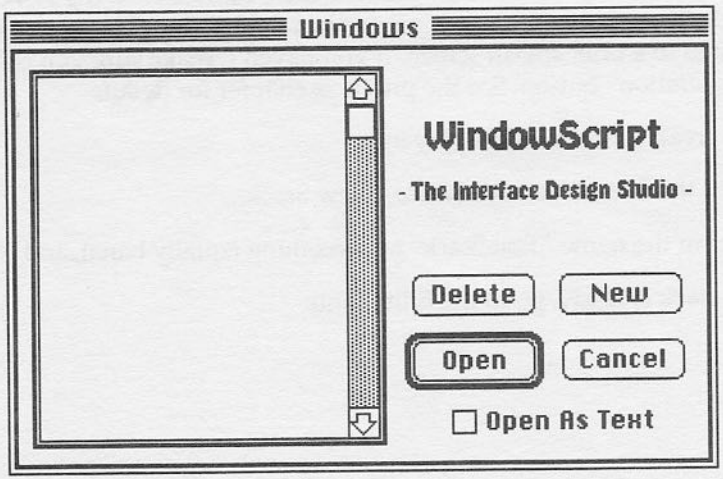
- If you've just installed, great.
- If you haven't yet installed WindowScript, double-click the WindowScript stack. The stack will open. If you've installed in your Home stack, you'll be treated to a brief splash screen. If you haven't, make sure you click on the "Installation" button. See the previous chapter for details.

First, let's create a new stack to play in.

- Go to the File menu and choose "New Stack..."
- Type in the name "Test Stack" or something equally banal, and hit Return.

When the stack is ready, go to the Edit menu:

You'll notice a new item at the bottom of the menu: "Window..." This menu item is like the "Icon..." item above it, only different: it takes you into an edit mode for windows. When you select it, you are presented with this dialog:

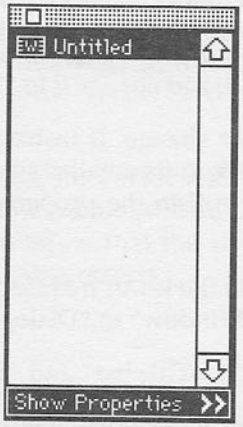


From this dialog, you choose the window you want to edit. You can also create a new window, or delete an existing window. The list is empty because we haven't created any windows in this stack yet. Let's take care of that now:

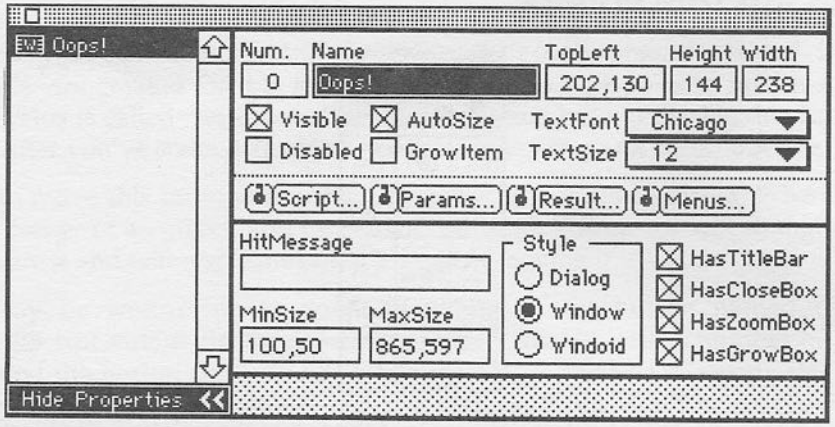
- Click on the "New" button.

After a moment, a new window will open, along with two floating palettes. These palettes are what you'll use to design your windows. One, the tool palette, is similar to HyperCard's tool palette; the other is a unique window called the property picker.

The Property Picker



This is the property picker. It is used to display and set properties of objects and the window as a whole. It is currently in its collapsed state. To see the rest of it, click on the "Show Properties" bar at the bottom. It will zoom out to display:



These are the properties of the window. If it's a little confusing, don't worry for now; we're only going to deal with a few of these things at the moment.

The collapsed form of the property picker contains just the object list. Every object in the window is listed in order, with the window itself being first. To view the properties of a particular object, click on the corresponding item in the list. The right-hand panel will then change to describe that object.

For now, let's just change the name of this window:

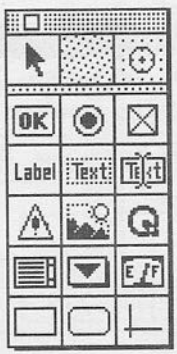
- Open the property picker, either by clicking on the "Show Properties" button or by double-clicking on the window in the item list.
- Tab over to the name field, and change it to "Oops!"

Press tab to complete the name change. If instead you hit return or enter, the property picker will shrink back to its smaller size. This is especially handy on smaller screens. While you're still in the property picker, change the window style to "Dialog":

- Using the radio buttons in the lower middle of the property picker, change the window style from "Window" to "Dialog".
- Uncheck the properties "HasTitle bar" and "HasClose Box".

This will make the final product act as a dialog box: the window will come to the front and demand user attention before continuing. Now hit return or enter, or click on the "Hide Properties" button, to collapse the property picker.

Using the Tool Palette



Like the HyperCard tool palette, the tools at the top of the palette edit objects; the tools at the bottom create objects. This palette also appears under the tool menu. You can select a tool from the menu, or you can tear the menu off and place it anywhere on the screen, just like the HyperCard tool palette.

There are fifteen object tools in the tool palette. To see a description of all the tools, see Chapter 5. We're going to concern ourselves with just one tool for now:

The Button tool, which is used to create standard Macintosh buttons.

Either click on this tool in the palette window, or select it from the Tool menu.

You may notice that the entire menu bar has changed. WindowScript has its own set of menus, which are active whenever a WindowScript window is in front. If you need to get back to the HyperCard menus, click on the card window.

When you click on the Button tool, the cursor will change to a crosshair cursor. You'll also notice that the window becomes filled with a grid of dots. This grid serves two purposes: to let you know that the window can be edited, and to provide an easy reference for aligning objects.

Creating a Button

Now create a button in the "Oops!" window:

- Click somewhere on the window, drag down and to the right a ways, and let go.

That's how easy it is to create objects!

Dragging the selection cursor in the window creates a gray rectangle—this is the rectangle your new object will fill. When you let go, the object is created. If the rectangle you created for it is too large, it will snap back so as to hold just the object. (This is called "autosizing" and we'll discuss how to disable this feature later.) After you've created an object, the selection cursor will become active.

You can move this button around by dragging it to a new location. When you grab a corner of an object, you can resize it as well. The cursor will change to a resize arrow and will resize the object instead of moving it.

You may be wondering to yourself, "Why is this button named OK?" WindowScript automatically names the button created with a number of 1 to "OK" and the button created with a number of 2 to "Cancel." Usually this is a convenience, especially if you're creating dialogs. If not, don't worry, you can easily rename them.

If you left the property picker open, you'll notice that the button has a property called "AutoClose" turned on. This means that if the button is clicked, the

window will be closed. Also the property "DefaultItem" is on; the default button is the button with the thick border, and pressing return or enter is the same as clicking the default button.

Another But ton

Let's create the "Cancel" button:

- Click on the button tool. Move the cursor to the window "Oops!" and drag down and to the right.

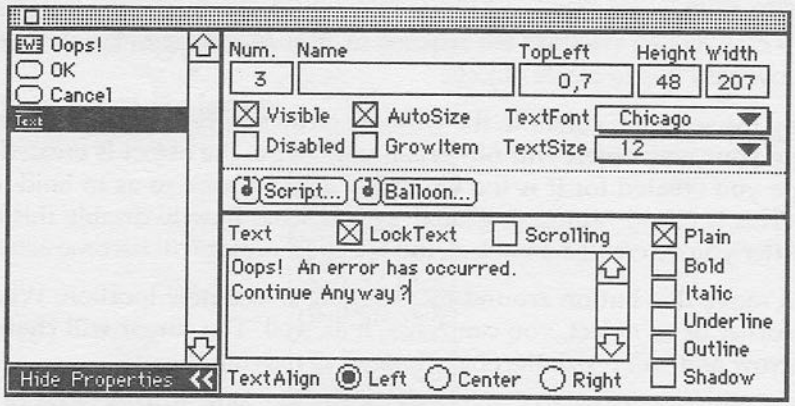
You'll notice that this button is, as expected, automatically named "Cancel." If you left the property picker open, you'll also notice that the checkbox "CancelItem" is set to true for this object. This means, when this object is clicked, it will cancel the window.

Now create some text for this dialog:

- Click on the static text tool 

. Click in the window "Oops!" and drag out a rectangle.

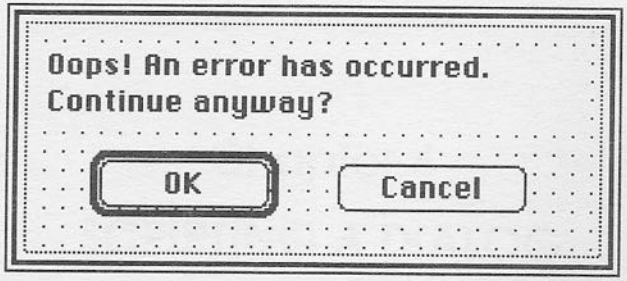
This will create a text object in the window. To fill it with text, type the text you want into the property picker:



- Type "Oops! An error has occurred. Continue anyway?"

When you hit tab, enter or return, the text is accepted and the object changed. If you typed enter or return, the property picker will shrink back to its smaller size.

Now the window is complete!



Save and close the window from the File menu. To try your creation out, type this from the message box:

```
WindowScript("Oops!")
```

and your dialog box will appear. Notice what happens in the message box after you dismiss the dialog with the "OK" button: the name of the button you used to close the window is returned. WindowScript always returns the name of button that closed it, or returns empty if the button was the cancelItem.

That's how easy it is to create simple windows. Many of the things you'd want a window to do are automatically taken care of. Of course, there are times when you want something other than the expected to happen; then you'll need to script your windows (hence the other half of the name "WindowScript"). To learn about scripting, go on to the next chapter.

Chapter 4
SCRIPTING A DIALOG

Let's add a little complexity to our dialogs. WindowScript gains much of its power, and half of its name, from its scripting capabilities. You can create scripts for every item in a dialog; these scripts are executed when the object is "hit," or clicked. You can also create scripts for the window as a whole, to respond to the kinds of events that happen to windows. You can even write scripts to run when the text in a WindowScript field is changed.

But first, we're going to explore the property picker. We'll start with the simple dialog created in Chapter one.

- Open the window "Oops!" with the file menu's "Open Window" item. If that command is not available, use the "Windows..." item from the Edit menu.

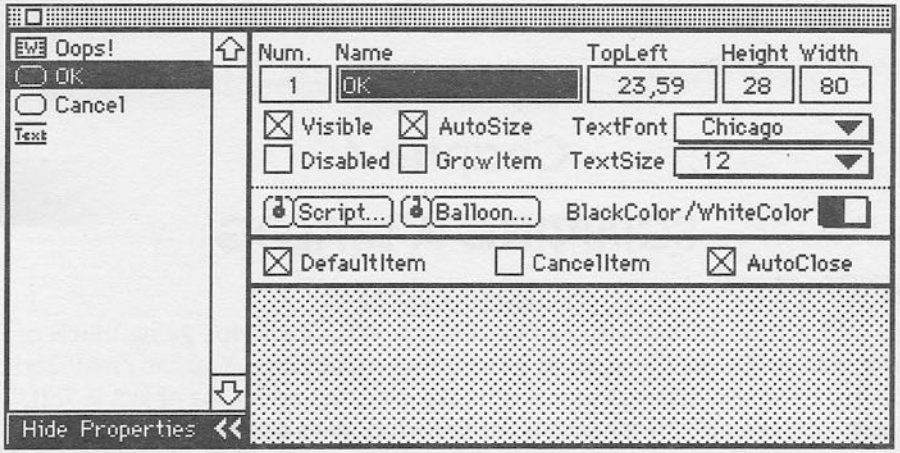
The window will open. If the selection cursor is not active, make it so using the Tools window.

We want to play with the "Okay" button. Using the selection cursor, try to resize the button and make it larger:

- Resize the button by dragging from the lower-right corner to an even lower point.

Nothing happens, right? This is because the button has a property, called AutoSize, that controls its sizing. To resize this button, we'd have to turn the AutoSize off. And we do that in the property picker.

- Double-click on the "Okay" button.



Don't let the array of options overwhelm you. All of these properties are covered in detail in later chapters; for now, we'll only deal with a few of them. Feel free to experiment as we go along. (For more on the property picker, see Chapter 5.)

Changing Properties

The property panels appear to the right of the object list. These panels are different for each object, and change depending on which object has been selected. Their contents represent the properties of that object. Changes made here are reflected immediately in the window. When entering information into a text area, changes take effect when you tab out of a field or hit enter.

Let's turn the Autosize option off.

- Click on the checkbox named "Autosize."

This change takes place instantly; the button will no longer AutoSize. Let's test this:

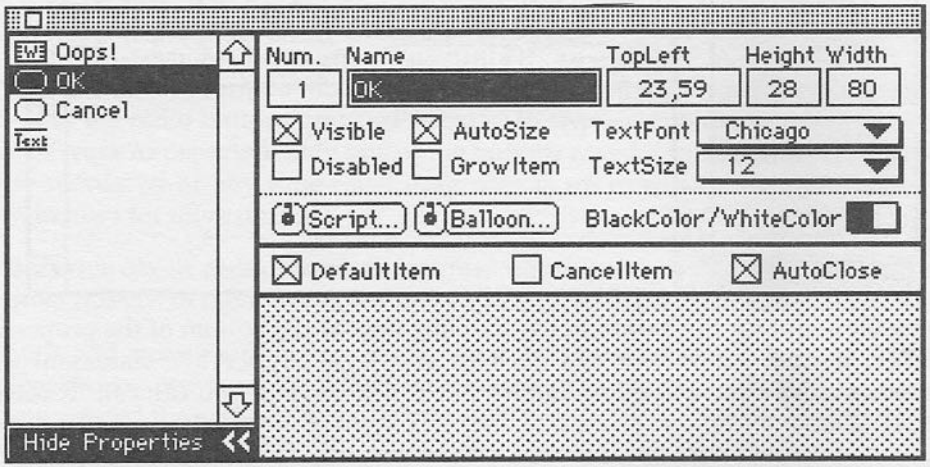
- Resize the button as before, either smaller or larger.

The button will shrink or grow as appropriate. You may notice that the button looks lousy at whatever size it's now at. (You may not, and that's perfectly okay.) The Autosize property is designed for just this purpose, to keep objects at their best-looking size. Apple Computer has spent years determining the optimum look for the various standard objects that appear in Macintosh applications. WindowScript conforms to Apple's recommendations wherever possible through the autosize property.

- Turn the Autosize property back on.

The button will return to its natural size. For buttons, you'll probably always want to have the AutoSize option on.

Scripting a Button



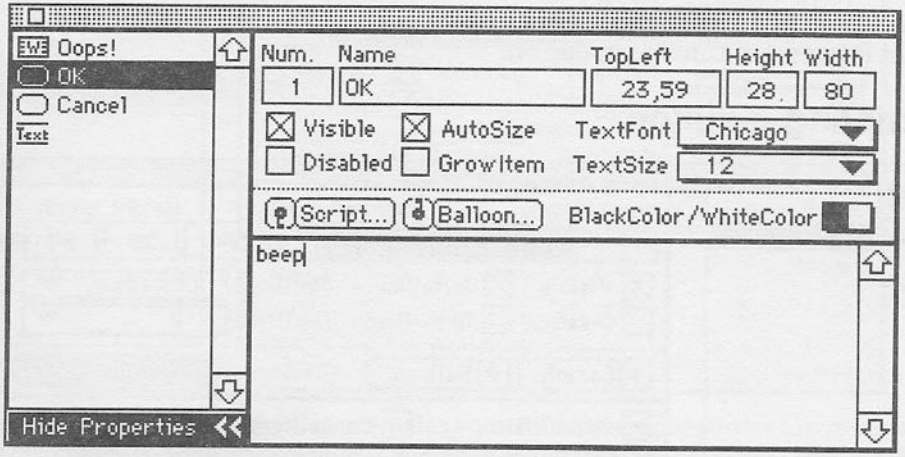
There is, however, a property we will want to turn off. The AutoClose property, when on, closes the window when the object is hit. This is handy for most buttons, where you want as much as possible done for you automatically. But since we're writing our own script—and want to do the messy work ourselves—turn it off.

- Uncheck to the "AutoClose" property for the button "OK."

Now we're ready to write a script for the button. Look at the middle of the property picker. You'll see an odd double button, known as a "Siamese" button: one half is a latch, the other is the "Script" button. To see the object's script, click on the "Script" latch—not the button. The "Script" button will open a scripting window. The latch, which is just beside the button, will let us write the script in the lower portion of the property picker. For now, that's all we'll need.

- Click on the "Script" latch.

The script property will be displayed in the lower portion of the property picker:



Type the script for the button in the scrolling field at the bottom of the property picker. In this script you can execute any legal HyperTalk statement or statements. You can call other handlers or functions, and you can call XCMDs and XFCNs.

We'll make a simple script first, so type this into the script field:

```
beep
```

Simple and to the point, right? Switch to the arrow cursor from the Tools menu, and click "OK".

You should hear a beep. Wasn't that easy? Now let's add a little more to the script. In complete defiance of the Macintosh Human Interface Guidelines, let's confirm our dialog with another dialog. Add another line to the script:

```
beep
Answer "I will now attempt to continue."
```

Switch to the arrow cursor and click "OK". You should hear a beep, then see the answer dialog pop up. When you dismiss that dialog, the Oops! dialog is still there. Close that dialog now. Add another line to the script:

```
beep
Answer "I will now attempt to continue."
wsSend "Oops!",0,"close"
```

The syntax is like this: we name the window that contains the object, the object we want to send the message to—in this case it's 0, the window itself—and the message, "close".

Now try it. The answer dialog will open with its message, and when you click"OK", both dialogs will disappear.

Radio Buttons and Check Boxes

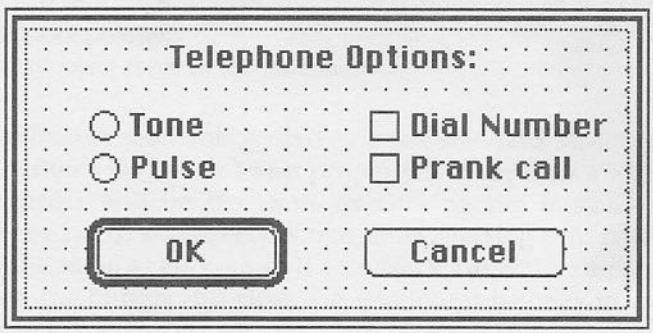
One of WindowScript's nicer features is that radio buttons require no scripting. All of the radio buttons grouped—that is, in sequential order in the object list—will work as expected: only one of the buttons may be on at a time. (The list can be reordered at any time—just drag objects up or down. See the chapter on Windows for more details.)

But what about check boxes? Sometimes turning on one checkbox should affect other objects. In these cases, you'll need a script to make it happen.

Let's add some more objects to the window. Make two radio buttons and two checkboxes, one of which will affect the other. Create these objects the same way you created the earlier objects—if you need help, refer back to the earlier chapter.

- Create two radio buttons.
- Create two check boxes.
- Change the text to read: "Telephone options:"

Name the radio buttons "Tone" and "Pulse" and the check boxes "Dial number" and "Prank call". Your window should look something like this:



Switch to the arrow cursor and try the radio buttons. They should work as advertised. Try the checkboxes; they work independently of each other. To make one checkbox dependent, one will need a script. Switch back to the selection cursor:

- Double-click on the "Prank call" button.

It will appear in the property picker. Click on the Script latch so we can edit its script.

- Type the following into the check box's script:

```
wsSet wdName, "Dial number", hilite, objValue
```

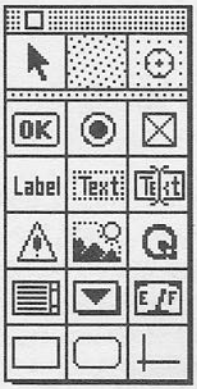
What this script says is, set the hilite of the object named "Prank call" to whatever the hilite of this object—the object clicked on—is. Switch back to the arrow cursor and try 'em out; clicking on "Prank call" will hilite "Dial number", but not vice versa.

Moving On

That should be enough to get you started in the wild world of scripting. As you've seen, WindowScript provides simple scripting power to your objects. Look at all the example windows, and examine their scripts—that's probably the best education in WindowScripting. Later we'll discuss the Script Assistant feature of WindowScript which will help you write scripts.

Chapter 5
WINDOWS

The Tool Palette



Like the HyperCard tool palette, WindowScript's tool palette is divided into two parts. The tools at the top are editing cursors; the tools at the bottom are objects. This palette also appears under the tool menu. You can select a tool from the menu, or you can tear the menu off and place it anywhere on the screen, just like the HyperCard tool palette.
There are fifteen object tools in the palette:

The **Button** tool. This is used to create standard Macintosh buttons. If a button's `defaultItem` property is true, then a Return or Enter keypress will be the same as clicking the button. If a button's `cancelIte m` property is true, then pressing command-period will be the same as clicking it. If a button's `autoClose` property is true, then the button will close the window it belongs to. It executes its script upon `mouseUp`.

The **Radio Button** tool. This is used to create radio buttons. All radio buttons in sequential order in the object list will be grouped, that is, selecting one will deselect the others. The `hilite` property tells you if the button is on or off. It executes its script upon `mouseUp`.

The **Checkbox** tool. This is used to create checkboxes. The check is controlled by the `hilite` property. It executes its script upon `mouseUp`.

The **Label** tool. This is used to create short labels. For text that requires user interaction, use one of the Text tools below. The label displayed is the same as the `name` of this object. It executes its script upon `mouseUp`.

The **Static Text** tool. This is text that the user cannot change. It executes its script upon `mouseUp`.

The **Editable Text** tool. This is text that the user enters from the keyboard. Both text tools create the same object—a text object—but their `lockText` property differs. The text displayed is contained in the `text` property. It executes its script upon `closeField`.

The **Icon** tool. For displaying icons and Finder icons. You choose the icon to be displayed with the `name` property; match the name of the object to the name of the icon resource you want to display. If you leave the `name` of an icon object blank, it will display nothing. Icons can behave like buttons; the `logic` property controls how they behave. Icons execute their scripts upon `mouseUp`.

The **Picture** tool. For displaying pictures in the window. You choose the picture to be displayed with the `name` property; match the name of the object to the name of the picture resource you want to display. If you leave the `name` of a picture object blank, it will display nothing—this can be useful for transparent buttons. Pictures can also be divided into a grid of buttons. The selection logic of these buttons is chosen with the `logic` property. Pictures execute their scripts upon `mouseUp`.

The **QuickTime** tool. For playing QuickTime movies. The movie file to be played is set in the `file` property. If the `scrolling` property is

true, then the movie will use the standard QuickTime controller. If the `wantFocus` is true, then the movie will accept Cut, Copy and Paste commands from the Edit menu. You can play the movie by clicking on the play button in the controller (if it's available, that is, if the `scrolling` property is true) or by sending a `play` command to the object. It executes its script upon `mouseUp`.

The **List** tool. For displaying all kinds of lists. The list items are contained in the `text` property, each separated by a return. If the `keyScroll` property is true, then pressing a key will select the first item that starts with that key. Arrow keys can be used to navigate the list. There are many different list display styles available—see the Notes Chapter. Lists execute their scripts upon `mouseUp`.

The **Pop-up Menu** tool. This pops up a menu. The `text` property contains the items to be popped up, each separated by a return. The `titleItem` property determines which object will be hilited as the menu title (usually this is a label). It executes its script upon `mouseUp`.

The **Control** tool. This is used to put Macintosh controls (other than buttons) in the window. Examples of controls include gauges, thermometers, and scrollbars. Specify the control with the `style` property. Advanced users can write their own controls in Pascal or C. It executes its script upon `mouseDown`. (Note: Lists, text, and pictures can have their own scrollbars. You don't need to use a control object for these.)

The **Box** tool. For drawing boxes. This is useful to set off a group of radio buttons.

The **RoundRect** tool.

The **Line** tool.

To create a new object in the window, select the appropriate tool and drag out the desired location for the object in the window. When you release the mouse button, it will be created and placed.

Most objects will snap to a "natural" size—for instance, a button will be resized to the standard button height; a picture will snap to its full size; an icon will snap

to a 32 by 32 pixel rectangle. If you want the object to be a different size, uncheck the "Autosize" property in the property picker.

The **Arrow** tool is used to "Play" a window, that is, to test a window. In the play mode, the window acts the same as it will for a user (except that, technically, it will be in the document layer —a consideration you'll probably never have to worry about).

The **Selection** tool is used to edit windows—whenever you wish to move, resize, or delete an object. When you double-click an object with it, the property picker opens to display that object's properties. (When in select mode, objects may be dragged and resized with the mouse.)

Modifier keys may be used to do the following:

Shift—constrains movement just like HyperCard.

Control—temporarily reverses the snap-to-grid state of the window as defined by the Grid menu item.

Option —clones the selected objects just like HyperCard. If not over an object, option allows you to drag or resize the window as a whole.

When one or more objects have been selected, they may be nudged from the keyboard. Once again, modifier keys may be used:

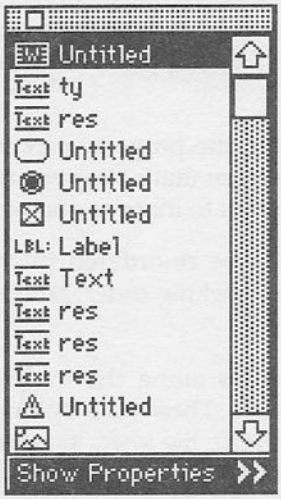
Control—temporarily reverses the snap-to-grid state of the window as defined by the Grid menu item.

Command—resizes the bottom right of the objects rather than moving the whole object.

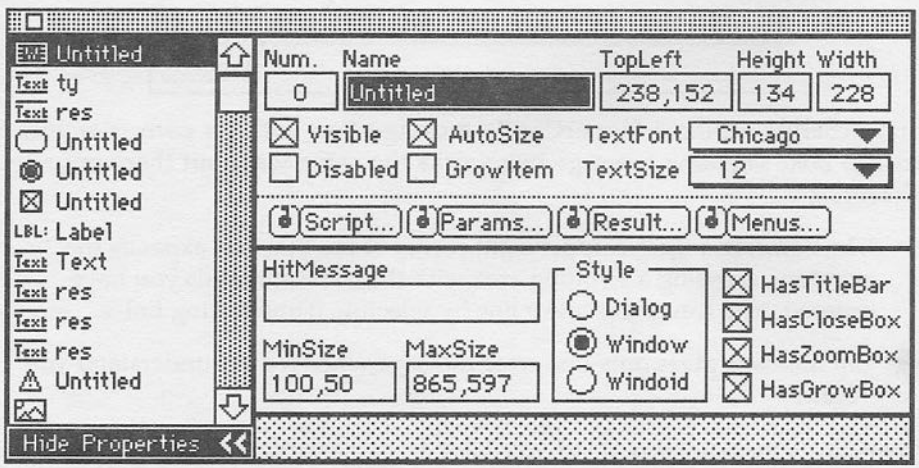
Delete —deletes the selected objects.

If you drag an object but change your mind before releasing the mouse button, simply drag the object to the menu bar. It will not be moved.

The Property Picker



This is the property picker. It is currently in its collapsed state. To see the rest of it, click on the "Show Properties" button at the bottom. It will zoom out to display:



Along the left edge of the property picker is the object list. Every object in the window is listed. Clicking on an object in the list displays its properties in the right-hand side of the window.

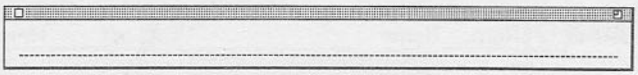
The right-hand side—the side that shows the properties—is divided into two parts. The top half shows properties that every object shares: name, location, font, etc. The lower half contains properties specific to each object. The lower half will change for different objects; the upper half will always display the same properties.

When you hit Return or Enter, the property picker shrinks back to its smaller size. This way, if you have a screen that's strapped for pixels, the property picker won't be in your way. If you want to move to another field, hit tab instead.

Objects may be re-ordered in the list by dragging. This affects their stacking order, tabbing order and radio button grouping.

There are four Siamese buttons along the middle of the window: Script..., Params..., Result..., and Menus.... These odd-looking buttons work two ways. If you click on the latch (on the left), the lower portion of the property picker will become an edit field, so you can edit the contents of that property directly. If you click on the button, a separate editing window will open, allowing you to edit the property in that window. See the Scripting chapter for more details.

Big Message



WindowScript replaces HyperCard's message box with its own customized message box. The new message box works the same way, but there are a few differences:

- The Zoom Box (in the upper right corner of the title bar) expands the window, showing a scrolling area with the last commands you have entered. You can re-execute a line by selecting it and hitting Enter.
- Big Message gives only one error message when it can't understand you: "Huh?"

HyperCard's message box is still available from HyperCard's menus, and when you put a value, it is put into HyperCard's message box, not into Big Message. It's easy to confuse the two, but there's a simple way to tell them apart: the place where you type in Big Message has a dashed underline, as opposed to HyperCard's gray underline.

The benefit of Big Message is that it provides a simple way to climb back into WindowScript when the tool palette and property picker are gone—just click on Big Message, and WindowScript's menus are back.

Chapter 6
MENUS

When you are editing a window, HyperCard's menus are replaced by WindowScript's own menus. While retaining HyperCard's menu structure and flavor, WindowScript's menus deal solely with creating and editing windows.

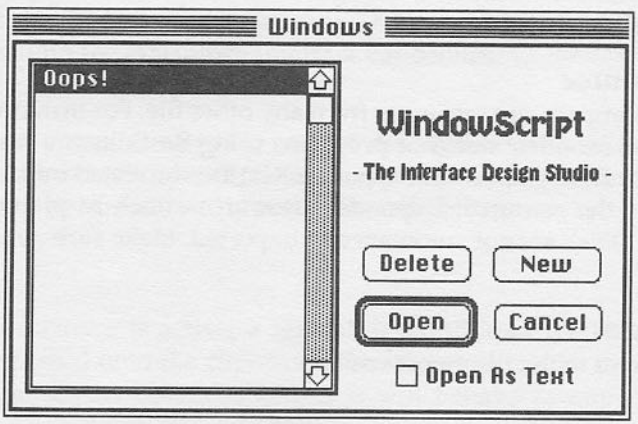
When a HyperCard window is frontmost, HyperCard's menus are active. When a WindowScript window is frontmost, WindowScript's menus are active. WindowScript provides its own message box (accessible from the Go menu) to facilitate switching back and forth: click on whichever message box you want.

File Menu

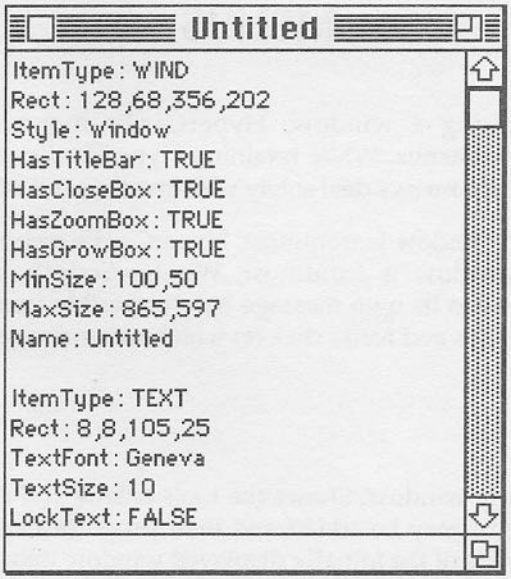
New Window

Creates a new, empty window. Shows the tools palette and the property picker windoid so that items may be added and their properties examined. The exact size, style and contents of the initially displayed window may be altered with the Save Default menu item described below.

Open Window...



Presents the windows dialog. From this dialog, you pick the window to edit. You can also create a new window, or delete an existing window. If you check the "Open as text" checkbox, WindowScript will display the window in text format; that is, it will display the text of the LENS resource that describes the window. This can be useful for creating or modifying many objects at once by simply copying and pasting descriptions.



Close Window

Closes the frontmost window. If changes have not been saved, you are asked if you wish to do so.

Import Resource

Allows you to import any resources from any other file. For instance, if you have created dialogs for other stacks or programs using ResEdit, you may bring them into your stack. Dialogs (resource type DLOGs) are converted into LENS with the same name as the resource. Dependent resources, such as pictures and icons (PICTs and ICONs), are not automatically imported. Make sure you import them, too.

Save Window

Saves all changes to the frontmost window.

Save Window Into...
Allows a copy of the window to be saved into a different stack.

Save Default
Saves the current window as the default. The default is used whenever a new window is created. It has effect only on the stack you are currently in—unless you do this in the WindowScript stack itself. This is useful if you will be creating a number of dialogs or windows that will share common elements such as lines, boxes, or pictures or a standard script for the window itself, so that all windows in a set behave in a standard fashion.

Delete Window
Deletes a window from the stack.

Page Setup...
The standard Page Setup dialog. Controls how windows will be printed by subsequent "Print..." commands.

Print...
Prints the frontmost window. It may be either the actual window or a textual description of that window (when it was opened with Open As Text).

Quit HyperCard
As you might guess, this will close WindowScript and quit HyperCard.

Edit Menu

Undo
Undo is not available for actions taken in WindowScript. It is provided for consistency within the Macintosh Interface Guidelines.

Cut
If the selection cursor is active, a textual description of the currently selected items will be placed onto the clipboard and the items will be removed from the window. If the arrow cursor is active, it will behave as appropriate for the context.

Copy
If the selection cursor is active, a textual description of the currently selected items will be placed onto the clipboard. The items will not be removed from the window. If the arrow cursor is active, it will behave as appropriate for the context.

Paste

If the selection cursor is active, items on the clipboard will be placed into the window. If a picture is on the clipboard, it will be added to the stack as a resource and to the window as an object. Likewise, icons, color icons and Finder icons will all be added to both the stack and the window if they are on the clipboard. If the arrow cursor is active, it will behave as appropriate for the context.

Select All

If the selection cursor is active, all the items in the window are selected. If the arrow cursor is active, it will behave as appropriate for the context.

Go Menu

All open WindowScript windows are listed. Select one to bring it to the front.

Message

Alternately shows and hides the WindowScript big message window. This is similar to the regular message box, but has a zoom box. If you zoom it, it will enlarge to show the last commands executed in a scrolling list.

Next Window

Sends the frontmost window to the back and activates the next window.

Tools Menu

Like HyperCard, the Tools menu is a palette of useful things, and can be torn off the menubar. There are two types of tools:

Cursor Tools

The top portion of the tool palette offers two types of interaction with the window being edited. The arrow cursor causes the window to behave "normally"—just as it will when you invoke it directly with WindowScript. The selection cursor allows you to select items in the window and to drag them around.

Object Tools

The bottom portion of the palette contains item tools. To create a new object in the window, select the appropriate tool and drag out the desired location for the object in the window. When you release the mouse button, it will be created and placed. Once an object has been created, the Selection cursor is automatically chosen. For more information, see Chapter 5.

Objects Menu

Show Info/Hide Info
Alternately shows and hides the information about the properties for the object selected in the property picker.

Bring Closer
Moves the selected items in front of the other objects in the window.

Send Farther
Moves the selected items behind the other objects in the window.

Grid
Either turns on or turns off the grid that covers the window being edited. When on, objects will "snap to" the grid position, making aligning objects a breeze.

Align Lefts
Align the left edges of the selected objects.

Align Tops
Align the top edges of the selected objects.

Align Rights
Align the right edges of the selected objects.

Align Bottoms
Align the bottom edges of the selected objects.

Duplicate
Duplicates the currently selected items.

Font Menu

This menu allows you to quickly change the textfont property of items selected in the window being edited. When the arrow cursor is active, this menu will allow you to set the font of the selected text, including popups and buttons.

Style Menu

If the selection cursor is active, this menu allows you to quickly change the textsize and textstyle property of items selected in the window being edited. In play mode, this affects the type in text objects.

Color Menu

This menu allows you to quickly change the whitecolor and blackcolor properties of the items selected in the window being edited. BlackColor is the color that will be used to draw the parts of an object normally drawn in black, such as the border and text of buttons; whiteColor is the color used to draw the white portion of objects, such as the white space inside a button. (For more information, see Chapter 7). Selections from this menu normally affect the blackcolor of the selected items. To change the whitecolor, hold down the option key while making a selection.

Chapter 7
PROPERTIES AND COMMANDS

Properties

Listed below are all the properties WindowScript uses. Not all are available from the property picker. Properties unavailable from the property picker are marked with an asterisk (*). To change or view a property that isn't in the picker, you can use `set` or `get` in the message box or from the script of an object.

AutoClose = [TRUE, FALSE]

This is a property of push buttons. If true, the window will be closed in response to clicking on the button.

- **Advanced Users:** Returning a non-empty value in a button's object script will prevent it from requesting that the window be closed. (You can use the "return" command in object scripts or handlers called by object scripts, just like functions.)

AutoSize = [TRUE, FALSE]

Determines whether or not an object will snap back to its natural size. If true then the bottom right of the object will be adjusted. "Natural" varies, obviously, from object to object. A button's natural height is 20 pixels; an icon's natural height is 32 pixels; a text object's natural size is an even multiple of the font size.

Balloon = [text | name | x]

This is a property of all objects. It is the text which will appear in a help balloon when help has been enabled and the cursor is placed over the object. This property can also be set to the name or ID of a picture resource, which will be displayed in the balloon instead of text. (Balloon help is only available for Macs running System 7, although you can set this property under either System.)

- **Advanced Users:** There is a set of string resources which provide default text for each type of object. These may be changed to better suit your particular application. There is a resource for each object, named "Balloon4Object", where object is the type of object.

BlackColor = [RED, GREEN, BLUE]

This is a property of all objects. On Macs capable of displaying color, it defines the color that should be used in place of black when drawing the object. The color is specified as a RGB value. Be careful when choosing colors for windows that may be used on black and white or less colorful monitors. The colors will map to the nearest color available, or to black if the monitor is black and white.

- **Advanced users:** This property may also be set with an index into the system CLUT id 256. The external function `AnswerColor` may be used to prompt a user for a color with the standard color picker. See Chapter 10 for more details.

CancelItem = [TRUE, FALSE]

This is a property of push buttons. If true, a command-period will be treated as being equivalent to clicking on this object.

- **Advanced Users:** This object has special significance to what is returned for modal dialogs. When the CancelItem closes a dialog, empty is returned as the value from WindowScript—regardless of what value was requested in Result or what value was explicitly set of the window as a whole.

*Commands = [commands]

This is a read-only property of all objects. It returns the names of all the commands supported by that object.

DefaultItem = [TRUE, FALSE]

This is a property of push buttons. If true, pressing the Enter key will be treated as being equivalent to clicking on this object. So will a Return if the cursor is not currently in an edit field with a visible scroll bar.

*Dirty = [TRUE, FALSE]

This is a property of text objects. This property is automatically set to true whenever anything is typed into a field. If true, a hit will be sent when the field is tabbed out of or the user clicks outside the field. If you set `dirty` to false, a hit will immediately be sent if the field had been dirty. If that message returns a non-empty result, the field will remain dirty.

- **Advanced Users:** This is also a read-only property of the window as a whole. In this case, it will return the object number of the dirty edit field, or zero if there is none.

Disabled = [TRUE, FALSE]

This is a property of all objects. If true, then object will appear "grayed out" and will not respond to typing or clicking.

DoubleClickItem = [i]

This is a property of lists. If not zero, a double click on an object in the list will be treated as being equivalent to clicking on object i. Either the actual object number or the name of the object may be specified.

File= [file name]

This is a property of QuickTime movie objects. It is the file name of a QuickTime movie.

Fill = [NONE, WHITE, BLACK, GRAY, LTGRAY, DKGRAY, x]

This is a property of boxes. It specifies the pattern used to fill the contents of the box.

- **Advanced Users:** Other patterns in the system's pattern list may be specified by x, an index value into the system pattern list. See *Inside Macintosh Vol. 1* for more information.

Grid = [x, y]

This is a property of pictures. It specifies how the object is divided into separate cells. For example, Grid: 2,3 would split the object into six cells, two across and three down. A grid may not contain more than 32 cells.

GrowItem = [TRUE, FALSE]

This is a property of any object. When true, the object will be resized whenever the window is resized. Objects whose bottom exactly coincides with the grow object's bottom will also be resized. Likewise, objects whose right side exactly coincides with the grow object's right side will be resized. All objects completely below the grow object will be moved so as to stay below it. Likewise, all objects completely to its right will be moved to stay to its right. Only one object may be specified as the GrowItem.

*Handle = [handle]

This is a property of all objects except box, line and round box objects. It provides access to the low-level toolbox structures used by WindowScript. Never change information in these structures if you know what's good for you.

- **Advanced Users:** You can set the handle of a picture object to a PICT you have somehow created or gotten a handle to. WindowScript will dispose of the handle to the PICT for you when the window is closed.

HasCloseBox = [TRUE, FALSE]
This is a property of the window. It determines if a window will have a close box.

HasGrowBox = [TRUE, FALSE]
This is a property of the window. It determines if a window will have a grow box in the lower right-hand corner. If a grow box is specified, scrollbars will also appear. Since dialogs do not have grow boxes, this property is ignored for dialogs.

HasTitleBar = [TRUE, FALSE]
This is a property of the window. It determines if a window will have a title bar. In the case of a windoid, this property will enlarge or reduce the title bar, not remove it.

HasZoomBox = [TRUE, FALSE]
This is a property of the window. It determines if a window will have a zoom box. Since dialogs do not have zoom boxes, this property is ignored for dialogs.

Height = [x]
This is a property of all objects. It specifies the height of the object.

Hilite = [TRUE, FALSE]
This is a property of radio buttons, checkboxes and controls. It is true if the object is "checked."

- **Advanced Users:** This is also a property of icons when their logic property has been set to Radio or Check.

HitMessage = [message name]
This is a property of the window. It is the name of the handler which will be responsible for interacting with the user. This handler is called by WindowScript whenever anything happens in the window. See Chapter 9 for more information.

*ID = [x]
This is a property of icons and pictures. It is the actual id of the resource to be displayed as the object. It is also a property of the window as a whole—in this case it is a unique identifier for the window. This id changes each time the window is displayed with WindowScript.

*IdleDelay = [x]
This is a property of the window as a whole. It enables windows to request that idle messages be sent to the hit handler (or the window's script, if present). X

defines how often (in ticks, which are 1/60ths of a second) the messages are to be sent.

*ItemList = [list of objects]

This is a read-only property of the window. It returns a return delimited list of the objects in the window.

*Inset = [x]

This is a property of text objects. It controls the amount of space between the frame of the object and the text itself. It defaults to 3 for 12 point text and higher and to 2 for smaller sizes. This is similar to the `wideMargins` property of a HyperCard field.

*ItemType = [WIND, PUSH, RAD, CHK, CNTL, LBL, TEXT, ICN#, ICON, PICT, LIST, POP, BOX, LINE, RBOX, MOOV]

This is a read-only property of all objects. It specifies the kind of object it is. Only object number 0 may be of type WIND.

*KeyFilter = [Key*f* resource name or ID]

This is a property of text objects. A KeyFilter controls what characters may be typed into a field. For example, OnlyDigits does not allow any keys except the digit keys to work in a text object which uses this filter. Custom ones may be added. See Appendix C for more information.

KeyScroll = [TRUE, FALSE]

This is a property of lists and QuickTime objects.If it is true, then all keystrokes and edit menu commands will be intercepted by the object when it has the focus—when it has been clicked on or tabbed to. Tab will move the focus to the next object that wants it. Lists will scroll to and select the first item in the list which matches the key typed. Arrow keys may also be used to manipulate the selection.

- **Note:** This is the same as the wantsFocus property.

Leap = [x]

This is a property of a control. It determines by how much the controls value should change as a result of clicking in the "Page-Up" or "Page-Down" region of the control.

LineHeight = [SINGLE, DOUBLE, TRIPLE, x]

This is a property of text objects. It determines the vertical line spacing of the text. It may be set to pre-defined values which vary according to the font size, or may be set to a fixed number.

- **Advanced Users:** When dealing with text objects that contain mixed styles, the rules change slightly. A `lineheight` of Single is interpreted to mean variable line heights as needed to accommodate the various sizes of text which potentially might appear on a line. Double and Triple will not double and triple this line-by-line height but rather set a fixed height for all lines based on the base style designated by the `texfont` and `textstyle` property.

LockText = [TRUE, FALSE]

This is a property of text objects. It determines if the contents of the object may be selected and changed.

Logic = [NONE, ANY, SINGLE, CONTIGUOUS, DRAG]

This is a property of lists. It determines what logic governs selections. NONE prevents any selection from being made. ANY allows any set of list objects to be selected. SINGLE allows only a single item to be selected. CONTIGUOUS allows only a single, contiguous block of list items to be selected. DRAG allows only a single item to be selected but also allow any item to be repositioned in the list by dragging.

- **Advanced Users:** The List Manager keeps a field for each list, selFlags, that governs the selection. You can set this field yourself by putting the value you want for selFlags into the logic property. For more information on the List Manager, see *Inside Macintosh Vol. IV*.

Logic= [NONE, PUSH, RADIO, CHECK]

This is a property of icons and pictures. It determines what logic governs selections within the parts of the object (as divided by the Grid property). NONE prevents any selection from being made. PUSH allows any single part to be selected—it is immediately deselected. RADIO allows any single part to be selected at a time—it remains selected. CHECK allows a set of parts to be selected at a time.

Max = [x]

This is a property of controls. It determines the maximum value that the control may have.

MaxSize = [x]

This is a property of the window as a whole. It defines the maximum size of the window. It is used during zooming and resizing.

*MenuBar = [MBAR resource name]

This is a property of the window as a whole. It specifies a menubar (MBAR) resource which lists the menus for the window. It may be used instead of the Menus property discussed below.

Menus = [m1 & return & m2 ...]

This is a property of the window as a whole. It specifies the names (or ids) of the menus which are to appear in the menubar when the window has "edit"—when keypresses will go to the window.

Min = [x]

This is a property of controls. It determines the minimum value that the control may have.

MinSize = [x]

This is a property of the window as a whole. It defines the minimum size of the window. It is used during zooming and resizing.

Name = [name]

This is a property of all objects. It is the name of the object. In the case of the window, push buttons, radio buttons, checkboxes and controls, the name entered into the property picker appears. In the case of text, lists, boxes and popups, the name defaults to the first line of the text of the object, but may be set to anything. In the case of icons or picture objects, the name is that of the graphic resource being displayed as that object; if set, it attempts to display a picture resource by that name.

Number = [x]

This is a property of all objects. It is a sequential number assigned to each object in turn as it is created. The window itself always has a number of 0. This sequence controls the front-to-back layering of objects and radio button grouping.

*NumberOfItems = [x]

This is a property of the window as a whole. It is the count of objects in the window, not including the window itself.

- **Advanced Users:** This property may be set to a value less than the current one, which has the effect of deleting all objects beyond that point.

Params = [parameter map]

This is a property of windows. It is a return-delimited list of object properties to be set by additional parameters in the initial call to WindowScript. For more information, see the end of this chapter.

Pen = [NONE, WHITE, BLACK, GRAY, LTGRAY, DKGRAY, x]

This is a property of boxes. It specifies the pattern used to draw the outline of the box.

- **Advanced Users:** Other patterns in the system's pattern list may be specified by x, an index value into the system pattern list. See *Inside Macintosh Vol. 1* for more information.

Pensize = [x,y]

This is a property of boxes. It specifies the width (x) and height (y) of the pen used to draw the box.

*Properties = [properties]

This property has two different meanings. When setting it, it is used to set a group of properties all at once. The property value should look like a return-delimited list where each line of that list is of the form:

```
ObjectName_Property: PropertyValue
```

When getting it, it returns the names of all the properties of that object. (See the Notes chapter for more details.)

- **Advanced Users:** The supported properties of each object are stored in a TEXT resource in WindowScript.

Rect = [l,t,r,b]

This is a property of all objects. It is the rectangle in which a particular object is displayed. The coordinates are usually given relative to the TopLeft of the window. The window's rect, however, is given relative to the screen as a whole.

Repeating = [TRUE, FALSE]

This is a property of QuickTime movie objects. If it is true, the movie will begin again from the first frame when it has reached the end.

Result = [result map]

This is a property of windows. It is a return-delimited list of object properties to be returned by the WindowScript function. For non-modal dialogs, the `value` of the window is controlled by this property. For more information, see the end of this chapter.

Script = [text]

This is a property of all objects. It is a set of HyperTalk statements which will be executed when that object is "hit." For windows, this means window events such as `activate`, `deactivate`, and `open`. For objects scripts, this means when a click appropriate for the object is detected. For instance, buttons are hit on mouse-up; edit text objects are hit on close field. An object script takes precedence over the window's hit handler. See the Scripting chapter.

*Scroll = [vertical] | [vertical,horizonal]

For text objects, lists and pictures, scroll indicates the value of the scrollbar's thumb, when present, in pixels. For QuickTime objects, it represents the value of the scrollbar's thumb in whatever units the movie was recorded in.

- **Note:** For a movie, the `scroll` divided by the `timeValue` equals seconds elapsed.

Scrolling = [TRUE, FALSE]

This is a property of text objects, lists, QuickTime objects and pictures. It determines the presence of a scrollbar attached to the object. Whereas lists and text objects get only a vertical scrollbar, pictures get both horizonal and vertical scrollbars. QuickTime objects get their special QuickTime controller.

*Selection = [i]

This is a property of the window. It indicates the object which is currently selected. Only objects which can receive keystrokes are considered to be selected. LIST objects (whose `keyScroll` is TRUE) and TEXT objects (whose `lockText` is FALSE) can accept keystrokes. Windows that have menus, or that have a `cancelItem` or a `defaultItem`, can also receive keystrokes.

- **Advanced Users:** Setting the selection to 0 has a special meaning—it causes the window to select the first appropriate object in the window. This comes in handy to ensure the edit goes to a particular window.

*Selection = [s1,...,sx]

This is a property of lists, popups, QuickTime objects and grids. It is a comma-delimited list of the selected items in the object. In the case of lists, the selection may also be specified using ranges such as "1-5" or even groups of ranges such as "1-5,10-35." For QuickTime objects, it is the first and last unit of the selection, in whatever units the movie uses (probably frames).

*SelectionColor = [RGB color]

This is a property of text objects. It controls the color of the currently selected text in the object. For text objects that don't use mixed styles, use the `blackColor` instead.

*SelectionFont = [font name]

This is a property of text objects. It controls the font of the currently selected text in the object. For text objects which do not have mixed styles, use the `TextFont` property instead.

*SelectionRect = [l,t,r,b]

This is a read-only property of lists. It returns the coordinates of the first object selected in a list. It will scroll the list if necessary to make the selection visible. This is very handy for making edit-in-the-list windows—this rect may be used to place an edit object over the selection.

*SelectionSize = [font size]

This is a property of text objects. It controls the size (in points) of the currently selected text in the object. For text objects which do not have mixed styles, use the `TextSize` property instead.

*SelectionStyle = [PLAIN, BOLD, ITALIC, UNDERLINE, SHADOW, OUTLINE, CONDENSE, EXTEND, GROUP]

This is a property of text objects. It controls the style of the currently selected text in the object. For text objects which do not have mixed styles, use the `TextStyle` property instead.

*SelectionStyle = [HILITE, INVERT, FRAME, LASSO]

This is a property of pictures and icons whose `logic` is not none. When a picture or icon is selected, this property determines how that selection is indicated. Hilite uses the hilite color (on color machines). Invert simply inverts the colors. Frame draws a black border. Lasso shrinks the selection color so it covers just the graphic. It shrinks the hilite to cover the area inside the `blackColor` of the object. (Lasso should be used only with push logic.)

*SelectionText = [t1 & return & ... & return & tx]

This is a property of lists. It is a return-delimited list of the text of selected objects in the list. If this property is set, the selection of the list is changed rather than the text of the selected objects. See `TextOfSelection`.

*SelectionTime = [time start,time stop]

This is a property of QuickTime objects. It returns the selection of the movie in seconds.

*SoundVolume= [volume]

This is a property of QuickTime movie objects. It is the volume at which the movie's sound will be played. You specify the volume as a percentage; for instance, "100" would be full volume.

*Speed= [speed]

This is the speed to play the movie at, represented as a percentage of full speed. "50" would be half speed, and "200" would be double speed.

Step = [x]

This is a property of a control. It determines by how much the controls value should change as a result of clicking in the "Line-Up" or "Line-Down" region of the control.

Style= [DIALOG, WINDOW, WINDOID]

This is a property of a window. It determines how the window appears and in which layer it is placed.

- **Advanced Users:** You may specify any WDEF as the style. If you do, you must also specify one of the above in the Logic property of the window.

Style= [CDEF Name]

This is a property of a control. It is the name of the resource which defines the custom behavior of the object.

- **Advanced Users:** You can specify the name or ID of any CDEF that exists in the stack.

Style= [LDEF Name]

This is a property of a list. It is the name of the resource which defines the custom behavior of the object.

- **Advanced Users:** You can specify the name or ID of any LDEF that exists in the stack.

Style= [MDEF Name]

This is a property of a popup object. It is the name of the resource which defines the custom behavior of the object.

- **Advanced Users:** You can specify the name or ID of any MDEF that exists in the stack.

StyleRun = [Selection; selectionFont; SelectionSize; SelectionStyle; SelectionColor]

This property is not used while the window is open. It is only used when the window is saved, to indicate a style for a set of characters in a text object. You may see it if you open a window as text.

Text = [text]

This is a property of popups, lists, text objects and boxes. It is the actual contents of those objects.

TextAlign = [LEFT, CENTER, RIGHT]

This is a property of text objects. It determines the alignment of the text within its rect.

TextFont = [font name]

This is a property of all objects. It determines the font in which the text of the object will be drawn. If you change the `TextFont` of the window, each object within the window which had the same font will be changed also.

*TextOfSelection = [t1 & return & ... & return & tx]

This is a property of lists. It is a return-delimited list of the text of selected objects in the list. If this property is set, the text of the selected objects is changed, rather than the selection of the list. To do the inverse, use the `selectionText` property.

TextSize = [x]

This is a property of all objects. It determines the size in which the text of the object will be drawn. If you change the `TextSize` of the window, each object within the window which had had the same `TextSize` will be changed also. If you want to change the size of a portion of a text field, it must first be selected and then its `selectionSize` property be set.

TextStyle = {PLAIN, BOLD, ITALIC, UNDERLINE, SHADOW, OUTLINE, CONDENSE, EXTEND}

This is a property of text objects. It determines what style effects are used in drawing the text of the object. It is a comma-delimited list of all the applicable styles. If you want to change the font of a portion of a text field, it must first be selected and then its `selectionStyle` property be set.

TimeValue = [time]

This is a property of QuickTime objects. It is the number of units (probably frames) per seconds for the movie.

- **Note:** `scroll / timeValue`= seconds elapsed

TitleItem = [x]

This is a property of popups. It specifies another object (usually a label) which is to act as the title for the popup. This title will be hilited when the popup is "popped."

TopLeft = [x,y]

This is a property of all objects. It specifies the location of the top-left corner of the object's enclosing rectangle, relative to the window. In the case of the window, it is relative to the screen.

*Value = [SelectionText] | [Hilite] | [Text]

This is a property of all objects. It represents the most "interesting" property of the object. In the case of push buttons, radio buttons and checkboxes, it is the `hilite`. In the case of controls it is the value of the control. In the case of popups and single selection lists, it is the `SelectionText`. In the case of pictures and lists capable of multiple selection, it is the `Selection`. Finally, in the case of a box, rounded box or line, it is empty.

*Version= [version]

This is a read-only property of the window. It returns the version of WindowScript being used.

Visible = [TRUE, FALSE]

This is a property of all objects. It determines if the object is displayed or not.

WantsFocus = [TRUE, FALSE]

This is a property of lists and QuickTime objects. If it is true, then all keystrokes and edit menu commands will be intercepted by the object when it has the focus—when it has been clicked on or tabbed to. Tab will move the focus to the next object that wants it. Lists will scroll to and select the first item in the list which matches the key typed. Arrow keys may also be used to manipulate the selection.

- **Note:** This property is the same as the `keyScroll` property.

WhiteColor = [RED, GREEN, BLUE] | [x]

This is a property of all objects. On Macs capable of displaying color, it defines the color that should be used in place of white when drawing the object. The color is specified as a RGB value. Be careful when choosing colors for windows that may be used on black and white or less colorful monitors. The colors will map to the nearest color available, or to white if black and white.

- **Advanced users:** This property may also be set with an index into the system CLUT id 256. The external function `AnswerColor` may be used to prompt a user for a color with the standard color picker.

Width = [x]
This is a property of all objects. It specifies the width of the object.

*Wordwrap = [TRUE, FALSE]
This is a property of text objects. It specifies whether or not the text should wrap when it reaches the right-hand margin of the object.

Commands

AdjustSize
This command tells the object to revert to its natural size. It has the same effect as having the object's autosize property set to true, but it does not affect the autosize property.

BringToFront
This puts the window in front of all the others in HyperCard's application layer.

Clear
This command may be sent to text and QuickTime objects. It is similar to its edit menu equivalent.

Copy
This command may be sent to text objects, or lists or QuickTime objects whose `wantsFocus` property is true. It is similar to its edit menu equivalent. For lists, it places a copy of the selected line onto the clipboard. For movies, it puts a copy of the selected frames onto the clipboard.

Cut
This command may be sent to text or QuickTime objects. It is similar to its edit menu equivalent. This command, unlike copy, does not work with lists.

DoUpdate
Redraws the object. This is used internally; it's rare that you would need it.

First
For QuickTime objects. This moves to the first frame of the movie.

GetBGFieldText
This command may be sent to a text object. It will cause a copy of the text found in the background field (whose name is the same as the text object's) to be copied into the object. All text styling will also be copied. This only works for fields on the current background.

GetCdFieldText
This command may be sent to a text object. It will cause a copy of the text found in the card field (whose name is the same as the text object's) to be copied into the object. All text styling will also be copied. This only works for fields on the current card.

Last
For QuickTime objects. This moves to the last frame of the movie.

Paste
This command may be sent to text or QuickTime objects. It is similar to its edit menu equivalent.

Pause
For QuickTime objects. This pauses the playing of a movie.

Play
For QuickTime objects. This plays the movie at the current speed.

Prev
For QuickTime objects. This moves to the previous frame of the movie.

Next
For QuickTime objects. This moves to the next frame of the movie.

SendToBack
For windows. This puts the window behind all the other windows in HyperCard's application layer.

SnapShot
For windows. This command places a picture of the window as it appears onto the clipboard. The picture will be in PICT format.

Stop

For QuickTime objects. This stops the playing of the movie.

ZoomIn

This command causes the window to zoom in to its minimum size. If the window has a grow box, then it will zoom in to the last size chosen by the user. If the window has no grow box, it will zoom to the minimum size as defined by the `MinSize` property.

ZoomOut

This command causes the window to zoom out to its maximum size as defined by the `MaxSize` property.

Chapter 8
NOTES ON PROPERTIES

Custom Properties

Often you'll want to associate information with a particular window, as when you want a window to be entirely self-contained and not dependant on any card or stack. WindowScript allows you to do this with custom properties. Basically, you can make up any property you want and set and get its value. The only restriction is that these property names must begin with a lowercase "x". That is how WindowScript knows to store the value for later retrieval.

WindowScript does not "understand" these properties; it is merely storing them for you.

For instance, you can create the property "user" with:

```
set xUser of window "myWindow" to "Bill"
```

and to retrieve it:

```
get xUser of window "myWindow"
```

These property values are not saved; when you close the window, the information is lost. They may, however, be supplied as parameters in the initial call to WindowScript using the `Params` property. You can also retrieve them when your window is closed and store them in a hidden field for use the next time the window is opened.

Notes on Style

Lists, controls, popups and windows have a `Style` property which allows you to specify the exact appearance and operation of the object. This is done through a self-contained program called a definition procedure. (These DEFs are defined by *Inside Macintosh Vol. 1.*) Some have been included for your use. Additional ones may be made in a conventional development environment, such as Pascal.

You add your own CDEFs, LDEFs, MDEFs and WDEFs by simply copying them into the resource fork of your stack and specifying their name with the style property.

A short description of those included in the package is provided below.

ListOfIcons (LDEF)

This LDEF allows you to display a two-dimensional list of icons in a list object. Each cell in the list is 40x40 pixels. The appropriate number of columns is automatically calculated. The TEXT of the list should contain the resource ids of the icons to be displayed. One handy way to obtain the ids of all available Icons is to use the ResourceList XFCN. Color icons are used on color machines when found.

ListOfFinderIcons (LDEF)

This LDEF allows you to display a two-dimensional list of finder icons in a list object. Each cell in the list is 40x40 pixels. The appropriate number of columns is automatically calculated. The TEXT of the list should contain the resource ids of the finder icons to be displayed. One handy way to obtain the ids of all available Finder Icons (ICN#) is to use the ResourceList XFCN.

ListOfPictures (LDEF)

This LDEF allows you to display a two-dimensional list of pictures in a list object. Each cell in the list tries to be 100 x 100 pixels. The appropriate number of columns is automatically calculated. The TEXT of the list should contain the resource ids of the pictures to be displayed. One handy way to obtain the ids of all available pictures is to use the ResourceList XFCN.

ListWithTabs (LDEF)

This LDEF allows you to display a list where each line is formated into columns. These columns are not true list cells, like a spreadsheet, since only the whole line may be selected. The TEXT of the list should contain the text to be placed in each line. To denote a tab in a line, you may use either the tab character or the "◊" character (option-shift-v).

The placement of the tab stops and their alignment (left, center, right) is controlled by an additional property: StyleInfo. For this LDEF, this contains a string of characters where each represents a horizontal measure of 8 pixels. The character may be a "." which acts as a placeholder, a ">" which denotes a left aligned tab, a "^" which denotes a center aligned tab, or a "<" which denotes a right aligned tab. For example, a Styleinfo property of ".....>....^.....<" would define three tab stops:

1. A left align tab stop at 40 pixels
2. A center align tab stop at 80 pixels
3. A right align tab stop at 160 pixels

ListWithIcons (LDEF)

This LDEF allows you to display a list where each cell contains an icon on the left and a text block area on the right. The cells will be adjusted in height to accommodate at least three lines of text. The TEXT of the list should contain the resource ids of the icons to be displayed, followed by a space and then the text to appear to the right of the icon. One handy way to obtain the ids of all available Icons is to use the ResourceList XFCN. Color icons are used on color machines when found.

ListAsMenu (LDEF)

This LDEF displays a list of items as if the list were a menu. Meta-characters (such as "/" and "(") have the same effect as in menus. This LDEF is handy if you need to create a menu-making window.

Digit (CDEF)

This control mimics the standard behavior of the control used in the Alarm Clock DA. It is usually used for changing the individual fields of dates and times.

Latch (CDEF)

This control looks like a little latch which may be either up or down. Most often this is used to control an optional portion of a window—which is either hidden from view or shown according to the state of the latch.

PushLatch (CDEF)

This strange control should be familiar to users of the property picker. It combines the abilities of a simple latch with a push button.

ColorPicker (CDEF)

This control displays the colors in a CLUT which is specified by the title of the control.

Clut (MDEF)

This menu definition displays the colors in a clut which is specified by the text of the first object in the menu. If you want the MDEF to display a previous selection, place its offset into the clut, or the actual RGB value after the clut name, separated by a comma. For example: if the text of the popup was "Rainbox,23" then the colors in the Rainbow clut would be displayed with the 23rd item indicated as the previous choice.

Tear (MDEF)

This menu definition allows you to specify one or more pictures to be used as the menu. Each picture may be split up into a grid similar to a picture object in WindowScript. To specify the pictures to be used, the text of the menu should be the names of the pictures followed by the desired grid. For example, if the text of a popup was "Example,3,4" then the menu would display the picture named "Example" and split it into a 3x4 grid.

A final piece of information may be appended onto a line of menu text: the initial selection. In the above example, if we wanted the second piece of the grid to be indicated as the prior selection, we would set the text of the object to "Example,3,4,2". Lastly, if we want the menu to allow itself to be torn off, the final item in the menu should be a rectangle, any rectangle. When the menu is torn off, WindowScript will return an itemValue of 512 and this line of the menu's TEXT will contain the actual global coordinates of the destination of the menu. It is then up to you to place a separately defined, but similar looking, palette at that location.

Notes on the Menus Property

A handy feature of WindowScript is its ability to associate a set of menus with a window. Whenever the window has the "edit," that is, when it is to receive keystrokes, WindowScript appends the menus in the `menus` property to the current menubar.

The `menus` property keeps a return-delimited list of the menus to be used. You can specify the menus by resource name or ID. If the first menu used is an Apple menu, the menu bar is completely replaced. Otherwise, the window's menus are appended to the current menu bar. These menus must exist as a resource in the stack. You create them using ResEdit or another menu resource creation utility.

Apple menu items (such as DAs) and font menu items (a list of available fonts) will be appended automatically. Also, standard Edit menu commands Cut, Copy, Paste will work automatically if their keyboard equivalents have been set to the standard X, C, and V.

Unfortunately, HyperCard's menu management commands do not work on these menus. So, two externals have been included which allow you to do basic menu maintenance: menuSet and menuGet. MenuSet is used as follows:

```
menuSet menu, menuItem, property, value
```

MenuGet has a similar syntax:

```
menuGet (menu, menuItem, property)
```

For more information, see the chapter on XCMDs and XFCNs.

Notes on the Params Property

This window property provides a mechanism for setting the properties of objects when a window is first opened. You can map values passed to the WindowScript XFCN directly to properties of an object. This map is stored in the `Params` property of a window.

For example, take a simple list dialog which had the list as object 3. If you set the `Params` property to:

```
i3_Text
i3_Selection
```

then you could invoke the dialog like this:

```
get WindowScript("ListDialog",myList,"1-4")
```

which would set the text of the list to the contents of the container myList and would select lines 1 to 4 of that list.

Parameter 1 is always the name of the window or the textual description of the window. So, the Params property "begins" with parameter two.

Notes on the Result Property

This property allows you to return whatever properties are of interest from any object in the window. These properties are collected for you whenever the `Value` of the window is requested or when a modal dialog is dismissed. (This later case is where the property gets its name since the value of the window is returned by the WindowScript XFCN.)

For example, take a simple list dialog which had the list as object 3 (object 1 is an OK button and object 2 is a Cancel button). If you set the Result property to:

```
i3_Selection
```

then when you invoked the dialog like this:

```
put WindowScript("ListDialog",myList,"1-4") into
myContainer
```

the value of myContainer would be two lines. The first line would contain the name of the object which closed the window (i.e., OK). The second line would contain the Selection property of the list (e.g., 1,3,8).

If a button is designated as the CancelItem and is used to dismiss the window, empty will be returned rather than the properties you requested.

Line 1 is always the name of the object which closes the window. So, the Result property "begins" with line two.

Some properties can span more than one line. If such a property is specified in Result, it could corrupt the line numbering of the subsequent property values. Therefore, unless the property is the last in the list, its value will be "scrunched" by changing its return characters into "¬" characters. You can use the `Unscrunch` XFCN to undo this. See Chapter 10 for details.

Notes on the Properties Property

Often, you'll have a batch of changes you want to make to a window. If you do a set for each of them, the window updates will be slow and ugly. An easy way to update a batch of properties is with the `Properties` property. You can collect a list of changes in the form of:

```
ObjectName_Property:Value [RETURN]
```

and then set the `Properties` property of the window to this container. All of the properties contained therein will be changed.

For instance, if you wanted to make the following changes:

- Hilight radio button 1
- Unhilight radio buttons 2 and 3
- Unlock the edit text field

then you would put these changes into a container, such as `it`:

```
get "Radio1_Hilite:TRUE"¬
& return & "Radio2_Hilite:FALSE" ¬
& return & "Radio3_Hilite:FALSE" ¬
& return & "someText_LockText:FALSE"
```

Then you set all of these at once:

```
set the properties of window "My window" to it
```

which will set the properties of the window described in `it`. There is no window updating between changes, so it happens quickly and invisibly. This is the best way to change a set of properties quickly.

You can also use the `properties` property to create a new object while the window is open. You'd probably only want to do this in the most esoteric of circumstances. To do it, you set a string container as before, but instead of modifying properties of existing objects, you simply make a new one. This example will create a new checkbox:

```
get "_itemType:CHK" &¬
return & "_rect:20,20,70,25"
set the properties of window "My window" to it
```

and the new checkbox will appear.

When you create a new object on the fly, you must define at least the `itemType` and the `rect` for the item.

Please note that any new objects you create will not be saved when the window is closed. They exist only for this instance of the window. There is no way to save these newly created objects without the WindowScript stack. If you need new objects to appear in a window, you can also create them beforehand and set their `visible` property to false until you need them.

Invisible Buttons

If you want to be able to respond to a click on an area of a window without obscuring other objects in the window, you can use an invisible button. For instance, if you have several items that are related, and clicking any one should invoke a single script, you can place an invisible button over them.

You create an invisible button by creating a picture object and setting its `name` to anything that won't display a picture—that is, any name that isn't the name of a PICT resource. This way, WindowScript will look for the picture to display, find none, and display an empty object.

Make sure you set the `logic` property of the picture, or there will be no selection. The chosen `selectionStyle` will still work, selecting whatever objects show through from underneath the picture. Whatever logic you have chosen will work; if you choose lasso, the selection will shrink to fit objects underneath the picture. You can still use the `grid` property to divide the picture into sections.

The Lasso `selectionStyle` matches to the `blackColor` set for that picture. That is, it begins selecting the entire picture, then shrinks the selection around the edges until it hits the blackColor. In this way, the white space of a picture is left white.

Notes on Group Style

The group style, like HyperCard's group style, can be used to group together a string of characters. When there are characters styled as a group in a locked text object, a hit in that object over any of the characters will select the entire group; the hit will return all of the characters in that group in the objValue variable. This makes creating HyperText windows a snap.

For an unlocked text object, holding down the command key will temporarily lock the field.

Chapter 9
SCRIPTING

WindowScript gains much of its power through its scripting capabilities. You can write scripts to be executed when an object is "hit", that is, clicked on. These scripts exist either internal to the object or within a handler in the stack. Both are useful for different situations.

WindowScript, unlike HyperCard, doesn't allow you to write scripts for each event associated with an object. WindowScript sends a hit only on the most interesting event for that object.

What are hits? Usually, hits correspond to either a `mouseUp` (when the object has something to do on mouseDown, like a popup) or a `mouseDown` (when it doesn't, like a box). In the case of editable text objects, WindowScript sends a hit on `closeField`. Also, a variety of special hit messages are sent to the window as whole, reporting such actions as the window being opened, zoomed, closed, etc.

Window events:

Open	CloseBox
Close	ZoomBox
Suspend	GrowBox
Resume	TitleBar

Given a pair of WindowScript windows, when you switch between them, you will receive an activate event first, then the deactivate. This is the reverse of what you would normally expect. This is an "anomaly" of HyperCard.

Object Scripts

Object scripts are contained in an object's script property. When an object has a script, the script is executed upon receiving a hit. If the window has a hit handler—that is, a handler in the stack that is executed for every hit—the hit handler is ignored if the object in question has its own script.

Unlike HyperCard, you can't create more than one handler in a script. An object script is just that—the lines of HyperTalk to be executed when the object is hit. You can't create functions or handlers that are local to that object, as you can for a HyperCard button or field.

Hit Handlers

Sometimes it's easier to create one large script to deal with everything for a window, rather than writing a script for each object. In this case, you can create a single hit handler to deal with all of a window's events. You specify the handler with the `HitMessage` property of a window. The `HitMessage` is sent whenever an object is hit or a window event is registered. (If the hit object has an object script, the `HitMessage` is suppressed.) A hit handler can be in the card, background or stack script.

For example, if the `HitMessage` property is "BillsWindowHit", this would be sent to the card:

```
BillsWindowHit wdID, wdName, objNo, objName,
objValue
```

Variables

WindowScript provides local variables to object scripts to let them know what has happened when there is a hit. These variables contain:

wdID	the ID of the window
wdName	the name of the window
objNo	the number of the object that was hit
objName	the name of the object that was hit
objValue	the value of the object hit

These same variables are passed as parameters to Hit handlers.

Hit Handler vs. Object Scripts

Which should you use, a hit handler in the stack or scripts in your objects? It all depends. There are benefits to both methods. Benefits of hit handlers:

- Hit handlers can use HyperCard's debugger
- One hit handler can be used by many windows
- HyperCard will precompile the script into memory for you, so it's faster

Object scripts, on the other hand:

- Will travel with windows, so windows can be self-contained entities
- Don't have a huge *if...then...else* statement to deal with
- Are easier to enter and modify, being in the property picker

In general, it's easier to write and debug everything in a hit handler first, then move the appropriate parts of the handler into the objects.

Checking the Result

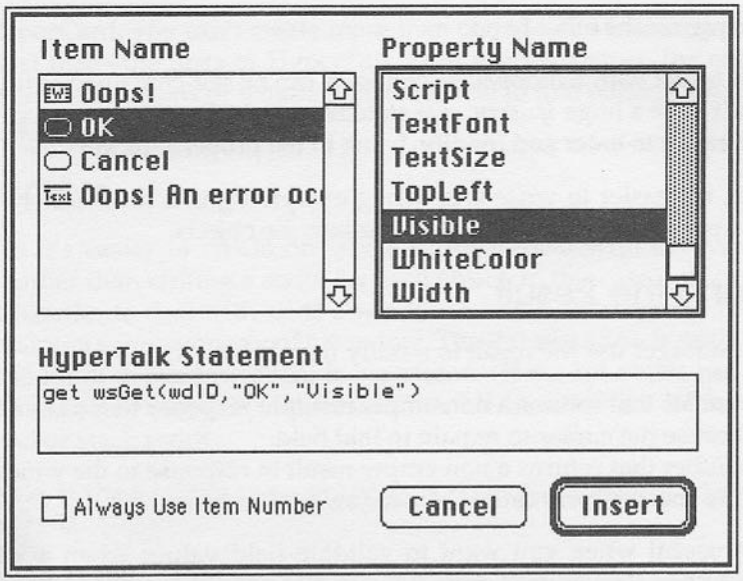
Some hit messages use the result to modify their behavior:

- A handler that returns a non-empty result in response to a `closefield` will cause the cursor to remain in that field.
- A handler that returns a non-empty result in response to the window's `close` message will cause the window to stay open.

These are useful when you want to validate field values when a window is closed, and keep the window open if the values are not valid. (You can return a value from any handler, not just from function handlers.)

The Scripting Assistant

When you are using a separate window for editing scripts—when you've clicked on the "script" button in the property picker—a menu appears: the Assistant menu. This menu can help you write a script for that object. You can use the scripting assistant to get a property, set a property, send a command, or open another WindowScript window. The assistant window looks like:



You can select the object and property you want to change, and when you click Insert, the HyperTalk fragment in the text box will be placed into the script where the insertion point was before the dialog was opened.

The checkbox "Always use item number" is helpful because, if you write a script now and change names of objects later, your script will use the old names. Using item numbers is a handy way around this. (Of course, item numbers can change, too, if you reorder the objects in a window.) Also, using item numbers is faster.

When closeField Is Sent

CloseField messages are sent to a "dirty" field when one of the following happens:

- The user clicks on any other object (including the closebox but not the cancel object)
- The user tabs out of the field (even if there is no other field)
- The user hits Enter or Return
- The window loses edit (in the case of windoids)

They are also sent when some properties are set:

- When the `Dirty` property is changed
- When the `Text` property of a dirty object is set

Speeding It Up

Use HyperCard's Get /Set syntax when possible. It tends to be quicker than `wsGet/wsSet`. Also, it is faster to accumulate a number of property sets into a local container and set them all at once using the `properties` property, than to do them all individually.

Chapter 10
XCMDS AND XFCNS

WindowScript works its magic through external commands and functions. These are usually stored in the WindowScript stack itself; when you install WindowScript, you are really just adding a "`start using`" to your Home stack's `startup` handler.

To use WindowScript in stacks you plan to distribute, you'll need to copy the appropriate XCMDs and XFCNs to that stack. WindowScript's installer card can do this for you. See the chapter "Distributing WindowScript."

Listed below are all externals that come with WindowScript. Some are essential to its operation; others are just helpful. You can use all of these in your stacks. There are other externals that WindowScript uses "behind the scenes." These are not listed here, as they are for internal WindowScript use only, and are not supported as stand-alone externals.

answerColor(prompt, inColor)

This external function returns the three-part number which represents the color selected in the color picker (also known as an RGB value) dialog. If cancel is clicked, empty is returned.

`prompt`	The text which is to appear in the color picker dialogas a prompt for the user. If omitted, no prompt is given.
`inColor`	The initial color which is being altered

menuGet(menu, menuItem, property)

This command gets the property of a menu item.

`menu`	The name or id of a menu
`menuItem`	The name or number of a menu item
`property`	One of the properties described below

The properties and their potential values are:

`Name`	The text of the menu item
`CheckMark`	True or false
`CmdChar`	Any single character
`MarkChar`	Any single character
`TextStyle`	The style of the object's text
`Disabled`	True or False
`Style`	The MDEF used by the menu
`Text`	Text of the whole menu (with menuitem = 0)

hasQT()

This external function returns either TRUE or FALSE, indicating whether or not QuickTime is present on this machine.

resourceList (file name, type, format)

This external function returns a return delimited list of resources from the specified file.

`file name` The full pathname of the file whose resources are to be listed. If omitted, all open resource forks are scanned. If empty, only the first resource file in the resource chain (usually the current stack) is scanned.

`type` The type of the resources to be listed. If omitted, all resources are listed.

`format` Determines what information is returned. It is either a selector for a built-in format or a prototype line for the result. Recognized selectors are:

ID Each line contains the id of the resource.

Name Each line contains the name of the resource. Only named resources are listed.

Either Each line contains the name of the resource if it has one otherwise it contains the number preceeded by a # sign.

Both Each line contains the id number left justified in a seven characters space followed by the name of there is one.

A prototype line may be supplied instead. Each line will then contain that prototype with resource information filled in where the following words appear (they must be in all uppercase).

ID Will be replaced by the resource's id.

NAME Will be replaced by the resource's name.

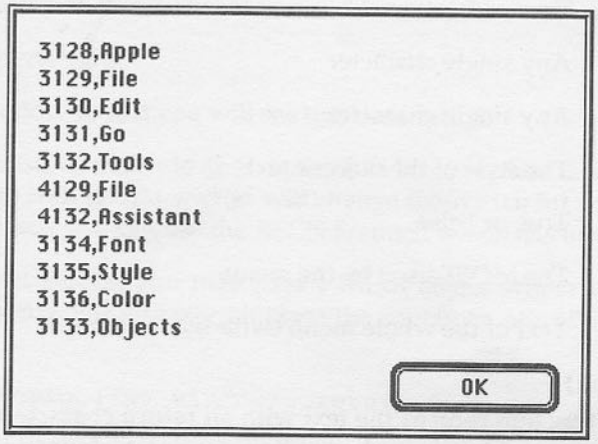
TYPE Will be replaced by the resource's type.

SIZE Will be replaced by the resource's size.

If you executed the following statement in the WindowScript stack:

```
answer resourcelist("", "MENU", "ID,NAME")
```

you would see:



screenInfo (MAIN | DEEPEST | rect | point)

This command returns information about the display screens being used. If you pass main, it returns information for the main screen. If you pass deepest, it returns information for the deepest. If you pass a rect, it returns the info for the deepest display that intersects that rect. If you pass a point, it returns the screen

that contains that point. If you pass nothing, it returns a return-delimited list of all screens currently attached.

The information it returns is in the form:

```
colors l,t,r,b
```

where first is the number of colors available on that screen, followed by the rectangle of that screen.

menuSet menu, menuItem, property, value

This command sets the property of a menu item.

`menu`	is the name or id of a menu
`menuItem`	is the name or number of a menu item
`property`	is one of the properties described below
`value`	is the value of the property

The properties and their potential values are:

`Name`	The text of the menu item
`CheckMark`	True or false
`CmdChar`	Any single character
`MarkChar`	Any single character
`TextStyle`	The style of the object's text
`Disabled`	True or False
`Style`	The MDEF used by the menu
`Text`	Text of the whole menu (with menuitem = 0)

scrunch (text)

This external function returns the text with all return characters replaced by "¬" (option-L) characters, thereby reducing it to one line as defined by HyperCard.

`text`	The text which is to scrunched.

This is useful when initializing the text property of text objects in the initial call to windowScript:

```
windowScript("myWindow" & return & "bob_text:" &
scrunch(bg fld 3))
```

unscrunch (text)

This external function returns the text with all "¬" characters replaced by return characters, thereby expanding it from one line as defined by HyperCard. This is most often used to undo the effect of scrunch.

`text` The text which is to unscrunched.

windowScript (window name | window description)

WindowScript is an XFCN that accepts a single parameter. This parameter may be either a texual description of a window to be displayed, or the name of a resource of type LENS which contains such a description. The XFCN returns the ID of the window followed by the window name in quotes (except in the case of modal dialogs, in which case WindowScript does not return until the user has dismissed the window; WindowScript then returns the value of the dialog).

The description used by WindowScript is simply a textual listing of the objects which compose the window and the properties of those objects. Each property of each object is presented on a different line and is of the form:

Property: PropertyValue

If you open a window as text you will see this description.

If a property consists of multiple lines, it needs to be scrunched (i.e., returns need to be converted to ¬'s). WindowScript will change them back for you. If you ever need to scrunch data, you can use the XFCN Scrunch which has been included.

To initialize object values, you may pass a list of object properties and values after the LENS name. For instance, to place the container `stuff` into a list, you could:

```
get windowScript("my window"& return &
"theText_Text:"& stuff)
```

would put `stuff` into the list object named `theText`.

wsGet (window, object, property)

This function returns the value of a property.

`window`	The name of the window that contains the object
`object`	The object being examined. It is either the name of the object or its item number.
`property`	The property to get

This can be used instead of HyperCard's `get` command, although HyperCard's command is faster. However, wsGet must be used for modal dialogs. See Overview for examples.

wsSet window, object, property, value

This external command sets properties for an object.

`window`	The name of the window that contains the object
`object`	The object being manipulated. It is either the name of the objector its item number.
`property`	The property to set
`value`	The value to set it to

This can be used instead of HyperCard's `set` command, and indeed must be used for modal dialogs. HyperCard's set command is faster, however. See Overview for examples.

wsSend window, object, command

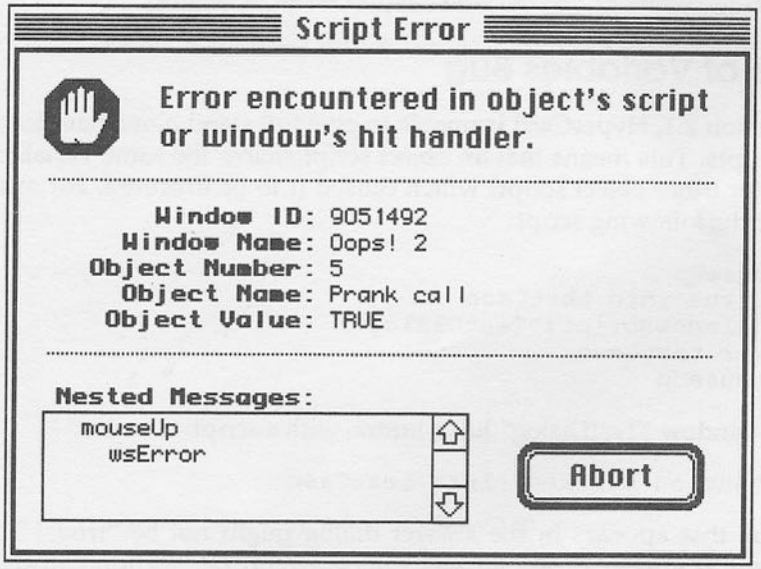
This will send a command to an object. If you specify 0 for object, the command will be sent to the window.

`window`	The name of the window that contains the object
`object`	The object being commanded. It is either the name of the object or its item number.
`command`	The command to send

This can be used instead of HyperCard's `send` command for modal dialogs.

Appendix A
TROUBLESHOOTING

What Happens When an Error Occurs?



When WindowScript detects an error in a script, it informs you of the problem with a dialog box. This dialog box gives certain information about the error condition:

- window name and ID which had the error
- name and ID of the object that caused the error
- what its value was

The list titled "Nested messages" lists the chain of events that led to the error; in this case, a `mouseUp` caused the window to be opened. The handler "`wsError`"

will always be listed, as it is the message sent by WindowScript to show the error dialog box.

The Message Watcher is very helpful in debugging hit handlers. Whereas tracing often causes window events (suspend and resume) and therefore interferes with the process, MW does not.

When WindowScript detects an error, it sends the message "`wsError`" up the hierarchy. The WindowScript stack responds to this message with the above dialog. If you have taken the WindowScript stack out of the hierarchy—e.g., you've done "`stop using`" or something similar—you'll need to handler the error message yourself. See the next chapter for more details.

Scope of Variables Bug

As of version 2.1, HyperCard is unable to give full stand-alone handler status to object scripts. This means that an object script shares the same variables as the handler (or other object script) which caused it to be executed. For example, if you have the following script:

```
on mouseUp
 put true into testCase
 get windowScript("TestDialog")
 answer testCase
end mouseUp
```

And the window "TestDialog" has a button with a script:

```
put "Button Clicked" into testCase
```

The value that appears in the answer dialog might not be "true"! The value which would appear in the answer dialog would be "Button Clicked" if the button had been clicked.

What's happening is that, when WindowScript sends its scripts to HyperCard to be executed, HyperCard doesn't create a clean slate of variable space for the script; it just runs in the variable space already there—and that comes from the last handler executed.

The bottom line:

- Variables in object scripts of dialogs are the same as those in the script which invoked the dialog.

- Variables in object scripts of windows and windoids are usually globals. It is possible to send messages to windows (either directly, like close window untitled, or indirectly like quit HyperCard) which cause item scripts to be executed (the window's script in the two examples). In this case, the variables in the script are not globals, but rather are the same as those in the script which caused the message to be sent.

This is a bug, not a feature. Don't count on this behavior in your windows; we expect it to disappear in a future version of HyperCard.

Appendix B
QUICK GLOSSARY

Activate Event: This event is sent to a window when it is activated. For instance, if a window behind all the others is clicked on, it comes to the front and is sent an activate event.

Hit: The most interesting event for an object. Depending on the object, it can be either a `closeField`, `mouseDown`, or `mouseUp`.

Hit Handler: A HyperTalk script, residing in the card, background, or stack script, that receives all hit messages for a window and responds to them accordingly. It will not be called if the object hit has its own script.

Modal Dialog: A modal dialog enters a "mode," so that nothing else can be done while the dialog is open. Menus are generally unavailable, and clicking outside the dialog window generates a beep. (HyperCard's answer dialog is an example of a modal dialog.)

Modeless Dialog: A modeless dialog box is like a modal dialog box, except that it doesn't enter a mode; it peacefully coexists with other windows. The menus are available and other windows can be clicked on and switched to. (Most word processors' Find dialogs are examples of modeless dialogs.)

Property: An attribute of an object that controls its appearance or behavior.

Resume Event: This event is sent to a window when it is about to become the frontmost (active) window.

RGB: Stands for Red, Green, Blue. This is a method for choosing a color by specifying the amount of individual red, green, and blue in the color. Values may range from 0 to 65535 for each. (The AnswerColor XFCN always returns a valid RGB value.)

Suspend Event: This event is sent to a window just before another window is moved in front of it.

Update Event: This event is sent to a window whenever it needs an update. For instance, if a dialog box temporarily obscures a portion of the window, when the dialog is closed the window is sent an update event so it can redraw its contents.

XCMD: An external command for HyperCard. By creating external commands, a user can add features to HyperCard that were never intended for it.

XFCN: An external function for HyperCard. By creating external functions, a user can add features to HyperCard (like WindowScript).

Appendix C
KEYFILTERS

Keyfilters are tiny pieces of code that either accept or reject the text that has been entered into a text field. You can use them, for instance, to keep characters out of a numeric field, or to verify that a correct entry has been given. One keyfilter, OnlyDigits, has been provided for your use.

You set the keyfilter property not from the property picker, but from a script somewhere, or as part of the window initialization. Alternately, you can edit the window as text and add the property "Keyfilter:x" to an object yourself.

A short keyfilter, written in Pascal, is shown below.

```
unit OnlyDigitsUnit;

interface

  uses
   hyperXCMD, Utilities;

  function Main (paramptr: XCMDPtr;
                 theText: handle): boolean;
implementation
  {$R-}

  function OnlyDigits (paramptr: XCMDPtr;
                 theText: handle): boolean;
  forward;

  function Main (paramptr: XCMDPtr;
                 theText: handle): boolean;
  begin
    Main : = OnlyDigits(paramptr, theText);
  end;
```

```
  function OnlyDigits (paramptr: XCMDPtr;
                       theText: handle): boolean;
    var
      readPtr, endPtr: ptr;
      aStr: str255;
  begin

    OnlyDigits : = TRUE;

    readPtr : = theText^;
    endPtr : = pointer(ord4(theText^) +
GetHandleSize(theText));

  while readPtr <> endPtr do
    if (readPtr^ < ord('0')) | (readPtr^ > ord('9'))
then begin
         OnlyDigits : = FALSE;
         readPtr : = EndPtr;
      end
      else
        readPtr : = pointer(ord4(readPtr) + 1);
  end; {OnlyDigits}
end. {unit}
```

Appendix D
PROPERTIES AND COMMANDS BY OBJECT

Properties

Universal properties

AutoSize
Balloon
BlackColor
Disabled
Height
Rect
Name
Number
Script
TextFont
TextSize
TopLeft
Visible
WhiteColor
Width
Value

Window properties

Dirty
HasCloseBox
HasGrowBox
HasTitleBar
HasZoomBox
HitMessage
IdleTime
Menus
NumberOfItems
Params
Result

Buttons

AutoClose
CancelItem
DefaultItem

Radio buttons

Hilite

Check boxes

Hilite

Labels

Text

Static and Edit Text

LineHeight
LockText
Scrolling
Scroll
Selection
SelectionText
Text
TextAlign
TextOfSelection
TextStyle
Wordwrap

Icons

ID
Grid
Logic

Pictures

ID
Grid
Logic
Scroll
Scrolling
SelectionStyle

QuickTime Objects

File
Repeating
Selection
SelectionTime
SoundVolume
Scrolling
Speed
TimeValue

Lists

DoubleClickItem
KeyScroll
Logic
Scroll
Scrolling
Selection
SelectionRect
SelectionText
Style
Text
TextOfSelection

Popups

Selection
SelectionText
Style
Text
TextOfSelection
TitleItem

Controls

Hilite
Leap
Max
Min
Step
Style
Value

Boxes and RoundRects

Fill
Pen
PenSize

Lines

Pen
PenSize

Commands

Universal commands

AdjustSize
DoUpdate

Window commands

BringToFront
SendToBack
SnapShot
ZoomIn
ZoomOut

Text objects

Cut
Copy
Paste
Clear
GetBGFieldText
GetCdFieldText

QuickTime objects

Cut
Copy
Paste
Clear
Prev
Next
Play
Stop
Pause
First
Last

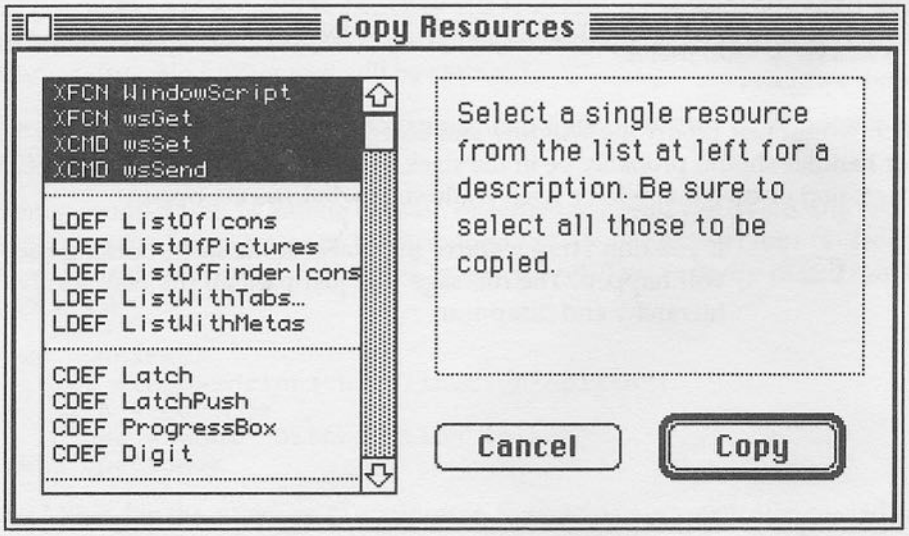
List objects

Copy

Appendix E

DISTRIBUTING WINDOWSCRIPT

You can create a stand-alone stack that uses WindowScript to display your windows. All that is required is that you copy the required resources. Use the installer dialog's "Copy" button to copy the required resources into your stack. This will display the following dialog box:



Simply select the XCMDs, XFCNs and resources you are using. Be certain to copy the first four: WindowScript, wsGet, wsSet, and wsSend. If you're not sure, it won't hurt to copy everything.

If you plan to distribute your stack, you must first arrange licensing. See Appendix F for the license form and terms.

wsError handler

WindowScript sends the message "wsError" when there's an error in a script it attempts to execute. The WindowScript stack has a handler for this message, and that handler displays the standard WindowScript error dialog. (This is described in the Troubleshooting chapter.)

If you want to ship a product that uses WindowScript, or just make a stack that you can distribute, you may have to handle errors in some other way. You can trap the "wsError" call in one of your own handlers, and do whatever you need to do. The WindowScript error dialog will only be displayed if the "wsError" message travels all the way up the hierarchy to WindowScript.

A simple handler could be:

```
on wsError wdName, wdID, objName, objNum, objValue
    answer "WindowScript error:" & wdName & ¬
 return & "objName"
end wsError
```

which would just report the window and object that the error occurred in. The error handler should probably be in the stack script, unless you are certain of the background or card that will be used while your windows are open.

If you don't trap wsError in a stand-alone stack, nothing bad will happen. The message will just pass all the way up the hierarchy and disappear.

Appendix F
LICENSING WINDOWSCRIPT

Fees

If your product is sold through retail channels OR is provided to one or more of your clients for a fee, and such clients are not registered users of WindowScript, OR you otherwise intend to distribute your product for value, then your license fee is One Hundred US Dollars (US$100) payable before you begin distribution of your product. No further fees will be required.

If you are a consultant, your clients must own their own copy of WindowScript, or you must pay the license fee.

If your product will be public domain or shareware, you can avoid any fees by displaying the LENS resource "Distribution Splash" for at least four (4) seconds when your product opens or closes (your choice). This is easily done from the stack script:

```
on openStack
  get windowScript("Distribution Splash")
  wait 4 seconds
  close window "Distribution Splash"
end openStack
```

The LENS for the window "Distribution Splash" is automatically copied into your stack by the installer when you copy the WindowScript XFCN. All you need to do is display it. You may *not* modify this LENS resource in any way . If you do not display this LENS resource, you must pay the fee stated above.

There is a license form at the end of this Appendix, as well as on the product diskette. Fill this form out and mail it back to us with a check. *Whether a fee is required or not, you must complete the form and return it to Heizer Software before you begin distribution.*

Terms:

This license is non-exclusive and non-transferable and grants the Licensee the right to distribute the WindowScript resources in the product named in the Distribution License, provided the following terms are complied with and for the fee detailed above.

Display of copyright and trademark The following must be displayed to users of your product:

> "This product uses certain copyrighted resources from WindowScript™ which are included under license from Heizer Software.
>
> WindowScript ©1990-94 Leonard Buck. All Rights Reserved.
> WindowScript is a trademark of Heizer Software."

The above notice must be displayed in a prominent location in Licensee's product on a title screen, an about box, or other similar location. The LENS resource "Distribution Splash" contains the above text and may be displayed to satisfy this requirement.

The first occurrence of any mention of WindowScript in Licensee's product, literature, or advertising must include a trademark symbol following the word WindowScript (e.g., WindowScript™) and the following line must appear in an appropriate location on the same material:

> WindowScript is a trademark of Heizer Software.

Licensed Resources The following WindowScript resources are covered by this license:

XFCN WindowScript
XFCN wsGet
XFCN AnswerColor
XFCN AnswerQT
XFCN HasQT
XFCN menuGet
XFCN ResourceList
XFCN ScreenInfo
XFCN Scrunch
XFCN Unscrunch

XCMD menuSet
XCMD wsSend
XCMD wsSet

Karl Marx
Karl Napolean
Karl Waldo

WDEF Windoid

Key*f* Only1Line
Key*f* OnlyDigits
Key*f* OnlyMath
Key*f* OnlyNever

LDEF ListAsMenu
LDEF ListOfFinderIcons
LDEF ListOfIcons
LDEF ListOfLabeledArt
LDEF ListOfLargeArt
LDEF ListOfPictures
LDEF ListOfScaledPictures
LDEF ListOfTruncPictures
LDEF ListWithIcons
LDEF ListWithMetas
LDEF ListWithTabs...

CDEF Digit
CDEF Latch
CDEF LatchPush
CDEF Progress7
CDEF ProgressBox
CDEF PushDown
CDEF Working

MDEF Clut
MDEF MenuOfColors
MDEF MenuOfFinderIcons
MDEF MenuOfPatterns
MDEF Tear

These are the only resources which may be distributed under the terms of this license.

Documentation No documentation on the use of the licensed WindowScript resources may be provided.

Title Licensee acknowledges the WindowScript resources are the sole property of Licensor. Licensee is not granted any title whatsoever in the WindowScript resources.

Warranty No warranty of any kind is extended to Licensee's product.

Acceptance Licensor reserves the right to refuse to license the WindowScript resources to any party at its sole discretion. If Licensor rejects a license, any fee received will be promptly returned to Licensee. Upon acceptance, Licensor will send Licensee a signed copy of the license agreement.

Effective Date The effective date of a license will be the date that Licensor signs the agreement.

Termination If licensee breaches any of the terms of this license, Licensor may terminate this license by notifying Licensee in writing. Licensee will have 30 days from the date notice is received to correct the breach or otherwise renegotiate this license, otherwise this license will terminate. In the event of termination, Licensor will NOT return any license fees which may have been paid by Licensee.

Right to Monitor Licensor may at any time and at its sole discretion, request a copy of Licensee's product covered by this license and/or any related manuals, literature, or advertising copy which make mention of WindowScript in order to monitor Licensee's compliance with the terms of this license. If Licensor exercises this right, any materials received will be kept in confidence and will be used solely to monitor compliance with the terms of this agreement. In no event will Licensor redistribute any materials received under this clause nor will Licensor return said materials to licensee.

Future Versions This License shall apply to only one product at a time. If more than one version of a product is distributed at the same time (e.g., a "Lite" version and a "Full" version) then each version will require a separate license. However, if a new version is released and the previous version is discontinued, this license will continue to be valid and will apply to the new version (e.g., v1.0, v1.1, v1.2.5, etc.).

Validity In the event that any portion of this license shall be rendered in a court of law as invalid, unenforceable, or illegal, all remaining portions of this agreement shall remain in effect.

Governing Law This license shall be governed in accordance with the laws of the State of California.

Amendments Licensor may amend this license from time to time as new versions of WindowScript are released. Such amendments will only apply to Licensee if

Licensee chooses to incorporate the upgraded version of WindowScript into Licensee's product.

Nothing in this license shall be construed as the formation of a partnership or joint venture between the parties.

This license supersedes any and all oral or written communications between the parties and is the entire agreement between the parties.

WindowScript Distribution License

If you will be distributing a product that uses WindowScript, you must fill in this license form and agree to the terms stated in Appendix F of the WindowScript manual and return this form to Heizer Software along with a check for the license fee if required. Photocopy this form as needed for additional licenses. No Faxes will be accepted; all license requests must be mailed to Heizer Software, ATTN: Licensing, PO Box 232019, Pleasant Hill, CA 94523. If your product requires a custom version of WindowScript or otherwise has special licensing requirements, call us at 510-943-7667. We'll do our best to meet your needs.

This license is between Heizer Software (Licensor), acting for and on behalf of Leonard Buck and theResult Software, Inc., *and*

Licensee: _____

name _____ company _____

address _____

city _____ state _____ zip _____ daytime phone ( ) _____

Licensee wishes to purchase a license to use the WindowScript resources, as set forth in Appendix F of the WindowScript manual, with the following product:

Product Name: _____

Brief Description (attach a product brochure if you prefer):

I agree with and understand the license terms stated in the WindowScript manual.

name and title _____

signature _____ date _____

☐ A check for $100 is enclosed (No. 30-0514) ☐ My product does not require a fee (No. 30-0515)

Upon acceptance of this License by Heizer Software,
a signed copy of this agreement will be returned for your records.

A license is hereby granted to Licensee by Heizer Software:

name and title _____

signature _____ date _____

Rev 1.5 4/15/94

Appendix G

SCRIPTING QUICK REFERENCE

To set properties use either HyperCard's set command or the wsSet XCMD. The syntax for setting properties is:

```
wsSet <windowRef>,<itemRef>,<propertyName>,<value>
set <propertyRef> of window <windowRef> to <value>
```

Example:
```
wsSet wdID,2,"text",myContainer
set i2_text of window id wdID to myContainer
```

To get properties use either HyperCard's get command or the wsGet XFCN. The syntax for getting properties is:

```
get wsGet(<windowRef>,<itemRef>,<propertyName>)
get <propertyRef> of window <windowRef>
```

Example:
```
get wsGet(wdID,2,"Text")
get i2_Text of window id wdID
```

To send commands to windows and window items use either HyperCard's send command or the wsSend XCMD. The syntax for sending commands is:

```
wsSend <windowRef>,<itemRef>,<command>
send <commandRef> to window <windowRef>
```

Example:
```
wsSend wdID,2,"Copy"
send i2_Copy to window id wdID
```

windowRef:	the name or ID of the window
itemRef:	the name or item number of the desired item
propertyName:	the property of the object that you are interested in
value:	what the property is being set to
propertyRef:	itemRef + _ + PropertyName (e.g. MyField_Text,or if a by number then preceed with i as in i2_Text)
commandRef:	itemRef + _ + Command (e.g. MyField_Copy, or if a by number then preceed with i as in i2_Copy)
command:	the command being sent to the window or window item

For modal dialogs wsSet, wsGet and wsSend must be used. For more information on using these properties and commands, please refer to Chapter 7.

	Pg Ref	Prop. Picker Access	Read/ Write Status	Window	Push Buttons	Radio Buttons	Checkbox	Labels	Editable Text Fields
AutoClose	37	Y	R/W		BOOL				
AutoSize	37	Y	R/W	BOOL	BOOL	BOOL	BOOL	BOOL	BOOL
Balloon	37	Y	R/W	TEXT	TEXT	TEXT	TEXT	TEXT	TEXT
BlackColor	38	Y	R/W	RGB	RGB	RGB	RGB	RGB	RGB
CancelItem	38	Y	R/W		BOOL				
Commands	38	N	RO	TEXT	TEXT	TEXT	TEXT	TEXT	TEXT
Cursor		N	R/W	STR [INT]	STR [INT]	STR [INT]	STR [INT]	STR [INT]	STR [INT]
DefaultItem	38	Y	R/W		BOOL				
Dirty	38	N	R/W						BOOL
Disabled	39	Y	R/W		BOOL	BOOL	BOOL	BOOL	BOOL
DoubleClickItem	39	Y	R/W						
File	39	Y	R/W						
Fill	39	Y	R/W						
Grid	39	Y	R/W						
Group		N	R/W						
GrowItem	39	Y	R/W		BOOL	BOOL	BOOL	BOOL	BOOL
Handle	39	N	RO	INT	INT	INT	INT	INT	INT
HasCloseBox	40	Y	R/W	BOOL					
HasGrowBox	40	Y	R/W	BOOL					
HasTitleBar	40	Y	R/W	BOOL					
HasZoomBox	40	Y	R/W	BOOL					
Height	40	Y	R/W	INT	INT	INT	INT	INT	INT
Hilite	40	Y	R/W			BOOL	BOOL		
HitMessage	40	Y	R/W	TEXT					
ID	40	N	R/W						
IdleDelay	40	N	R/W	INT					
Inset	41	N	R/W						INT
ItemList	41	N	RO	STR					
ItemType	41	N	RO	STR	STR	STR	STR	STR	STR
KeyFilter	41	N	R/W						STR
KeyScroll	41	Y	R/W						
Leap	41	Y	R/W						
LineHeight	41	Y	R/W						STR [INT]
LockText	42	Y	R/W						BOOL
Logic	42	Y	R/W	STR					
Max	42	Y	R/W						
MaxSize	42	Y	R/W	INT					
Menubar	43	N	R/W	INT [STR]					
Menus	43	Y	R/W	INT [STR]					
Min	43	Y	R/W						
MinSize	43	Y	R/W	INT					
Name	43	Y	R/W	STR	STR	STR	STR	STR	STR

BOOL boolean (True or False)
INT integer
RGB can be either an RGB ("integer, integer, integer") or a color index (integer)
STR single line string (max 255 characters)
TEXT string which may contain multiple lines (max 32K characters)

Static Text Fields	ICONs & Finder Icons	Pictures	Quick-Time Movies	List Boxes	Popup Menus	Controls	Boxes & Round Rect Boxes	Lines
BOOL	BOOL	BOOL	BOOL	BOOL	BOOL*	BOOL*		
TEXT	TEXT	TEXT	TEXT	TEXT	TEXT	TEXT	TEXT	TEXT
RGB	RGB	RGB	RGB	RGB	RGB	RGB	RGB	RGB
TEXT	TEXT	TEXT	TEXT	TEXT	TEXT	TEXT	TEXT	TEXT
STR [INT]	STR [INT]	STR [INT]	STR [INT]	STR [INT]	STR [INT]	STR [INT]	STR [INT]	STR [INT]
BOOL	BOOL	BOOL	BOOL	BOOL	BOOL	BOOL	BOOL	BOOL
				INT				
		STR	STR					
							STR [INT]	
		INT						
STR								
BOOL	BOOL	BOOL	BOOL	BOOL	BOOL	BOOL	BOOL	BOOL
INT	INT	INT	INT	INT	INT	INT		
INT	INT	INT	INT	INT	INT	INT	INT	INT
	BOOL							
	INT	INT						
INT								
STR	STR	STR	STR	STR	STR	STR	STR	STR
				BOOL				
						INT		
STR [INT]								
BOOL								
	STR	STR		STR [INT]				
						INT		
						INT		
STR	STR	STR	STR	STR	STR	STR	STR	STR

( ) alternate value type (your choice)
R/W read or write
R/O read only
* Supported by some, but not all, CDEFs and MDEFs

	Pg Ref	Prop. Picker Access	Read/ Write Status	Window	Push Buttons	Radio Buttons	Checkbox	Labels	Editable Text Fields
Number	43	Y	R/W	INT	INT	INT	INT	INT	INT
NumberOfItems	43	N	RO	INT					
Params	44	Y	R/W	TEXT					
Pen	44	Y	R/W						
PenSize	44	Y	R/W						
Properties	44	N	R/W	TEXT					
Rect	44	Y	R/W	INT	INT	INT	INT	INT	INT
Repeating	44	Y	R/W						
Result	44	Y	R/W	TEXT					
Script	45	Y	R/W	TEXT	TEXT	TEXT	TEXT	TEXT	TEXT
Scroll	45	N	R/W	INT					INT
Scrolling	45	Y	R/W	BOOL					BOOL
Selection	45	N	R/W						INT
SelectionColor	46	N	R/W						RGB
SelectionFont	46	N	R/W						STR
SelectionRect	46	N	RO						
SelectionSize	46	N	R/W						INT
SelectionStyle	46	N	R/W						STR
SelectionText	46	N	R/W						TEXT
SelectionTime	46	N	R/W						
SoundVolume	47	N	R/W						
Speed	47	N	R/W						
Step	47	Y	R/W						
Style	47	Y	R/W	STR					
StyleRun	48	N							STR
Text	48	Y	R/W						TEXT
TextAlign	48	Y	R/W						STR
TextFont	48	Y	R/W	STR	STR	STR	STR	STR	STR
TextOfSelection	48	N	R/W						TEXT
TextSize	48	Y	R/W	INT	INT	INT	INT	INT	INT
TextStyle	48	Y	R/W						STR
Time	48	Y	R/W						
TimeScale		N	RO						
TitleItem	49	Y	R/W						
TopLeft	49	Y	R/W	INT	INT	INT	INT	INT	INT
Value	49	N	R/W	TEXT	INT	INT	INT		TEXT
Version	49	N	RO	INT					
Visible	49	Y	R/W	BOOL	BOOL	BOOL	BOOL	BOOL	BOOL
WantsFocus	49	N	R/W						
WhiteColor	49	Y	R/W	RGB	RGB	RGB	RGB	RGB	RGB
Width	50	Y	R/W	INT	INT	INT	INT	INT	INT
Wordwrap	50	N	R/W						BOOL

BOOL	boolean (True or False)
INT	integer
RGB	can be either an RGB ("integer, integer, integer") or a color index (integer)
STR	single line string (max 255 characters)
TEXT	string which may contain multiple lines (max 32K characters)

Static Text Fields	ICONs & Finder Icons	Pictures	Quick-Time Movies	List Boxes	Popup Menus	Controls	Boxes & Round Rect Boxes	Lines
INT	INT	INT	INT	INT	INT	INT	INT	INT
							STR [INT]	STR [INT]
							INT	INT
INT	INT	INT	INT	INT	INT	INT	INT	INT
			BOOL					
TEXT	TEXT	TEXT	TEXT	TEXT	TEXT	TEXT	TEXT	TEXT
INT		INT	INT	INT				
BOOL		BOOL	BOOL	BOOL				
INT		INT	INT	INT	INT			
RGB								
STR								
				INT				
INT								
STR	STR	STR						
TEXT				TEXT				
			INT					
			INT					
			INT					
						INT		
				STR	STR	STR		
STR								
TEXT				TEXT	TEXT			
STR								
STR				STR	STR	STR	STR	
TEXT				TEXT	TEXT			
INT				INT	INT	INT	INT	
STR								
			INT					
			INT					
					INT			
INT	INT	INT	INT	INT	INT	INT	INT	INT
TEXT	STR	INT		INT	STR	INT		
BOOL	BOOL	BOOL	BOOL	BOOL	BOOL	BOOL	BOOL	BOOL
			BOOL					
RGB	RGB	RGB	RGB	RGB	RGB	RGB	RGB	RGB
INT	INT	INT	INT	INT	INT	INT	INT	INT
BOOL								

( ) alternate value type (your choice)
R/W read or write
R/O read only
* Supported by some, but not all, CDEFs and MDEFs

	Page Ref.	Window	Push Buttons	Radio Buttons	Check box	Labels	Editable Text Fields	Static Text Fields
AdjustSize	50	X	X	X	X	X	X	X
BringToFront	50	X						
Clear	50						X	X
Close		X						
Copy	50						X	X
Cut	50						X	X
DoUpdate	50	X	X	X	X	X	X	X
First	51							
GetBGFieldText	51						X	X
GetCdFieldText	51						X	X
Last	51							
Next	51							
Paste	51						X	X
Pause	51							
Play	51							
Prev	51							
SendToBack	51	X						
SetBGFieldText	NEW						X	X
SetCdFieldText	NEW						X	X
SnapShot	51	X						
Stop	52							
ZoomIn	52	X						
ZoomOut	52	X						

BOOL	boolean (True or False)
INT	integer
RGB	can be either an RGB ("integer, integer, integer") or a color index (integer)
STR	single line string (max 255 characters)
TEXT	string which may contain multiple lines (max 32K characters)
X	commands supported by item

	ICONs & Finder Icons	Pictures	Quick-Time Movies	List Boxes	Popup Menus	Controls	Boxes & Round Rect Boxes	Lines
AdjustSize	X	X	X	X	X			
BringToFront								
Clear			X					
Close								
Copy			X	X				
Cut			X					
DoUpdate	X	X	X	X	X	X	X	X
First			X					
GetBGFieldText								
GetCdFieldText								
Last			X					
Next			X					
Paste			X					
Pause			X					
Play			X					
Prev			X					
SendToBack								
SetBGFieldText								
SetCdFieldText								
SnapShot								
Stop			X					
ZoomIn								
ZoomOut								

BOOL boolean (True or False)
INT integer
RGB can be either an RGB ("integer, integer, integer") or a color index (integer)
STR single line string (max 255 characters)
TEXT string which may contain multiple lines (max 32K characters)
X commands supported by item

INDEX

WindowScript Limited Warranty
LIMITED WARRANTY
ON MEDIA AND REPLACEMENT

If you discover physical defects in the manual or media distributed with this product, Heizer Software will replace the media or manual at no charge to you, provided you return the item to be replaced with proof of purchase to Heizer Software during the 90-day period after you purchased this product.

ALL IMPLIED WARRANTIES ON THE MEDIA AND MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Heizer Software has tested the software and reviewed the documentation, HEIZER SOFTWARE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS SOFTWARE, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS SOFTWARE IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE. IN NO EVENT WILL HEIZER SOFTWARE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE SOFTWARE OR DOCUMENTATION, even if advised of the possibility of such damages. In particular, Heizer Software shall have no liability for any programs or data stored in or used with this product, including the costs of recovering such programs or data. Should the programs prove defective following their purchase, the buyer assumes the entire risk, including any and all necessary servicing, repair or correction, as well as any and all incidental or consequential damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Heizer Software employee, dealer, agent, or distributor is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

WindowScript™ *The Interface Design Studio*

The Ultimate Design Tool for Hypercard!

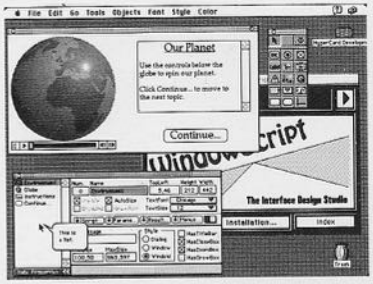
WindowScript is a tool for the design of Macintosh® user-interfaces within Claris HyperCard®. In the past there were HyperCard stacks and Macintosh applications — each with its own distinctive interface objects and feel. WindowScript brings to HyperCard the look and feel of a real Macintosh user-interface.

WindowScript is integrated into HyperCard — it's always available when you need it. WindowScript's interface editor is WYSIWYG — the objects you add are usable immediately, letting you test your interface as you develop it.

All standard objects in most Macintosh applications are available in this powerful interface design studio:

- ❖ Dialog boxes, windows and floating palettes.
- ❖ Scrolling lists of text, icons and pictures.
- ❖ QuickTime® movies.
- ❖ Standard, tear-off, color palette and pop-up menus.
- ❖ Text fields with mixed fonts, styles, sizes, even color text.
- ❖ Icons, including color Finder icons.
- ❖ Color pictures, including scrolling pictures.
- ❖ Standard radio buttons, round rect buttons and checkboxes.
- ❖ Simple graphic objects like lines and boxes.

All objects have properties which can be controlled through HyperTalk®. Object properties include color, System 7 Balloon Help®, location, style, labels, and object-specific properties such as text of a field, items in a menu, or selection logic of a list.



Some History

WindowScript is a new product from Leonard Buck of theResult Software, Inc. Buck is the author of Dialoger Professional, winner of the 1990 *MacWorld* SuperStacks Contest as Best Developer Tool. WindowScript, which offers far more power, ease of use and flexibility, replaces Dialoger Professional

Licensing Policy

If you create an interface using WindowScript, you must include the WindowScript XCMD when you distribute your stack. You must purchase a separate license for each product you distribute commercially that includes the WindowScript XCMD. Each license has a one-time, flat fee of $100. Shareware or public domain stacks need only display a splash screen (no fee is required). Stacks developed for "in house" or personal use do not require licensing.

System Requirements: 2 MB RAM. System 6.05 or higher. HyperCard 2.0v2 or higher. Any Macintosh that can run HyperCard, and a hard disk. Requires color QuickDraw® for QuickTime and System 7 to display Balloon Help.

Heizer Software

PO Box 232019
Pleasant Hill, CA 94523 **510-943-7667**