

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

OmniScript

the General Purpose Script Language

Richard G. Gibbs
158 Cranberry Rd.
North Attleboro, MA 02760

email: rgibbs@aol.com

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

License Agreement

This software is protected by the copyright laws of the United States and other countries. You are free to try it for 30 days. If after this time you decide to keep it you should send the \$25 dollar shareware fee to the author. This fee licenses you to use the OmniScript application and any droplet applications derived from it on one machine at a time.

TABLE OF CONTENTS

Introduction.....	3
What It Does.....	3
Scripts.....	5
Running OmniScript.....	5
Script Processing.....	5
Error Handling in Scripts.....	6
Variables.....	6
General Variables.....	6
File Variables.....	7
Alias Variables.....	8
Target Variables.....	8
Path Variables.....	8
Global and Local Variables.....	9
Saving Global Variables.....	9
Expressions.....	9
OmniScript Features.....	13
The OmniScript Display.....	13
Setup Script.....	14
Droplet Applications.....	15
Lists.....	15
Menus.....	16
Dialogs.....	16
Modal Dialogs.....	16
Modeless Dialogs.....	17
Dialog Definition Lists.....	18
Apple Events.....	20
Finder Control.....	22
File Tailoring.....	22
Application Control.....	23
SCRIPT Functions.....	25
Function Syntax.....	25
#ABS.....	25
#ACOS.....	25
#ASIN.....	25
#ARG.....	25
#ATAN.....	26
#ATAN2.....	26

#CHAR.....	26
#CONV.....	26
#COS.....	26
#COSH.....	26
#CQD.....	26
#DATE.....	27
#DOUB.....	27
#DT.....	27
#E.....	27
#EXP.....	28
#EXPR.....	28
#EXTE.....	28
#F.....	28
#FILE.....	28
#FORM.....	30
#FPU.....	30
#G.....	30
#LC.....	30
#LEN.....	30
#LOC.....	31
#LOG.....	31
#LOG10.....	31
#LONG.....	31
#MAC.....	31
#MENU.....	31
#NUM.....	31
#PATH.....	32
#PI.....	32
#RAND.....	32
#RET.....	32
#SCRIPT.....	32
#SEL.....	32
#SHORT.....	32
#SIN.....	33
#SING.....	33
#SINH.....	33
#SQRT.....	33
#SYS.....	33
#TAN.....	33

#TANH.....	33
#T.....	34
#TIME.....	34
#UC.....	34
SCRIPT Commands.....	35
Summary.....	35
Display Control.....	35
File and Folder Control.....	35
File Routines.....	35
Input.....	36
Lists.....	36
Menus.....	36
Script Control.....	36
Process Control and Apple	
Events.....	37
Standard File Package.....	37
Variables.....	37
Command Syntax.....	38
ACCEPT.....	38
AEADD.....	39
AELISTADD.....	39
AELISTNEW.....	39
AENew.....	39
AEOPT.....	39
AEREPLY.....	40
AESEND.....	40
AETARGET.....	40
ALIAS.....	41

BEEP.....	41	MENUSCRIPT.....	52
CHANGE.....	41	MENUSET.....	53
CLEAR.....	42	MODELESS.....	53
CLOSE.....	42	MOVE.....	53
CONVERT.....	43	NEWDIR.....	53
CURSOR.....	43	OPEN.....	53
CYCLE.....	43	PATH.....	54
DEBUG.....	43	PAUSE.....	56
DECREMENT.....	43	PUTFILE.....	56
DELETE.....	43	QUIT.....	56
DESELECT.....	44	READ.....	56
DIALOG.....	44	REMOVE.....	56
DIRECTORY.....	44	RENAME.....	56
DISPLAY.....	44	REORDER.....	57
DO.....	44	REPORT.....	57
DROPLET.....	45	REWIND.....	57
ELSE.....	45	RESOLVE.....	57
END.....	45	RESTART.....	57
ERROR.....	45	RFCLOSE.....	58
EXEC.....	45	RFOPEN.....	58
EXIT.....	45	SAVE.....	58
FILE.....	46	SCRIPT.....	58
FIND.....	46	SCRIPTPATH.....	59
FINDER.....	46	SELECT.....	59
FIXLIST.....	48	SET.....	59
FRONT.....	48	SETINFO.....	59
GETFILE.....	48	SORT.....	59
GOTO.....	49	SORTD.....	60
IF.....	49	SOUND.....	60
INCREMENT.....	49	SRAND.....	60
KILL.....	49	TAILOR.....	60
LAUNCH.....	49	TEST.....	61
LIST.....	50	UNMOUNT.....	61
LISTDIR.....	51	VALUES.....	61
LISTNEXT.....	51	VLOAD.....	61
LISTP.....	51	VSAVE.....	61
LOADSCRIPT.....	52	WAIT.....	61
LVALUES.....	52	WINDOW.....	62
MENU.....	52	WRITE.....	62

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

Resources.....63

Introduction

What It Does

OmniScript is a general purpose script language with the following features:

- Complete control structure including if and else statements and do loops.
- Use of string, integer or floating point variables. Subscripts can also be used.
- A display window that uses the Macintosh text edit functions. This enables a user to:
 - interrupt a script and scroll through the display. Data that scrolls off the screen is not lost.
 - use the text edit functions to modify the display and copy parts to other programs.
 - save the display to a file.
 - change the size and location of the display window.
- MultiFinder aware and System 7 compatible and can process in the background.

Some of the features of OmniScript are:

- Easily create lists that can be processed with the Macintosh List Manager.
- Add menus to OmniScript that can be used to set options or execute scripts.
- Easily process user defined dialogs (both modal and modeless).
- Create and delete files and folders. Change file information.
- Read and write data to and from files.
- Create, send and receive Apple Events.
- Launch and control other applications.
- Make a ‘droplet’ application, which executes a predefined script to process all files opened by the application.

SCRIPTS

Running OmniScript

A script is a TEXT type file containing a series of commands to process, one command per line. A script can be executed in one of five ways:

- i) by opening the ‘OmniScript’ application. The application attempts to execute the script file ‘Default Script’. If this file is in the same folder as ‘OmniScript’ and is not empty it will be executed, otherwise the Standard File Package is used to prompt the user for a script to execute.
- ii) by using a force open. The user selects one or more scripts and the ‘OmniScript’ application (by shift-clicking) then selects Open (Open from the File Menu, Command-O or double clicking on one of the selected icons). (This is for System 6 and does not work under System 7.)
- iii) by opening one or more OmniScript documents (scripts showing the OmniScript File icon). The user selects one or more OmniScript documents (by shift-clicking) then selects Open.
- iv) under System 7.0 by selecting one or more scripts and dragging over the OmniScript icon.
- v) under System 7.0 by using Apple Events to send scripts to the ‘OmniScript’ application. (See below.)

Any TEXT type file can be converted to an OmniScript document by changing the creator to 'ExPr'. The script ‘Change Creator.Setup’ is a script that can be used to make a droplet application to do this.

Script Processing

The general format of a command line is

[*label* ;] COMMAND *parameter1* , *parameter2* , ... !! comments

A command line is limited to 254 characters. Any line starting with an * is treated as a comment and ignored.

The label is optional. It is identified as a label by the semi-colon following it. It is permissible to code just a label on a command line. When the script file is first read into memory any label

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

is stripped off and a note made of the line to which it refers. Thus during command line processing the label is no longer present. The label name should follow the same rules as variable names, although it may also contain the underscore character.

Whenever a script is first referenced, whether by the ‘OmniScript’ application or by another script (using the EXEC or LOADSCRIPT commands), the file is first completely read into memory, any command line labels are identified, and all DO and IF commands are matched with the corresponding END commands. The processed script is saved in memory. For any subsequent invocations of the script via an EXEC command during this execution of the application the script file is not read again. However if a script is reopened (using the Open, Immediate or List commands on the File menu or by dragging a script and dropping it on to the application) it will be reloaded if it has been changed.

It is possible to include more than one script in a single file. The occurrence of a line starting with the key word **SCRIPT** terminates the current script and starts the pre-processing of another script with the name that follows the **SCRIPT** key word. Thus if an **EXEC** command subsequently references a script by this name it will already be in memory and will not be read in. (If the script named by the **SCRIPT** key word is the same as a script previously read in then the old script will be replaced with the new one. However any static local variables defined during the execution of the old script will be available to the new script .)

A script can be executed recursively, ie. it can **EXEC** itself. In this case the dynamic local variables defined in each execution are separate, but the static local variables are common to all executions.

The names of commands and functions and the various keywords are not case sensitive. However variable names are case sensitive.

Error Handling in Scripts

A fatal error occurs if the command processor detects a syntax error or does not receive the expected type of parameter. A non-fatal error occurs when those commands that receive an error code from the Macintosh operating system receive a non-zero error code or when a script executed by the **EXEC** command returns a non-zero value with the **EXIT** command.

Fatal errors cause the immediate termination of the script and those scripts in the chain of **EXECs** that invoked the script . If the application was opened with more than one script file (methods ii - v above) the application will execute the next script .

The handling of non-fatal errors depends on whether an error trap has been set. The **ERROR** command defines a label to jump to if a non-fatal error occurs. If no error trap has been set then execution continues with the next command. The error code value can be accessed using the **#RET** function.

The **REPORT** command displays information about a non-fatal error.

Variables

A variable name consists of an alphabetic character or one of the special characters (****, **@** or **\$**) followed by 0 to 30 alphanumeric characters. **Variable names are case sensitive.** There are four types of variable: General, File, Alias and Path.

A variable name may be indexed by adding the index in square brackets at the end of the name, for example **\$a[1]**. A variable may have more than one index, but each index reduces the

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

maximum length of the variable name by two characters. The index value can be any valid integer expression in the range -32768 to 32767.

The script language permits the indirect reference of variables, that is the name of a required variable can be contained in another variable or, more generally, in a string expression. This is indicated by enclosing the expression in braces.

General Variables

General variables are used to hold a text string or numeric value. The value of a general variable is flagged as being string, integer or floating point. A variable can have a null value, that is non-numeric with length zero.

When a variable has a string value it is possible, when evaluating an expression, to refer to a subrange of its value by following the variable name with the range enclosed in parentheses. The range consists of two integer expressions separated by a colon or a single expression to define a single character.

The following examples illustrate these rules. They also show how to reference indexed variable names. Suppose the following variables have been defined

I = 1

DE = 'XXX'

DEF = 'WXYZ'

DEF[1] = 'QRS'

G = 'DEF'

The following table gives the results of variable substitution

Initial string	Substitution	Comment
DEF	WXYZ	
DEF[I]	QRS	The variable name DEF[1] is first generated.
DEF(I:3)	WXY	The subrange is the first thru third characters
DEF(:2)	WX	The subrange is the first two characters
DEF(3:)	YZ	The subrange is the third and following characters
DEF(2)	X	The subrange is the second character.
DEF[I](2)	R	The value is the second character of DEF[1].
{G}	YYYY	The variable name DEF is first generated
{G}[I]	QRS	The variable name DEF[1] is first generated
{G[I]}	null	G[1] not defined
{G(1:2)}	XX	The variable name DE is generated from the first two characters of the value of G

File Variables

File variables are set by the GETFILE, LISTDIR and FILE commands. They actually define a general variable containing the name of a file (or possibly a directory for the FILE command) and a file variable of the same name that contains information about the file. (LISTDIR generates a list of file names and a corresponding list of file variables.) This information can be accessed using the #FILE function and modified by the CHANGE command. The SETINFO command is then used to actually apply the information to the file.

Whenever information about a file or directory is obtained it is stored in an internal buffer. Information to be saved as a file variable is copied from this buffer. The FILE, CHANGE and

SETINFO commands and the #FILE function can use this buffer directly instead of a file variable. However other commands may invalidate this buffer. Thus it is possible to use the FILE or GETFILE commands to set the buffer, followed by the CHANGE or SETINFO commands or #FILE function.

Alias Variables

Alias variables are used to hold alias information about files or directories. An alias variable is set by the ALIAS command. Information is retrieved using the RESOLVE command. Since aliases can only be used with System 7, these commands use full path names under System 6. (See the PATH command for a description of paths.)

Target Variables

Target variables are generated by the AETARGET command, which uses the PPC Browser function of System 7 to select a target for an Apple Event. These target variables can also be used to define items of type 'targ' in Apple Events.

Path Variables

Path variables are used to define directories. Various file operations are performed in the current default directory. This can be changed by the PATH command. When this command is used it is possible to save the information about the directory in a variable, thus simplifying references to this directory in subsequent PATH commands. The SCRIPTPATH command is similar to the PATH command but is used to define the directory to be searched for scripts.

At the start of execution of a script in the document list two path variables are predefined: APPL is the directory containing the application and SCRIPT is the directory containing the script. (For a droplet application SCRIPT is the directory containing the file.)

Under System 7 some path variables describing certain system folders are initialized. They are given in the following table.

Variable	Folder
amnu	Apple Menu Items
ctrl	Control Panels
desk	Desktop Folder
extn	Extensions
pref	Preferences
prnt	PrintMonitor Documents

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

empt	Shared, network Trash directory
trsh	Single-user Trash directory
strt	Startup Items
macs	System Folder
temp	Temporary Items

Global and Local Variables

If a general, file or alias variable name begins with a \$ or @ character it is a global variable otherwise it is local. The value of a local variable is available only in the script that defines it. Any other script that uses the same name defines a different local variable. The value of a global variable is available to all scripts. The @ type variables are intended for use with the VLOAD and VSAVE commands. (See below.) When the VSAVE command is executed all the @ type variables are saved to a file. The VLOAD command is used to read these variables back into memory. This allows the predefinition of a set of variables used by a script and can result in improved execution speed.

If a general, file or alias variable name begins with a \ character it is a static local variable, that is if a script is exited then executed again this variable will have the value left over from the previous execution. If a variable name does not begin with one of the special characters (\, @ or \$) then it is a dynamic local variable. Its value exists only during the current execution of the script.

Path variables are always global (equivalent to those variables beginning with a \$ character) and may not be saved with VSAVE.

Saving Global Variables

The VSAVE command is used to save the @ type variables to a file. For example the command

```
VSAVE 'Saved Variables'
```

saves the @ type variables to the file 'Saved Variables'. These variables could then be loaded in a subsequent execution of 'OmniScript' by

```
VLOAD 'Saved Variables'
```

It is possible to define up to 10 additional blocks of variables to be saved and loaded by the use of a special form of variable name. For example the command

```
@3_val = 25
```

defines a variable that would be saved by the command

```
VSAVE 'Additional Variables', 3
```

The number following the @ character can have a value in the range 0 - 9 (for a total of 10 additional blocks). These files of saved variables do not have to be loaded back with the same

number. For the above example the file could be loaded with

VLOAD 'Additional Variables', 5

In this case the variable defined above would now be referenced by @5_val.

Expressions

Several of the commands and functions use expressions. They can be used in the assignment of variable values, as parameters for commands, in variable subscripts and ranges, and in the arguments of script functions. An expression has the general form

field operator field operator ... field

The expression can be terminated by the end of the line or by a suitable delimiter as required by the syntax, such as a comma for a command or function parameter, or a right parenthesis for a function parameter or a nested expression.

A *field* may have a string, integer or floating point value and can be specified as a hard coded value (numeric literal or string literal in single or double quotes) or as the value of the variable specified by field or the value returned by a script function. The variable can have subscripts. If a string value is desired then a subrange can be specified. The variable can be referenced indirectly by the use of {}. An integer is stored as a 4 byte signed value.

If the operator is omitted then the two fields are treated as strings and concatenated. An integer field is converted to a string but a floating point field will generate an error. Comparison operations generate an integer value of 1 if true or 0 if false. Comparisons are valid between two numeric fields (even if one is integer and one is floating point) or two string fields. Expression evaluation can be nested with the use of parentheses.

The operators used in expressions are given in the following table in the order of their precedence for evaluation. Generally this is the same as the C language. Evaluation of an expression is performed from left to right.

operator	representation	precedence	notes
string concatenation	//	1	can usually be omitted
logical not	!	2	integer operand only
bitwise complement	~	2	integer operand only
exponent	**	3	
multiply	*	4	
divide	/	4	
remainder	%	4	
add	+	5	
subtract	-	5	
bitwise left shift	<<	6	two integer operands only
bitwise right shift	>>	6	two integer operands only
less than	<	7	

greater than	>	7	
less than or equal to	<= , =<	7	
greater than or equal to	>= , =>	7	
equal to	=, ==	8	
not equal to	<> , >< , !=	9	
bitwise and	&	10	two integer operands only
bitwise exclusive or	^	11	two integer operands only
bitwise or		12	two integer operands only
logical and	&&	13	numeric operands
logical or		14	numeric operands

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

OMNIScript FEATURES

The OmniScript Display

When the application is executing the display window is always shown. This is a standard Macintosh document window with size and zoom boxes and scroll bars. A small area of the window just below the title is used to display the current status, which can be one of the following:

- Select Script (when the user is prompted for the script file)
- Select File (when the user is prompted for a file from a droplet application)
- Executing Script
- Pause in Script
- List Selection (when the user is prompted to select from a list)
- Select continue ... (when the program was in the background but requires user input for list selection or a dialog)

During execution the Menu Bar displays the standard Apple and Edit menus and a File menu. The options in this menu are:

Quit	Immediately stop execution of OmniScript. If the application is not in a pause the user will first be prompted by an alert to confirm the Quit. This protects the user from inadvertently cancelling when running in the background.
Exit	If executing a script immediately terminate. If there are any scripts remaining in the list selected at start up the next is executed.
Pause	This suspends execution until the user selects another option from the File menu.
Continue	This continues execution following a Pause. The user may also press the ENTER key to continue.
Restart	If executing a script then the main script is restarted (ie. the script on the file list). If the program is in a Pause because there are no more scripts to execute the last script will be restarted.
Save	This causes the current contents of the display to be saved to a file and clears the display. If a file was not previously defined (using the SAVE command) then a default will be opened in the root directory of the start up volume. The name will start with 'Saved on' followed by the data and time
Page Mode	Normally when the display window is full additional output causes the

	display to scroll enough to display new text. Selecting Page Mode causes the display to be erased when it is full and the new text is displayed at the top of the window. This mode can display large amounts of data more quickly. This menu selection is used to toggle between Page Mode and the regular mode. When Page Mode is on it is checked in the menu.
Document List	This displays OmniScript's list of documents. This list includes those files opened when OmniScript is started, any documents added by selecting Open or Immediate from this menu, or any documents added by Apple Events. Documents not yet processed are shown highlighted. When the current script terminates the next highlighted document on the list is processed. By selecting or deselecting documents this order of execution can be modified, enabling some documents to be skipped or executed again.
Open	This prompts the user to select a script to be added at the end of the Document List.
Immediate	This prompts the user to select a script for immediate execution. The document is added to the Document List but it is executed immediately. The current script is interrupted and resumed when this interrupting script has terminated.
Command	The current script is interrupted and the user prompted to enter a single command that is immediately processed. This command could execute another script.
Terminate	This option is available only when OmniScript is waiting for an application to terminate following a Launch command with the wait option on. It immediately terminates the wait and the script continues.
Debug	This is used to turn debug on and off. When on each executed command is displayed (see the DEBUG command). This option toggles debug on and off. It is checked when on.
Execute Selection	Similar to the Command option of this menu except that instead of entering a command the command is the currently selected text in the display window. (If the selection includes a return character the command is the text up to the first return.)

The display software uses the TextEdit functions and as such is limited to a display of 32K bytes. An attempt to display information that exceeds this limit causes some of the display to be deleted and saved to a file. (This is like the Save Menu option except that not all of the display is deleted.)

The standard TextEdit functions can be used to paste some of the display into desk accessories or, if MultiFinder or System 7.0 is running, into other applications.

Setup Script

A setup script is a special script used to customize OmniScript. When OmniScript is launched it checks for a file with the name 'OmniScript.Setup'. This script is executed first and 'Default Script' is not used if no other files were opened with OmniScript.

The special use of the setup script is that after it is loaded the processed script is saved in the data fork of OmniScript. When OmniScript is executed again later the script does not need to be reloaded from OmniScript.Setup. It will only be reloaded if OmniScript.Setup is in the same folder as OmniScript and it has changed since it was last loaded.

It is possible to have several different customized versions by making and renaming copies of OmniScript. If it is renamed to XXX then the corresponding setup script is XXX.Setup

Droplet Applications

In general the term ‘droplet’ is used to describe an application that processes one or more files in a specified way and that supports ‘drag and drop’, where one or more files are selected and then dragged and dropped onto the application icon. OmniScript supports droplets by using a setup script that executes the DROPLET command. A droplet is created by making a copy of OmniScript, changing the name to whatever is desired, using the CONVERT command to define the droplet’s signature and the types of file it opens and executing the application so that the corresponding setup script is processed.

When a document is opened by the droplet (ie. added to the Document List by the various methods available) it is not executed as a script, but instead it is treated as a file to be processed by the script identified by the DROPLET command. The script defined on the DROPLET command is executed once for each file processed by the droplet. The general variable \$File contains the name of the file and the corresponding file variable contains the file information (see the #FILE function). The file is also the current file at the start of execution of the script.

When writing the setup script it should consist of two parts. The first part is the setup and should include the DROPLET command and any necessary initialization for the droplet, such as creating menus for example. Following this should be the script defined by the DROPLET command that is executed by the droplet for each file.

There are two droplet examples included (‘Change Creator’ and ‘File Info’).

Lists

Several commands make use of lists of general variables. Suppose the list name is chosen to be MYLIST. This variable is set to the number of items in the list (e.g. by using the SET command). The variable MYLIST[1] contains the first item in the list, MYLIST[2] the second and so on, up to a theoretical maximum of 32767 items.

One basic use of a list is to prompt the user to select items from a list. The LIST command creates a separate window and displays the list. The user can select as many items in the list as appropriate in the context. (The LIST command also has an option to restrict the user to selecting only one item.) If the user selects the OK button after selecting items from the list, the list will be updated to indicate the selected items. Following the LIST command the user can then process the list to see which items have been selected. The #SEL function is used to see if a particular item was selected. The LISTNEXT command is used to find which items were selected in order.

The items in a list can be manipulated directly by the SELECT and DESELECT commands to

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

change their selected status. When these commands have been used the LISTNEXT command won't work unless the FIXLIST command is used to reset the internal pointers in the list. This also applies if the value of any item in the list is changed. Note that the variable with index 0 (ie MYLIST[0] in the example) is used internally in the list structure and should not be directly referenced by the user. After the LIST command is completed the variable with index -1, eg. MYLIST[-1], contains the number of items selected.

If a list has had items flagged as selected then the LISTP command can be used to display the list. In this case the list will show these items as already selected.

The LISTDIR command creates a special list from the contents of the current default directory. This list can contain all items or may be filtered by file type and creator. In addition to the normal list of general variables containing the file names a corresponding list of file variables (with the

same name) is also created. When this list is displayed for user selection the corresponding file icon is also displayed.

A list can be sorted by using the SORT and SORTD commands (ascending and descending order respectively). This command can be used to sort a single list or several lists in the order determined by the sorting of the first list.

The REMOVE command can be used to remove an item from a list. The REORDER command can be used to change the position of an item in a list. (To insert an item in a list add the new item to the end of the list then use the REORDER command.)

Menus

A list can be used to define a menu. This is illustrated in the sample script 'Dialogs Lists and Menus'. The MENU command is used to create a menu from the list. Each item in the list defines a menu item. Meta characters can also be included. The user supplies numbers for the menuid and location and the list name as fields of this command. A location number of -1 implies a submenu or pop-up menu, otherwise the location number specifies where in the menu bar the menu appears. If the menu is to appear in the menu bar the menu name is derived from the list name by deleting the first character.

Menus defined this way can have several uses. The simplest use is to specify values to be used by a script. The item last selected from a menu can be checked by using the #MENU function. The MENUSET command can be used to specify an initial value to be returned by this function if the menu has not been selected.

The MENUSCRIPT command can be used to specify a script to be executed when an item is selected from a menu. This script interrupts the current script to execute. The menu id and item number are passed to the script as arguments.

A menu can be used as a pop-up menu in a dialog.

Dialogs

It is possible to use dialogs (both modal and modeless) in scripts. The DIALOG command displays a user defined modal dialog. The user must respond to this before the script can continue. The MODELESS command defines a modeless dialog. Execution of the script continues after the creation of a modeless dialog. If the user makes a selection from the dialog a script associated with the dialog is executed (interrupting the current script).

It is up to the user to create the DLOG and DITL (and possibly dctb) resources, typically by

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

using ResEdit. Dialogs can contain pop-up menus and lists. (These are defined in the dialog as user items.) Pop-up menus in a dialog do not execute scripts. They can only be used for selecting one item in the menu. This choice is tested by the #MENU function.

Modal Dialogs

The items of a dialog are controlled via a list associated with the dialog. Suppose a modal dialog has been created with DLOG resource id 10. The list \$dloglist specifies the dialog items. The dialog is displayed by the command

```
dialog $dloglist, 10
```

The dialog is displayed until the user selects a button. Up to that time the user may select radio buttons or check boxes, enter text in text edit items, or select from pop-up menus and lists. The return code from the dialog command is the item number of the selected button minus one. Thus item number 1 should be the OK button so that it returns a zero error code. Any other button generates a non-fatal error. If item 1 is a button pressing the Enter or Return key is equivalent to selecting this button. The state of radio buttons and check boxes and the value of text edit items are set in the dialog list. See the section ‘Dialog Definition Lists’ below. The example script ‘Dialogs Lists and Menus’ processes a dialog. It does nothing useful but demonstrates all the different options.

Modeless Dialogs

A modeless dialog would be created by the following example.

```
modeless new, 11, ModeScript , $dloglist
```

This command creates the modeless dialog described by the DLOG resource with id 11. The script associated with the dialog is ModeScript. This script is executed whenever a dialog button is selected. It is also executed if the mouse click was in a list and the list was defined so that the script is executed. When the dialog is created the script is executed with a value of 0 for the first argument. When the script is executed because a dialog button was selected or a list was selected the value of the first argument is the item number. If the item was a list then the second argument is set to 1 if the script is responding to a single click or 2 if responding to a double click. If a list item is set up to respond to both single and double clicks the script will be first executed with a single click and then for the double click.

Note: When the script is being executed the Macintosh event loop is not processed in order to avoid being interrupted by other selections in the dialog. Do not use the LIST command in this script.

If item 1 in the dialog is an enabled button it is treated as the response if the Return or Enter key is pressed. In this case the button is framed to show this.

Two other options are available when a button or its equivalent is selected. Before the script is executed it is possible to do an update, which means that the value of text edit items and the state of user defined lists is checked and the corresponding list variables are set to reflect the values. After the script is executed it is possible to regenerate the dialog, thus permitting the changing of item values or names or enabling or disabling items in the dialog.

If the dialog was defined as initially invisible it would not be shown until the following command was executed.

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.
modeless show, 11

To close the dialog either click in the close box (if it was defined with one) or execute the modeless command with the delete keyword, eg.

modeless delete, 11

When the dialog is closed the script is executed with a value of -1 for the first argument.

In this example the list \$dloglist controls the dialog items. See the section ‘Dialog Definition Lists’ below. The example script ‘Modeless Dialogs’ illustrates a modeless dialog.

Dialog Definition Lists

This section describes the use of a list to define the items of a dialog. In the example the list has the name \$dloglist. Thus the variable \$dloglist[1] describes item number 1 in the dialog. The following sections describe how to define the list item for each type of dialog item.

Except for text items the first character of the list item is used as a control character. If in the use of a modeless dialog the dialog is regenerated after execution of the script the items are individually checked to see if they are to be changed. The item will be changed only if the control character is upper case. After the regenerate operation the control character is explicitly changed to lower case. Thus the control character will have to be set to uppercase each time the item needs to be changed in a regenerate operation.

Button:

The first character is the control character. The rest of the list item is a new title for the button. It is not necessary to define this list item if the dialog template correctly describes the button.

If the control character has the value 'd' then the button is disabled, otherwise it is enabled.

For a modeless dialog the control character has the additional meanings. If it is 'u' an update is performed. If it is 'r' a regenerate is performed. If it is 'b' both are performed. For any other value neither is performed. However the script is always executed and the first argument is the item number. If the list item was not defined then no update or regenerate is performed except for the case where the button is item 1, in which case an update is performed.

Radio Buttons and Check Boxes

The first character is the control character. The rest of the list item is a new title for the dialog item. The control character has three possible values: 'd' means it is disabled, 'e' means it is enabled but not selected and 's' means it was selected. When the dialog is initialized (or a modeless dialog is regenerated) these control characters are read to set the item values. When one of these items is selected in the dialog the control character is immediately set to 's' or 'e' as appropriate. If the list item is not defined the control character will be initialized to 'e' for enabled items.

Radio buttons are meant to be used so that exactly one of a set of options is selected. To support this radio buttons should be defined in the dialog template in groups. A group is a set of consecutively numbered radio buttons. In the example dialog template items 3 and 4 form one group and items 6 and 7 form a second group. The program treats these as separate groups because item 5 is not a radio button. Whenever a radio button is selected the other buttons in

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

the group are deselected, so only one radio button in the group is shown as selected. The list describing the dialog should select exactly one button in each group to ensure proper initial appearance of each group.

Text Items

No control character is used for a text item. If the list item is initially specified it is used to replace the text specified in the template. For edit text items the value of the item is returned in the corresponding list item when a modal dialog is terminated via the OK button (item 1) or a modeless dialog does an update.

Pop-up Menus

A pop-up menu is defined by a user item. The list item is set to the id number of the menu, which must have been created previously with the MENU command.

The dialog item following the user item must be a static text item that defines the menu title. If the list item corresponding to the static text is undefined the menu title is the value of the text item defined in the dialog template. If the list item is defined the first character is a control character and the rest is the new title. A value of 'd' for the control character means that the menu will not appear.

When the user item is defined in the template its size does not have to be specified precisely. The right side of the rectangle will be adjusted to fit the menu width and the height will be adjusted to fit the text size. The title rectangle will also be adjusted. This item should be to the left of the menu. If it overlaps the menu it will be shrunk.

In a modeless dialog a menu is always regenerated whatever the case of the control character in the title.

Lists

A list is defined by a user item. The list will appear in the rectangle defined by the user item. A scroll bar will automatically be included if necessary. The dialog list item that describes this list consists of four control characters followed by the name of the list, eg.

```
$dloglist[16] = 'oB80$aaa'
```

This says that the items of the list are described by the list \$aaa. The third and fourth characters, '80' in the case, are the hexadecimal representation of the selfFlags byte of the ListRec structure defining the list (See Inside Macintosh, Vol. IV, page 267). In this case '80' means only one item can be selected. A value of '00' is the standard case with multiple selection.

The second control character controls the response to a single click in the list and is really meaningful only for a modeless dialog. The character has the following meanings. A value of 'n' means there is no response to a single click, that is the script is not executed. Any other character means the script is executed. A value of 'u' means there is an update, a value of 'r' means the dialog is regenerated and 'b' means both are done. Any other value means neither is done.

The case of the second control character is used to control list preselection. If it is lower case any items in the list marked as selected will appear in the dialog as selected.

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

The first control character controls the response to a double click in the list. The meanings of the control character value are the same as for the a single click. Additionally a value of 'o' means that a double click is treated as equivalent to selecting item one in the dialog if it is an enabled button. For a modal dialog the effective meanings of the control character are 'o' (item 1 button), 'n' (no action) or any other value equivalent to a button with item number the same as the user item.

Care should be taken if a list processes the script for both single and double clicks. If the user double clicks the script will be executed for a single click first and interrupted for the processing of the double click.

Remember that the case of the first control character is used to control regeneration of the list.

Apple Events

OmniScript contains several commands that enable the user to set up a variety of Apple Events in a fairly general way. An Apple Event basically is a list of parameters that is identified by an event class (such as ‘aevt’ for core Apple Events) and an event ID (such as ‘oapp’ to open an application). Also associated with the event is a target application.

The commands implemented allow a user to build an Apple Event with arbitrary event class and ID. The target can be specified by signature or type or can be selected using the PPC Browser, which enables the user to select applications on other Macs on the user’s network as the target.

An Apple Event is set up in the following way. First the user executes the AETARGET command to select the target using the PPC Browser if this is the desired method of selecting the target. Next the user executes the AENEW command, which initializes a new Apple Event with the event class and ID and the target.

The user may then add an arbitrary number of parameters to the event by using the AEADD command. Each parameter is identified by a type (such as ‘shor’ for a short integer or ‘alis’ for an alias record) that describes the nature of the data, an arbitrary keyword to identify the purpose of the data (such as the core open documents event where the keyword ‘----’ is used to identify the list of alias or file spec records describing the files to be opened).

Since an Apple Event parameter may itself be a list, two commands support the generation of lists. AELISTNEW initializes a new list and AELISTADD adds parameters to the list. The AEADD command is used to add the list to the Apple Event. These commands limit the user to building only one list at a time, and thus limit the nesting of lists. However this shouldn’t be a serious limitation in practice. The types of data recognized by the AEADD and AELISTADD commands are described below.

When the Apple Event has been built it is sent with the AESEND command. The event can also be sent with the LAUNCH command.

Because it is the user’s responsibility to dispose of the structures created for Apple Events a user list is deleted following the AEADD command that adds the list to an event and the Apple Event is deleted following the AESEND or LAUNCH command.

OmniScript has its own class of events (the class is ‘Expr’). The following table describes the possible Event IDs.

Event ID	Description
----------	-------------

'quit'	Terminates execution of the program
'exit'	Makes the program exit the current script (same as Exit menu choice)
'rstr'	Makes the program restart the current script (same as Restart menu choice)
'wait'	Makes the program enter the pause state (same as Pause menu choice)
'cont'	Makes the program leave the pause state (same as Continue menu choice)
'save'	Saves the display (same as Save menu choice)
'odoc'	Used to send a single script to the program to be added to the document list (like the Open menu choice). The event should contain a parameter that is of type 'fss ' or 'alis' with keyword '----'
'imed'	Same as 'odoc' except the script interrupts the current script and is executed immediately. This script can send data back to the application by using the AEREPLY command. The error code from the script can also be sent back by adding a parameter to this event that has a keyword '#ret' and type 'TEXT'. The value of this parameter is the name of the variable that is to contain the returned error code.
'cmnd'	Used to send a single line script command to be executed immediately. The event should contain a parameter that is of type 'TEXT' with keyword '----'. The error code from this command can be obtained in the same way as for the 'imed' type described above.
'gnrl'	This is used to send information between two different executions of OmniScript (on the same machine or across a network) The following table identifies the data types recognized by this event ID. These same data types are added to an event by using the AEADD and AELISTADD commands

The following table gives the data types recognized by the AEADD and AELISTADD commands and describes how they are processed when received by an event of class 'ExPr' and ID 'gnrl' or received as a reply to such an event. The general syntax of the two commands is

AEADD *keyword* , *type* , *data*

AELISTADD *type* , *data*

On receipt each parameter is decoded and assigned to a variable with name the same as *keyword* for a single parameter or the corresponding subscripted name for an item in a list.

Data Type	Data Description
'list'	<p>This is used by the AEADD command to add a list (created by AELISTNEW and AELISTADD) to an event. (This type cannot be used with AELISTADD.) No <i>data</i> is used with this type.</p> <p>On receipt the list is broken down into its component items and assigned to variables. The keyword is used as the list name. Each item is assigned to the corresponding list item. (For example, if the keyword was '\$abc' then the first item is assigned to variable \$abc[1] and the number of items in the list is assigned to variable \$abc.) The list items can be any of the recognized individual data types and can be mixed in the list.</p> <p>This is a standard Apple Event data type and can be sent to other applications.</p>
'varl'	<p>This is used by the AEADD command to create a list from a variable list (general variables only). (This type cannot be used with AELISTADD.) The <i>data</i> parameter is the name of the list to be copied. The data is added to the event as type 'list'. Each item in the list is of type 'vari'.</p>
'vars'	<p>This is similar to 'varl' except that the list is created from several variables rather than a single list. The <i>data</i> parameter consists of a sequence of variable names. Names are separated by commas.</p>
'vari'	<p>This is used to add a single general variable to an event or list. The <i>data</i> parameter is the name of the variable to be copied. On receipt the item is converted into a general variable.</p>
'alis'	<p>This is used to add an alias record to an event or list. The <i>data</i> parameter is the name of the alias variable to be copied. On receipt the item is converted into an alias variable.</p> <p>This is a standard Apple Event data type and can be sent to other applications.</p>
'targ'	<p>This is used to add a target record to an event or list. The <i>data</i> parameter is the name of the target variable to be copied. On receipt the item is converted into a target variable.</p> <p>This is a standard Apple Event data type and can be sent to other applications.</p>
'magn'	<p>These are standard Apple Event data types that are numeric values. The <i>data</i></p>

'long' 'shor' 'sing' 'doub' 'exte'	parameter is an expression that is evaluated to generate the numeric value. On receipt the item is converted into a numeric general variable. 'magn' is an unsigned long value, 'long' and 'shor' are 4 and 2 byte integer values and 'sing', 'doub' and 'exte' are 4, 8 and 10 byte floating point values.
'TEXT'	This is a standard Apple Event data type. The <i>data</i> parameter is the string that is the text of the parameter. On receipt the item is converted into a non-numeric general variable.
'qrv'	This is used to request the value of a general variable. The <i>data</i> parameter is the name of the variable. On receipt the value of the variable is retrieved and sent back in the reply as type 'vari'. The keyword of the item in the reply is the same as the original keyword. On receipt of the reply the item is converted into a general variable with name given by the original keyword. The original Apple Event should have been sent with the wait or queue parameter specified.
'qrvl'	This is like 'qrv' except that the request is for a general variable list instead of a single variable. The reply is of type 'list' with the same keyword and consists of the corresponding items of type 'vari'. On receipt of the reply the list is converted into a general variable list with name given by the original keyword. The original Apple Event should have been sent with the wait or queue parameter specified.
any other value	Arbitrary data types can be specified for sending to other applications. The <i>data</i> parameter is a string that is the data to be added.

The example script ‘Core Apple Open Event’ illustrates the construction of the core open documents Apple event.

Finder Control

OmniScript can control the Finder by sending Apple events to it. Use the Finder command to do this. The example script ‘Finder Play’ illustrates this. The example script ‘Folder Organizer’ illustrates organizing a Finder window.

File Tailoring

File tailoring is the process of generating an output text file from a skeleton by substituting general variables in each line of the skeleton. A skeleton is just a script file that includes the skeleton output lines. During command processing a line that starts with a right parenthesis, ‘)’, is treated as a file tailoring output line and the string expression following the right parenthesis is written to the file that has been identified by the TAILOR command as the file to receive this output. Since the skeleton is a script file it can include any command. Thus it is possible to conditionally process lines by using the IF command or include another skeleton by using the EXEC command. Typically the skeleton will use global variables that were set in other scripts.

Application Control

The Macintosh operating system lacks the equivalent of a Job Control Language that is used on other computers to control a user's data processing applications. OmniScript can be used to substitute for this lack of a JCL. Basically the idea is to design an application that reads input parameters from a text file. Based on the user's input OmniScript would create this text file (using File Tailoring) then it would launch the application and wait for it to terminate, at which point it could repeat the process with the next set of input parameters. A useful skeleton for such an application, that permits easy definition of files to be opened and easy specification of parameter values and that can process in the background, is available separately from the author as a Think C project.

SCRIPT FUNCTIONS

Function Syntax

Script functions are used in expressions. They return string, integer or floating point values. They are identified by the # character.

In the following description of each function the elements of the syntax line have the following meanings.

Square brackets [] indicate an optional quantity. *variable* represents the name of a variable. *string* is a string expression. *numeric* is an expression with a floating point value (an integer value will be converted). *integer* is an expression with an integer value.

For those functions that are not followed by arguments the function name can be delimited by any character not recognized as legal in a variable name. For those functions that are followed by arguments the function name is delimited by a space or the left parenthesis preceding the arguments.

#ABS

syntax: #ABS(*numeric*)
returns: floating point or integer: same as argument

This function returns the absolute value of the argument. It returns an integer if the argument is an integer.

#ACOS

syntax: #ACOS(*numeric*)
returns: floating point

This function returns the arc cosine of the argument.

#ASIN

syntax: #ASIN(*numeric*)
returns: floating point

This function returns the arc sine of the argument.

#ARG

syntax: #ARG(*integer*)
returns: depends on argument

This function is used to obtain the arguments passed to a script invoked by the EXEC command. #ARG(0) is the number of arguments passed to the routine. #ARG(1) is the value of the first argument, etc.

If the function argument is non-integer or less than 0 a fatal error is generated. If *integer* is greater than the number of arguments then a null string is returned.

#ATAN

syntax: #ATAN(*numeric*)
returns: floating point

This function returns the arc tangent of the argument.

#ATAN2

syntax: #ATAN2(*num1* , *num2*)
returns: floating point

This function returns the arc tangent of *num1/num2*, using the signs of both arguments to determine the quadrant of the returned value.

#CHAR

syntax: #CHAR(*integer*)
returns: string (one character)

This function generates a single character with ASCII value given by *integer* , which should be in the range 0 to 255. As a short cut the #T function generates a tab character instead of writing out #CHAR(9)

#CONV

syntax: #CONV(*integer*)
returns: string

This function formats the value of *integer* into a date and time. It would be used to convert the file dates (such as creation, modification and backup) from their internal integer values to a date and time. The output is a 20 character string in the format MM/DD/YY at HH:MM:SS, unless integer is 0, in which case the single character 0 is returned.

#COS

syntax: #COS(*numeric*)
returns: floating point

This function returns the cosine of the argument.

#COSH

syntax: #COSH(*numeric*)
returns: floating point

This function returns the hyperbolic cosine of the argument.

#CQD

syntax: #CQD
returns: integer

This function returns 1 if the Macintosh has color QuickDraw otherwise 0.

#DATE

syntax: #DATE
returns: string

This function returns the date in an 8 character field in the form YY/MM/DD.

#DOUB

syntax: #DOUB(*numeric*)
returns: string

This function converts the argument to an 8 byte floating point value and returns it as an 8 byte string.

#DT

syntax: #DT
returns:

This function returns the date and time in a 20 character field in the form YY/MM/DD at HH:MM:SS.

#E

syntax: #E(*numeric* , [*integer*] , [*integer*])
returns: string

This function formats the value *numeric* into a string. The first integer is the length of the string generated (if omitted then the length of the string will be the minimum necessary). The second is the precision used (if omitted 0 is used). The formatting is done using the C sprintf function and the e format. The #F and #G functions are similar, but use the f and g formats. The following table describes the differences

E	Generates exponential format, ie 1.23e1. The precision is the number of digits after the decimal point. If the precision is zero no decimal point is printed.
F	Generates fixed point format, ie 12.30. The precision is the number of digits after the decimal point. If the precision is zero no decimal point is printed.
G	This format chooses an exponential format or fixed point format depending on the size of the number. The precision is the total

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

	number of significant digits printed. If the precision is zero one digit is printed. Trailing zeroes are not printed.
--	---

More general formatting of floating point numbers can be done with the #FORM function.

#EXP

syntax: #EXP(*numeric*)
returns: floating point

This function returns e to the power of the argument.

#EXPR

syntax: #EXPR(*string*)
returns: depends on string

This function treats *string* as an expression and evaluates it. This could be used if the value of a variable is assigned by reading from a file and this value is an expression. As a simple example if the variable *a* has the string value '3 + 5' then #EXPR(*a*) returns the integer value 8. This function can also be used after the accept command to convert the input to numeric.

#EXTE

syntax: #EXTE(*numeric*)
returns: string

This function converts the argument to an extended floating point value and returns it as a 10 byte string.

#F

syntax: #F(*numeric* , [*integer*] , [*integer*])
returns: string

See the #E function.

#FILE

syntax: #FILE(*type* [, *variable*])
returns: integer except as noted in table

This function returns information about a file. If the *var* parameter is present the information is taken from the file variable *var* , otherwise it is taken from the current file, as defined by the FILE or GET commands. The field *type* is a literal string, not a string expression, so quotes are not necessary. The possible values of the field *type* and the corresponding value returned are given in the following table. The field *type* may be abbreviated to three or more characters

alias	1 if file is an alias: otherwise 0
invisible	1 if file is invisible: otherwise 0
bundle	1 if file has a bundle: otherwise 0
locked	1 if file is locked: otherwise 0
stationery	1 if file is a stationery file: otherwise 0
custom	1 if file has a custom icon: otherwise 0
inited	1 if file has been inited: otherwise 0
shared	1 if file is an application available to multiple users: otherwise 0
color	an integer in the range 0 to 7 representing the color used to display the icon on the desktop
vertical	vertical coordinate of file/folder's location
horizontal	horizontal coordinate of file/folder's location
type	4 character string: file's type ('DIR ' for a folder)
creator	4 character string: file's creator ('DIR ' for a folder)
dirid	file/folder's directory id
len	length of data fork
reslen	length of resource fork
date	creation date
mod	modification date
backup	backup date
parid	directory id of folder containing this file/folder
info	1 for a file: -1 for a folder
view	for a folder returns a 4 character string describing the view for this folder when open on the desktop. Values are SMAL, NAME, DATE, SIZE, KIND, ICON, LABL
top	for a folder the top coordinate of the rectangle that describes the folder's window when the folder is open on the desktop
left	for a folder the left coordinate of the rectangle that describes the folder's window when the folder is open on the desktop
bottom	for a folder the bottom coordinate of the rectangle that

	describes the folder's window when the folder is open on the desktop
right	for a folder the right coordinate of the rectangle that describes the folder's window when the folder is open on the desktop
open	for a file returns the path number if open otherwise 0
fork	for a file returns 1 if the data fork is open, 2 if the resource fork is open, 3 if both forks open or 0 if neither fork is open

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

#FORM

syntax: #FORM(*string*, *numeric*)
returns: string

This function formats a floating point value using the C function `sprintf` with the format defined by *string*. (See documentation of the C language for format specifics.) This format should specify only one floating point value to be formatted. The format string is an expression and does not have to be a hard coded literal string. This permits the easy use of variable values to give field size and precision. For example

#FORM('%#+ ' I ' ' J 'f,F)

would print the floating point value in the variable `F` using the `f` format with field size and precision specified by the variables `I` and `J` and would also force printing of trailing zeroes and a leading plus sign.

#FPU

syntax: #FPU
returns: integer

This function returns 1 if the Macintosh has a floating point unit, 0 if it does not.

#G

syntax: #G(*numeric* , [*integer*] , [*integer*])
returns: string

See the `#E` function.

#LC

syntax: #LC(*string*)
returns: string

This function converts *string* to lower case. This could be used to avoid case sensitivity in string comparisons.

#LEN

syntax: #LEN(*string*)
returns: integer

This returns the length of *string expression*. If the expression is simply a variable name then it returns the length of the value of that variable.

#LOC

syntax: #LOC(*string1* , *string2*)
returns: integer

This returns the first occurrence of *string1* in *string2* . The value returned is the character position in *string2* of the first character of *string1* where the match is found. If it is not found a value of zero is returned.

#LOG

syntax: #LOG(*numeric*)
returns: floating point

This function returns the natural logarithm of the argument.

#LOG10

syntax: #LOG10(*numeric*)
returns: floating point

This function returns the base-10 logarithm of the argument.

#LONG

syntax: #LONG(*integer*)
returns: string

This function converts the argument to a 4 byte integer value and returns it as a 4 byte string.

#MAC

syntax: #MAC
returns: integer

This function returns a code for the type of Macintosh the program is running on.

#MENU

syntax: #MENU(*integer*)
returns: integer

This returns the item number last selected from the menu with id *integer* if the menu was

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

created by the MENU command. If no item has been selected a value of 0 is returned.

#NUM

syntax: #NUM(*variable*)

returns: integer

This tests whether the value of *variable* is numeric and returns 1 if it is an integer, -1 if it is a floating point number or 0 otherwise.

#PATH

syntax: #PATH
returns: string

This returns the full path name of the current default directory.

#PI

syntax: #PI
returns: floating point

This returns the value of pi.

#RAND

syntax: #RAND
returns: integer

This function returns a random integer in the range 0 to 32767. The function generates a sequence based on a seed. The SRAND command sets the seed, which defaults to 1. To return the same sequence as in previous calls set the seed to the same value.

#RET

syntax: #RET
returns: integer

This returns the current value of the error code.

#SCRIPT

syntax: #SCRIPT
returns: string

This returns the name of the script currently being executed.

#SEL

syntax: #SEL(*variable*)
returns: integer

This tests whether the variable *var* was selected during list selection and returns -1 if it was or

0 if it wasn't.

#SHORT

syntax: #SHORT(*integer*)
returns: string

This function converts the argument to a 2 byte integer value and returns it as a 2 byte string.

#SIN

syntax: #SIN(*numeric*)
returns: floating point

This function returns the sine of the argument.

#SING

syntax: #SING(*numeric*)
returns: string

This function converts the argument to a 4 byte floating point value and returns it as a 4 byte string.

#SINH

syntax: #SINH(*numeric*)
returns: floating point

This function returns the hyperbolic sine of the argument.

#SQRT

syntax: #SQRT(*numeric*)
returns: floating point

This function returns the square root of the argument.

#SYS

syntax: #SYS
returns: string (4 character)

This function returns the system software version number, eg. 0604 for 6.0.4 or 0701 for 7.0.1.

#TAN

syntax: #TAN(*numeric*)
returns: floating point

This function returns the tangent of the argument.

#TANH

syntax: #TANH(*numeric*)
returns: floating point

This function returns the hyperbolic tangent of the argument.

#T

syntax: #T
returns: string

This function returns the tab character. It's easier than #CHAR(9)! It's useful for writing tabs to a file.

#TIME

syntax: #TIME
returns: string

This function returns the time in an 8 character field in the form HH:MM:SS.

#UC

syntax: #UC(*string*)
returns: string

This function converts *string* to upper case. This could be used to avoid case sensitivity in string comparisons.

SCRIPT Commands

Summary

The following is a summary of the various commands.

Display Control

CURSOR	Used to display a rotating cursor during long periods of processing
DISPLAY	Displays a line of text
PAUSE	Causes the application to pause and optionally display some text
SAVE	Indicates that the screen display should be saved to a file.
WINDOW	Change size and position of display windows

File and Folder Control

ALIAS	Makes an alias variable
CHANGE	Changes information about a file (Use SETINFO to complete changes)
CONVERT	Used to define a droplet's signature and file types
DELETE	Delete a file or empty directory
FILE	Gets information about a file or directory
FIND	Finds the application corresponding to a given signature
MOVE	Move a file or directory
NEWDIR	Create a new directory
PATH	Changes the current default directory
SCRIPTPATH	Changes the directory used to find the script in an EXEC command
RENAME	Renames a file or directory
RESOLVE	Resolves an alias variable into file name and directory
SETINFO	Changes file information
UNMOUNT	Unmounts a volume

File Routines

CLOSE	Close a file opened by the OPEN command
OPEN	Opens a file
READ	Reads text from a file

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

REWIND	Sets file position to beginning of file
RFCLOSE	Close a resource file opened by RFOPEN
RFOPEN	Opens a resource file
TAILOR	Defines file to receive tailored output
WRITE	Writes text to file

Input

ACCEPT	Prompts user to enter a string
DIALOG	Prompts user for information via a user defined dialog
MODELESS	Create a modeless dialog

Lists

DESELECT	Marks a list item as not selected
FIXLIST	Used after SELECT or DESELECT to reset pointers so that LISTNEXT works correctly
LIST	Prompts user to make selections from a list (See also LISTNEXT)
LISTDIR	Makes a list of the files and directories in the current default directory
LISTNEXT	Successively returns selected list items following LIST command
LISTP	Like LIST but displays preselected items in list (See SELECT and DESELECT)
REMOVE	Remove an item from a list
REORDER	Change the position of an item in a list
SELECT	Marks a list item as selected
SORT	Sorts one or more lists in ascending order
SORTD	Sorts one or more lists in descending order

Menus

MENU	Creates a menu from a list or deletes a menu
MENUSCRIPT	Defines a script to be executed when an item is selected from a menu created by MENU
MENUSET	Sets the value to be returned by the #MENU function. This value is also set by selecting an item from a menu.

Script Control

BEEP	Makes the Macintosh beep
CYCLE	Starts next iteration of a DO loop
DEBUG	Displays each command as it is executed
DO	Start a DO loop

DROPLET	Used in a droplet's Setup script to convert application to a droplet
ELSE	Used in an IF block
END	Terminates an IF block or DO loop
ERROR	Defines a label to jump to if a command sets the return code non-zero
EXEC	Execute another script
EXIT	Return to the script that called this one
GOTO	Jump to a label
IF	Start an IF block
KILL	Immediately terminates OmniScript
LOADSCRIPT	Loads a script into memory
QUIT	Immediately terminate a DO loop
REPORT	Displays information about last statement to set error code
SOUND	Plays the sound from the specified 'snd' resource
SRAND	Set random number seed
TEST	Conditionally executes a command
WAIT	Suspends execution for a specified time

Process Control and Apple Events

AEADD	Add parameter to current Apple Event
AELISTADD	Add parameter to current Apple Event list
AELISTNEW	Define new Apple Event list
AENEW	Define new Apple Event
AEOPT	Define optional parameters for an Apple Event
AEREPLY	Add parameter to reply Apple Event
AESEND	Send Apple Event
AETARGET	Define target for Apple Event
FIND	Find the application with the specified signature
FINDER	Control the Finder by sending Apple Events
FRONT	Bring an application to the front
LAUNCH	Launch an application, optionally with an Apple Event

Standard File Package

DIRECTORY	Prompt the user to select a directory
GETFILE	Prompt the user to select an input file
PUTFILE	Prompt the user to select an output file

Variables

CLEAR	Clears a list of variables (gives them null values)
DECREMENT	Decreases the value of a numeric variable by one
INCREMENT	Increases the value of a numeric variable by one
LVALUES	Generates a list from the component fields of a string
SET	Assigns a numeric or string value to a variable
VALUES	Splits a string into its component fields
VLOAD	Loads a set of @ type global variables from a file
VSAVE	Saves all of the @ type global variables to a file

Command Syntax

A command consists of a command name possibly followed by parameters separated by commas. In the following documentation a command is in general described in the following format.

COMMAND *parm1*, *parm2*, *parm3* [, *parm4*]

This example represents a command that takes 3 required parameters and an optional fourth parameter. A parameter can be one of five types as described in the following table.

<i>variable</i>	The parameter is the name of a variable. This name can be indirectly referenced by using {} in the same way as in expressions. However the resulting variable name cannot have a range specification.
<i>integer</i>	The parameter is an integer expression (terminated by a comma or end of command). It could simply be a hard coded value, a variable that has an integer value, or it could be an actual integer expression
<i>string</i>	The parameter is a string expression (terminated by a comma or end of command). It could simply be a literal value (in quotes), a variable that has a string or integer value, or it could be an actual expression with a string or integer value. If the expression has an integer value it is converted to a string.
<i>key word</i>	The parameter is a literal value and does not have to be in quotes. However a string expression enclosed in parentheses can be used if required. The keyword must be chosen from a list of possible values. Sometimes the keyword will be followed by = and an expression (string or integer as required).
<i>field</i>	The parameter is a literal value and does not have to be in quotes. However a string expression enclosed in parentheses can be used if required.

Some commands may be abbreviated. The command syntax line has the abbreviation underlined in the full name.

ACCEPT

syntax: ACCEPT *var* [, *string*]
error code: set

This command causes a dialog to be displayed to prompt the user for input from the keyboard.

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

string is displayed as a message to prompt the user for the type of input desired. The user's reply is stored in the variable *var* as a string. If the user selects cancel the error code is set to 1. If numeric input is required use the `#EXPR` function on the returned value.

AEADD

syntax: AEADD *key* , *type* [, *data*]
error code: set

See the section ‘Apple Events’ for a description of the use of this command. The error code will be set to the error value returned by any of the Apple Event Routines.

AELISTADD

syntax: AELISTADD *type* , *data*
error code: set

See the section ‘Apple Events’ for a description of the use of this command. The error code will be set to the error value returned by any of the Apple Event Routines.

AELISTNEW

syntax: AELISTNEW
error code: set

A new Apple Event list is initialized. The error code will be set to the error value returned by any of the Apple Event Routines.

AENEW

syntax: AENEW *class* , *id* , *type* [= *target*]
error code: set

The *class* parameter is a 4 character string that is the event class. The parameter *id* is a 4 character string that is the event id. The parameter *type* is a keyword that indicates target type. The following table gives the possible values of *type* and the corresponding value of *target* .

self	The target is this execution of OmniScript. The <i>target</i> parameter is omitted
sign	The <i>target</i> parameter is the 4 character signature of the application that is the target. For example 'ExPr' for OmniScript or 'MSWD' for Microsoft Word.
type	The <i>target</i> parameter is the 8 character combined type and signature of the target. For example 'FNDRMACS' for the Finder or 'PROJKAHL' for a Think C project.
targ	The <i>target</i> parameter is the name of a target variable defined by the AETARGET command.

AEOPT

syntax: AEOPT *string1* [, *string2*, ...]
error code: set

This command creates the 'optk' attribute of the Apple Event. It is the list of parameters in the event that are not required. Each *stringx* parameter in the command is the 4 character keyword of an event parameter that is to be treated as optional rather than required. If there are any optional event parameters this command should be called once after the AENEW command with the list of all the optional parameter keywords.

AEREPLY

syntax: AEREPLY *key* , *type* [, *data*]
error code: set

This command can be used in a script that is sent from one execution of OmniScript to another using the 'ExPr' event class and 'imed' event id. It adds data to a reply Apple Event. Its use is the same as AEADD except that the data is added to the reply instead of the Apple Event the user is creating.

AESEND

syntax: AESEND [*key1* , *key2* , ...]
error code: set

This command sends the Apple Event that has been created by the other Apple Event commands. It takes a number of optional *key word* parameters as specified in the following table. Generally these keywords set the corresponding flags in the Apple Event routine AESend (see Inside Macintosh VI).

no	kAENoReply. No reply wanted (the default)
queue	kAEQueueReply. The reply will appear in the event queue
wait	kAEWaitReply. OmniScript will wait for a reply. The time limit for the wait (in ticks) is specified in the time = parameter
never	kAENeverInteract
can	kAECanInteract
always	kAEAlwaysInteract
switch	kAECanSwitchLayer
dont	kAEDontReconnect
receipt	kAEWantReceipt
high	high priority: the event goes to the front of the queue

time =	The = is followed by an integer expression for the number of ticks to wait if wait or receipt is specified.
--------	---

AETARGET

syntax: AETARGET *variable*
error code: set

This command is used to select a target for an Apple Event by using the PPC Browser. If the user selects a target it will be stored in the target variable with name specified by *variable*. The error code is set to 1 if the user selects cancel in the dialog.

ALIAS

syntax: ALIAS *alias* = [*path*], *string*
error code: set

This command is used to create an alias variable describing a file or folder. The syntax is similar to the PATH command, except that the first variable name is not optional. The name of the target is given in the expression *string*. This can be a full path name or a path name relative to the folder given in the path variable *path* or, if *path* is omitted a path name relative to the current default folder. See the PATH command for further explanation.

BEEP

syntax: BEEP
error code: unchanged

This command causes the Macintosh to beep.

CHANGE

syntax: CHANGE [var = *target* ,] [use = *source* ,] *keyword list*
error code: unchanged

This command is used to change information about a file. The changes to the file are actually made by the SETINFO command. This command is used to set the file information in a file variable or the information for the current file prior to calling SETINFO.

If the *target* parameter is specified the CHANGE command alters the file information in the file variable with this name. If the *target* parameter is omitted the command alters the file information for the current file. Generally the values set by the CHANGE command can be either given explicitly or copied from the source file information, which is a file variable specified by the *source* parameter or the current file if the *source* parameter is omitted. If the target of the command is a file variable, but this variable does not already exist, it is initialized by copying the source file information.

The values to be set by the command are in the *keyword list*., which is a list of keywords separated by commas, generally in the form

keyword [= *value*], keyword [= *value*], ...

If *value* is omitted then the new value is copied from the source file information. The following table describes the possible keywords. Each keyword can be abbreviated to three

characters.

Function	Keywords	Values
Copy all of source file information to target file information	ALL	None
Set the Finder attributes contained in the fdFlags variable in the FInfo record	ALIAS INVISIBLE BUNDLE LOCKED STATIONERY CUSTOM INITED SHARED	A zero value clears the corresponding bit. A non-zero value sets it.
Set the icon color displayed by the Finder. This is the same as setting the color using the Label (System 7) or Color (System 6) menu	COLOR	A value in the range 0 to 7. A zero value means no color/label. A value of 7 means the first color in the menu.
Position of icon when displayed by Finder in Icon or Small Icon view	HORIZONTAL VERTICAL	The coordinates of the icon
The rectangle a folder is displayed in when opened by the Finder	TOP LEFT BOTTOM RIGHT	The coordinates of the rectangle.
File type and creator	TYPE CREATOR	The value is a four character string
File creation, modification and backup dates	DATE MOD BACKUP	These dates are specified internally as unsigned long integers. Here the equivalent signed values are used, which generally means a negative value since the largest positive date is in 1972.
The view used by the Finder to display a folder, corresponding to the View menu.	VIEW	A string corresponding to the menu items in the View menu. However only the first two characters are significant and they are not case sensitive.

CLEAR

syntax: CLEAR *var1* [, *var2* , *var3*]
error code: unchanged

This command clears the values of the listed variables, that is each variable is set non-numeric with a string value of length zero.

CLOSE

syntax: CLOSE *unit1* [, *unit2* , ...]
error code: set

This closes the files previously opened with unit number given by the expressions *unit1*, *unit2*, etc.

CONVERT

syntax: `CONVERT file , signature [, types]`
error code: set

This command is used when creating a droplet application. After making a copy of OmniScript and renaming it this command can be used to change the signature and the types of file the droplet can process. The string expression *file* is the name of the new droplet. The string expression *signature* is the 4 character signature of the application. The optional string expression *types* is a list of 4 character file types. Use '****' so see all files. Use '****diskfold' to see all files, folders and disks.

CURSOR

syntax: `CURSOR INIT / WAIT`
error code: unchanged

This command is used to display a spinning beach ball cursor during a time consuming process to let the user know that something is happening. Everytime the command is called with the WAIT keyword the cursor is rotated. At the end of the process the command is called with the INIT keyword to restore the arrow cursor.

CYCLE

syntax: `CYCLE`
error code: unchanged

This command is used within a DO loop to stop processing the current iteration of the loop and start the next iteration. This would often be used with the TEST command for conditional execution.

DEBUG

syntax: `DEBUG ON/OFF`
error code: unchanged

This command controls diagnostic display of commands during script development. If any command sets the error code to a non-zero value, this value is displayed when debug is on.

DECREMENT

syntax: DECREMENT *var*
error code: unchanged

If *var* exists and is numeric its value is decremented by one otherwise *var* is initialized to minus one.

DELETE

syntax: DELETE *string*
error code: set

This command attempts to delete the file or empty directory with name *string*, which can be either a full path name or a partial path name relative to the current default directory.

DESELECT

syntax: DESELECT *var*
error code: unchanged

This command is used to mark a list item as not selected. See the section ‘Lists’.

DIALOG

syntax: DIALOG *var* , *num*
error code: set

See the section ‘Dialogs.’

DIRECTORY

syntax: DIRECTORY
error code: set

The user is prompted to select a directory, which becomes the current default directory. The prompt uses the Standard File Package Get function with a modified dialog. Only directories are shown in the list. If a directory in the list is highlighted then the open and directory buttons are enabled. Selecting the directory button selects this directory and the function terminates. Selecting the open button opens this directory and the list of directories in this directory is displayed. The select current directory button is used to select the directory displayed above the directory list. Pressing return is equivalent to selecting the directory button. Double clicking on a directory in the list is equivalent to the open button for the directory.

Under System 7, if the alias of a directory is selected in the list, the directory button is disabled but the open button is enabled. To select the directory select the open button then the select current directory button. This forces the alias to be resolved. Note that when an alias is selected pressing return or enter is equivalent to selecting the open button.

DISPLAY

syntax: DISPLAY*string*
error code: unchanged

This command displays the string *string*.

DO

syntax: DO [WHILE *expression*]
error code: unchanged

The DO command enables the user to iterate through a block of commands. The block is terminated by the corresponding END command.

If the WHILE parameter is omitted the loop repeats until it is terminated by either an EXIT or QUIT command. If the WHILE parameter is included then the loop will be repeated while the *expression* is non-zero.

DROPLET

syntax: DROPLET *script*
error code: unchanged

This command converts OmniScript into a droplet application. The field *script* is the name of the script to execute when the application is run as a droplet.

ELSE

syntax: ELSE [IF *expression*]
error code: unchanged

See the IF command.

END

syntax: END
error code: unchanged

The END command terminates an IF/ELSE block or a DO block.

ERROR

syntax: ERROR [*field*]
error code: unchanged

This command defines the label to which control is transferred if a non-fatal error occurs. If *field* is omitted then any previous error trap definition is cancelled.

EXEC

syntax: EXEC *script* [, *arg1* , *arg2* ...]
error code: set

This command executes the script with name given by the field *script*. If the script has not already been loaded the application tries to load it from the file of the same name in the folder defined by the SCRIPTPATH command. If the SCRIPTPATH command has not been executed the folder which contained the main script in the document list is searched. If the file is not found then the System Folder (if any) on the same volume will be searched.

Any number of arguments may be passed in the EXEC command. These arguments are

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.
accessed by using the #ARG function.

The error code is set by the EXIT command that terminates the executed script.

EXIT

syntax:	EXIT [<i>expression</i>]
error code:	set

This command is used to immediately terminate processing of the current script. If this was a called script execution continues in the calling script. If this was the main script execution terminates and the application checks if there are any remaining scripts to be executed. For a called script if *expression* is specified then the error code is set to the value of *expression*, otherwise it is set to zero.

FILE

syntax: FILE [*var* =] *expression* [,*string*]
error code: set

This command gets information about a file or directory from the current directory. The value of *expression* is used as an index to the alphabetically ordered list of files and directories in the current directory. If this index exceeds the total number of files and directories in the current directory an error is returned. If this index is positive information is returned about the corresponding file or directory. If the index is -1 then information about the current directory is returned.

If the index is zero then information about an explicitly named file is returned. If the *string* parameter is present then this is used for the name of the file. If this parameter is absent then the name must have been previously assigned to *var*. This name may also contain path information (full or partial).

After execution, if the file or directory was found, the current file variable is set with the information. If *var* was defined then the general variable *var* contains the file name and the file information is copied to the file variable of the same name.

After executing this command the RENAME command can be used to change the name of the file or the CHANGE and SETINFO commands can be used to change information about the file.

FIND

syntax: FIND *var* , *string*
error code: set

This command can be used under System 7 to locate an application by specifying its signature in *string*. If the application is found the application name is assigned to the variable *var* and the default directory is changed to the directory containing the application. The error code is set if the application is not found.

FINDER

syntax: FINDER KEYWORD [, *var* [, *options*]]
error code: set

This command can be used under System 7 to control the Finder. It works by sending Apple Events to it. The following table lists the options for the command that require no other parameters.

Keyword	Description
SHOWABOUT	Shows the Finder's 'About This Macintosh' window.
HIDEABOUT	Hides the Finder's 'About This Macintosh' window.
SHOWCLIP	Shows the Clipboard.
HIDECLIP	Hides the Clipboard.
EMPTY	Empties the trash.
RESTART	Makes the Macintosh do a Restart.
SHUTDOWN	Makes the Macintosh do a Shutdown.
SLEEP	Makes a powerbook enter Sleep mode.

The following table lists those options that act on a list. The command keyword is followed by a variable name, *var*. The value in *var* is the number of items in the list. Each item in the list is an alias variable for a file or folder previously defined by the ALIAS command. These options are equivalent to selecting one or more items in an open folder and then selecting the corresponding choice from the Finder's File menu.

With these options the files and folders described by the list of aliases do not have to reside in the same folder.

Keyword	Description
OPEN	Equivalent to selecting Open or double clicking on the selection
DUPLICATE	Equivalent to selecting Duplicate
PRINT	Equivalent to selecting Print
ALIAS	Equivalent to selecting Make Alias
PUTAWAY	Equivalent to selecting Put Away
INFO	Equivalent to selecting Get Info
SHARING	Equivalent to selecting Sharing

The following table lists the options that are equivalent to selecting one or more files or folders and then dragging the selection to another folder. As in the previous case the variable *var* is a list of items to be moved. The destination of this move is specified by defining the alias variable which is the list name with index 0 to be the destination folder. In addition this form of the command takes an optional specification of an offset for the move in the form

FINDER MOVE, *var*, VER=*num*,HOR=*num*

If this option is included the position of the selection in the destination window will be offset from the position in the original window. The position will be shifted down by the amount specified by the VER keyword and right by the amount specified by the HOR keyword.

Keyword	Description
MOVE	Equivalent to dragging the selection and dropping into another folder (which must be on the same volume for this option).
DRAG	Equivalent to dragging the selection with the Option key down and dropping into another folder. The selection is duplicated in the destination folder.

The following table lists those options that act on a single item. The variable *var* is an alias for the item, which should be a folder (except for the CLOSEINFO case when either a file or folder alias can be specified).

Keyword	Description
CLOSE	This closes the window for the specified folder.
CLOSEINFO	This closes the 'Get Info' window for the specified file or folder.
CLOSESHARING	This closes the 'Sharing' window for the specified folder.
PRINTWINDOW	This is equivalent to selecting Print Window from the File menu.
PAGESETUP	This is equivalent to selecting Page Setup from the File menu.
VIEW	This is equivalent to using the View menu. The view is chosen by specifying one of the keywords SMALL, ICON, NAME, DATE, SIZE, KIND, COMMENT, LABEL, VERSION
MOVEWINDOW	This is equivalent to dragging the window to a new location. The keywords TOP= <i>num</i> and LEFT= <i>num</i> must be included.
SIZEWINDOW	This changes the size of the window. The keywords WIDTH= <i>num</i> and HEIGHT= <i>num</i> must be included.
ZOOMIN	This zooms the window in.
ZOOMOUT	This zooms the window out.
SHOW	This opens the window (if not already open) and brings it to the front.

FIXLIST

syntax: FIXLIST *list*
error code: unchanged

If a list has been manipulated with the SELECT and DESELECT commands then this command should be called to fix the internal list pointers so that the LISTNEXT command will work properly. It is not necessary to call this command if the LIST command is used on the list first.

FRONT

syntax: FRONT [keyword = *string*]
error code: set

Under System 7 this command brings an application to the foreground if it is in the background. An error of value 1 is returned if this command is executed under System 6. If the keyword is omitted then OmniScript is brought to the foreground. If the keyword is 'sign' then the expression *string* is the 4 character signature of the application to be brought to the foreground. If the keyword is 'file' then the expression *string* is the name of the application to be brought to the foreground.

GETFILE

syntax: GETFILE *var* [, *string*]
error code: set

This command calls the Standard File Package to obtain the name of an input file. The user will be prompted with a list of files from the current default directory. The string expression *string* is a concatenation of the types of the files to be shown in the dialog, eg. 'APPL' for an application or 'TEXT' for a text file.

If the user selects a file the name will be returned in *var* and the file variable of the same name will contain the file information. If the user changes the directory, the default directory will be set to the directory from which the user selected the file. If the user cancels file selection then the error code is set to 1.

GOTO

syntax: GOTO *label*
error code: unchanged

This command causes a jump in the script to the line with label given by the field *label*.

IF

syntax: IF *expression*
 commands executed if *expression* is non-zero
 [ELSE IF *expression1*
 commands executed if *expression1* is non-zero
 and *expression* is zero]
 [ELSE
 commands executed if all above *expressions*
 are zero]
 END
error code: unchanged

The syntax of the IF/ELSE/END is illustrated above. An arbitrary number of ELSE IF commands can be included. An error results if *expression* is non-numeric. Other IF/ELSE/END blocks and DO/END blocks can be nested within an IF/ELSE/END.

INCREMENT

syntax: INCREMENT *var*
error code: unchanged

If *var* exists and is numeric its value is incremented by one otherwise *var* is initialized to one.

KILL

syntax: KILL
error code: unchanged

This command immediately terminates the OmniScript application. One use is to avoid the

pause at the end of the last script if this is not wanted.

LAUNCH

syntax: LAUNCH *keyword1* = *application* [, *keyword2*]
error code: set

This command is used to launch an application. If the application is already open control is transferred to it. This command can be used under System 6, but OmniScript will be terminated. Under System 7 OmniScript continues. The application can be specified in three different ways as described in the following table.

Value of <i>keyword1</i>	Value of <i>application</i>
FILE	The value is a string expression that is the full or partial path name of the application or an alias of the application. If a partial path name is used it is relative to the current default directory.
ALIAS	The value is the name of a previously defined alias variable that points to the application or an alias of the application. This keyword is only valid under System 7.
SIGN	The value is a string expression that is the 4 character signature of the application. The desktop database is searched for the most recent version of the application. This keyword is only valid under System 7.

The other keywords are given in the following table. They are only meaningful when the command is used under System 7. The user should select one of the first three and optionally the fourth.

Value of <i>keyword2</i>	Meaning
FRONT	The launched application is in the foreground. This is the default
BACK	The launched application is in the background.
AEVT	An apple event is to be sent to the application when it is launched. The event is set up using the appropriate commands (eg. AENEW and AEADD) but the event is not sent with AESEND but is sent at launch time. The launched application is in the foreground.
WAIT	If this keyword is included then OmniScript waits until the launched application terminates. This wait can be terminated by selecting the Terminate option from the file menu, which will cause the script to continue, or by selecting the Exit or Restart options.

LIST

syntax: LIST *var* [, [*string*] [, *num*]]
error code: set

The LIST command uses the Macintosh List Manager to display a list. It enables the user to define a list, display it and select items from it. It works by using subscripted variables. Suppose the *var* parameter is \$LIST. Prior to calling the LIST command the user must generate the list. The variable \$LIST is set to the number of items in the list. \$LIST[1] contains the first list item, \$LIST[15] contains the fifteenth and so on.

When the LIST command is executed a window containing the list is displayed. If the optional *string* is included the title of the list window will be set to the value of *string*. This can be used to give a short description of the type of list selection required.

The user may select a list item by clicking on it with the mouse. Selection of more than one item depends on the value of *num*, which defaults to 0. With this default value the standard Macintosh rules are followed. More than one item may be selected by using the shift or command keys in combination with the mouse. Generally one selects several items by clicking on an item while holding down the command key. An item already selected is deselected by clicking on it while holding down the command key. Dragging while holding down the command key extends selection or deselection to the items dragged over. Using the shift key with the mouse selects a range of items. Different uses of the shift and command keys can be specified by the value of *num*. This is actually the value of the *selfFlags* byte of the *ListRec* structure defining the list (See *Inside Macintosh*, Vol. IV, page 267). The most likely use is to assign a value of 128 so that only one item may be selected. If *string* is omitted but *num* is included both commas must be specified.

The list display is terminated by selecting either the OK button (the normal exit) or the Cancel button, which generates an error code of 1. Double clicking on an item is equivalent to selecting OK. If a list item is checked using the *#SEL* function a value of 1 will be returned if it was selected or 0 if it was not. If the user only needs to process the selected items then the *LISTNEXT* command can be used. The number of items selected is stored in the item with index -1, eg. *\$LIST[-1]*.

When the list is displayed none of the items is shown as selected. Use the *LISTP* command if some of the items should be shown as already selected.

LISTDIR

syntax: LISTDIR *var* [, *types* [, *creators*]]
error code: unchanged

This command generates a list to be used by the *LIST* command from the names of files and directories in the current default directory. The parameter *var* is the name of the list. The files and directories to be included in the list can be filtered by specifying the file type and creator. The optional string expression *types* is a concatenation of the 4 character types of the files to be included in the list. If *types* is omitted all names in the directory will be included in the list, otherwise the list will contain a file only if its type was included in *types* and directories will only be included if the string 'DIR' is in *types*. The string expression *creators* is a concatenation of the 4 character creators of the files to be included in the list. (For example 'MSWD' would restrict the list to files created by Microsoft Word.) Note if *types* is omitted and *creators* is included both commas must be present with no field between them.

LISTNEXT

syntax: LISTNEXT *var1* ,*var2*
error code: unchanged

This command is used to obtain the selected items from a list. The parameter *var1* is the name of the list. On input *var2* should be a numeric variable with a value less than the number of items in the list. The command returns in *var2* the number of the next selected item that is greater than the input value of *var2* or zero if there are no more selected items in the list. Thus if *var2* is initialized to zero and the LISTNEXT command repeatedly called until a zero is returned the user can obtain the item number of each selected item.

LISTP

syntax: LISTP *var* [, [*string*] [, *num*]]
error code: set

This command is the same as LIST except that when the list is displayed those items in the list that have previously been selected (either by previously displaying the list or using the SELECT command) are shown as already selected.

LOADSCRIPT

syntax: LOADSCRIPT *script*

This command loads the script with name given by the field *script*. It uses the same search rule as the EXEC command.

LVALUES

syntax: LVALUES *var1* , *var2*
error code: set

This command takes the string value of the variable *var1* and breaks it into its component fields. A list with name *var2* is generated and the component fields are assigned to each list item in turn. The component fields are separated by spaces. A field should be enclosed in quotes if it contains spaces. The variables in the list will be set to be string, integer or floating point as appropriate. For example the following commands

```
SET a = "ABC DEF 'GHI JKL' 45"  
LVALUES a b
```

would generate a list of four items. The variable b would have value 4, b[1] would be ABC, b[2] would be DEF, b[3] would be GHI JKL and b[4] would be an integer with value 45.

MENU

syntax: MENU NEW , *menuid* , *position* , *list*
 MENU DELETE , *menuid*
error code: unchanged

This command is used to create a menu specified by a general variable list. The number *menuid* is the menuID as described in ‘Inside Macintosh’ and should be in the range 4 to 235. This value is used in the MENUSCRIPT and MENUSET commands, the #MENU function and for defining pop-up menus in dialogs. The number *position* corresponds to the beforeID parameter of the InsertMenu script described in ‘Inside Macintosh’. A value of -1 indicates the menu is to be used as a submenu or pop-up menu and does not appear in the menu bar. A value of 0 places the menu after all others on the menu bar and a positive value places it after the menu with menuID *position* . The name of the menu is derived from the list name *list* by deleting the first character. (Deleting this first character allows the use of list names starting with @,\$, or \ without having this character appear in the menu name.

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

The menu may be deleted by the second form of the command.

When an item is selected from a menu created by this command the item number is saved and can be checked by the #MENU function. Optionally the MENUSCRIPT command can specify a script to be executed.

MENUSCRIPT

syntax:	MENUSCRIPT <i>menuid</i> , <i>script</i>
error code:	unchanged

This command specifies a script to be executed when an item is selected from the menu specified by *menuid* . The current script is interrupted to execute the script named in the field *script* . Two arguments are passed to this script. The first is the number *menuid* and the second is the number of the selected item.

MENUSET

syntax: MENUSET *menuid* , *num*
error code: unchanged

The #MENU function returns the item number of the item last selected from a menu created by the MENU command or 0 if no item has been selected. This value can be changed by the MENUSET command. One use is to initialize the display of a pop-up menu. The number *menuid* specifies the menu to be set and *num* specifies the value.

MODELESS

syntax: MODELESS NEW , *id* , *script* , *list*
 MODELESS SHOW/DELETE , *id*
error code: set

This command is used to set up modeless dialogs. The dialog is stored as a resource with id number *id* in a file that must have previously been opened. The NEW keyword is used to setup the dialog. The field *script* is the name of the script to be executed when a button is pressed in the dialog. The variable *list* is the name of a list that describes the dialog. This is discussed above in the section on dialogs. Once a dialog has been set up the command is used with the SHOW keyword to show it (if the dialog was defined so that it is invisible when created) and the DELETE keyword to delete it.

MOVE

syntax: MOVE [*string*] , *var*
error code: set

This command moves the file or folder named *string* (full or partial path name relative to the current default directory) to the directory defined by the path variable *var* (set by a PATH or SCRIPTPATH command). If *string* is omitted then the current default directory is moved.

NEWDIR

syntax: NEWDIR *directory*
 error code: set

This command creates a new directory with name given by the string expression *directory*, which can be either a full path name or a partial path name relative to the current default directory.

OPEN

syntax: OPEN *num* , [KEY1 , ... , KEYn]
 error code: set

This command is used to open a file and associate a unit number *num* with the file for use by the other file commands.

The keywords are

FILE = <i>name</i>	The string expression <i>name</i> is the name of the file to be opened. Aliases will be resolved under System 7.
BOTH READ WRITE	This decides whether the file is to be opened for reading, writing or both. BOTH is the default.
NEW OLD UNKNOWN	This decides whether to use an existing file (OLD), to create a new file (NEW) or to create a new file only if one does not already exist (UNKNOWN), which is the default. If new is specified and the file already exists then the data fork is truncated to zero length instead of actually creating a new file. This leaves the resource fork and Finder information unchanged.
TYPE = <i>string</i>	This specifies the type and creator of the file (even if it already exists). For example TYPE = TEXTEDIT defines a TEXT document created by the application with signature EDIT.
LABEL = <i>num</i>	This sets the file label (color under System 6). 7 is the highest priority (first priority in the Label menu or first color in the Color menu) 0 is the lowest priority or last color
APPEND	Writing will be at the end of the file instead of the beginning for an existing file.
ACCESS = <i>string</i>	As an alternative to using the above keywords to specify the file access mode this keyword can be used with the corresponding ANSI

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

	C code for file access. See the table below.
--	--

The following table gives the equivalents of command keywords and ANSI C codes for file access.

Command Keywords	ANSI code
READ	r
WRITE	w
WRITE, APPEND	a
BOTH - File does not exist	w+
BOTH - File already exists	r+
BOTH, APPEND - File already exists	a+

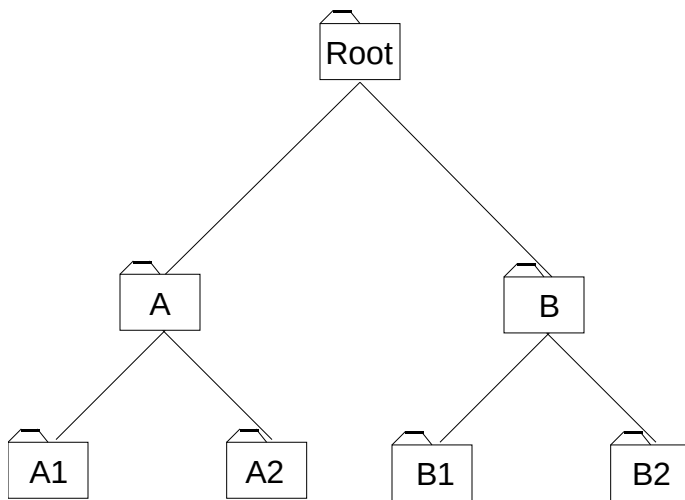
PATH

syntax: PATH [*var1* =] [*var2*], [*string*]
error code: set

This command is used to set the current default directory and to define path variables. If *var1* is included then this path variable is set by the command. If *var2* is included it is a path variable generated by an earlier PATH command or it is one of the default path variables, (see the section on Path Variables).

If the expression *string* is a full path name (that is it starts with a volume name and contains at least one colon but does not start with a colon) then the PATH command sets the default directory to this path. If *string* is a partial path name (that is it begins with a colon or contains no colons) then the PATH command generates a path relative to the directory defined by the path variable *var2*. If *var2* is omitted then the current default directory is used.

The following examples applied to the illustrative directory hierarchy should clarify the PATH command. It is assumed that 'Root' is the volume name.



PATH X = 'Root:A'	The default directory is now A.
PATH Y = X,'A1'	The default directory is now A1.
PATH , '::A2'	The default directory is now A2. The double colon causes a move one up the hierarchy from the current default directory, A1.
PATH , ':::B:B1'	The default directory is now B1.
PATH W = ,	This is valid. It gives the name W to the current default directory. This would be useful after a call to PUTFILE or GETFILE which can change the default directory.
PATH W = , ':'	This is also valid and is the same as above.
PATH R = , 'Root'	This is invalid. Root is a partial path name and cannot refer to the Volume.
PATH R = , 'Root:'	This is valid. A full path name has been given. R refers to the Volume Root.
PATH R,'B:B1'	This is invalid. B:B1 is an full path name and B is not a volume name.

PATH R,':B:B1'	This is valid. The default directory is B1.
----------------	---

PAUSE

syntax: PAUSE *string*
error code: unchanged

This command displays *string* and waits for the user to press the enter key or select continue from the File menu.

PUTFILE

syntax: PUTFILE *var* [, [*default*] [, *prompt*]]
error code: set

This command calls the Standard File Package to obtain the name of an output file. The user will be prompted with the current default directory. The string *default* , which may be null, is the default name, and the string *prompt* is the string used to prompt the user.

If the user selects a file the name will be returned in *var*. If the user changes the directory, the default directory will be set to the directory from which the user selected the file. If the user cancels file selection then the error code is set to 1.

QUIT

syntax: QUIT
error code: unchanged

This command is used to immediately terminate a DO loop with no further iterations. This would often be used with the TEST command for a conditional quit.

READ

syntax: READ *num* , *var*
error code: set

This command reads a record from the file previously opened with unit number *num* and saves it in the variable *var*. The error code is set to 1 if the file was not previously opened. Reading is terminated by a return character in the input file.

REMOVE

syntax: REMOVE *var* , *num*
error code: unchanged

This command is used to remove item number *num* from the list *var*. (Actually item number *num* is moved to the end of the list and the number of items in the list is decremented.)

RENAME

syntax: RENAME *field*
error code: set

This command can be used to rename a file or directory. It must be used after a FILENAME command and renames the file or directory selected by the last FILENAME command. The new name is *field*. The error code is set to 1 if there is no file to rename (because FILENAME was not previously called or the internal file information was overwritten by other commands).

REORDER

syntax: REORDER *var* , *num1* , *num2*
error code: unchanged

This command moves an item in the list *var*. Item number *num1* becomes item number *num2* and the items between positions *num1* and *num2* are shifted appropriately. If *num2* exceeds the number of items in the list then item *num1* is moved to the end of the list.

REPORT

syntax: REPORT
error code: unchanged

This command displays information about the last command to set the error code if it was set non-zero. The value of the return code, the command and the name of the script are displayed after a line saying '**** ERROR REPORT ****'.

REWIND

syntax: REWIND *num*
error code: set

This command rewinds the file previously opened with unit number *num*. The error code is set to 1 if the file was not previously opened.

RESOLVE

syntax: RESOLVE *var*
error code: set

This command is used to retrieve information from an alias variable created by the ALIAS command. If the alias variable *var* refers to a file then the current default directory is set to the directory containing this file and the general variable *var* is set to the name of the file. If the file does not exist, but the path information is valid the error code is set to -43, but this is not treated as a non-fatal error. If the alias variable *var* refers to a directory then the current default directory is set to this directory and the general variable *var* is set null (ie. a string of length 0).

RESTART

syntax: RESTART
error code: unchanged

This System 7 only command sends a Restart event to the Finder (class = 'FNDR', ID = 'rest'). This is equivalent to selecting 'Restart' from the Special Menu. Although the Apple Event commands can also be used to do this, this command avoids the possibility of OmniScript responding to the Finder's quit event with a dialog requesting the user to cancel the current script.

RFCLOSE

syntax: RFCLOSE *num1* [, *num2* , *num3* ...]
error code: set

This command is used to close one or more files that were opened by the RFOPEN command. *numx* is the path reference number of the file, which can be saved by the RFOPEN command. The error code is the status returned by the last file closed.

RFOPEN

syntax: RFOPEN *string* [, *var*]
error code: set

This command is used to open a resource file. One use is prior to the DIALOG command to open the resource file containing the dialog template. The string expression *string* is the file name. The file must be in the current default directory. If the optional *var* is present the file reference number will be stored in this variable for possible use by the RFCLOSE command.

SAVE

syntax: SAVE [*file* , [APPEND/NEW , [*type*]]]
error code: unchanged

This command is used to save the screen display to a file. When this command is executed the display is deleted and saved to the file. The display is also automatically saved when 'OmniScript' terminates. The command may be called more than once, either to specifically save the display at that point or to change the file to which the display is saved.

If no parameters are specified then the file is saved to the file specified on a previous execution of SAVE. If the first execution of SAVE did not specify a file the display is saved to a file with a name in the format 'Saved on mm/dd/yy at hh/mm/ss' located in the root folder of the start up volume.

If the file is specified then the keyword APPEND may be specified to add the display to a previously created file. The default NEW creates a new file or erases the contents of an existing file. The type and creator of the file can be set or changed if the string *type* is specified. This must be an 8 character string. The first four characters are the file type and the second four are the creator.

SCRIPT

syntax: SCRIPT *field*

This is not a command but a directive to the pre-processor when a Script is initially being read in. Pre-processing of the current script is terminated and a new Script with name *field* is started.

SCRIPTPATH

syntax: SCRIPTPATH [*var1* =] [*var2*], [*string*]
error code: set

This command is used to define the directory searched when the EXEC command invokes a script not previously called. The use of this command is identical to the PATH command except that it does not change the current default directory.

SELECT

syntax: SELECT *var1* [, *var2* , *var3* ...]
error code: unchanged

This command is used to mark a list item as selected. See the section ‘Lists’.

SET

syntax: [SET] *var* = [*expression*]
error code: unchanged

This command assigns a value to the general variable *var*. The value of *expression* can be string, integer or floating point and *var* is flagged correspondingly. If *expression* is omitted then *var* is set null, that is non-numeric with a string of length zero. The command name SET can be omitted if there is no conflict between the variable name and a command name.

It is also possible to set a sub-range of a variable by specifying a range with the variable name. In this case *expression* should be either string, integer or null. If the length of expression is less than the length of the subrange then the string will be left justified and remainder of the subrange will be space filled. (If > is used instead of = the string will be right justified. If the length of expression is greater than the length of the subrange then expression will be truncated. It is left justified and truncated on the right unless > is used in which case it is right justified and truncated on the left.

SETINFO

syntax: SETINFO [*var*]
error code: set

This command is used to change information about a file or directory following a CHANGE command. If *var* is absent the current file is used otherwise the file defined by the file variable *var* is used (and this becomes the current file). The error code is set to 1 if there is no file to

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

rename (because FILE was not previously called or the current file information was overwritten by other commands).

SORT

syntax: SORT *list1* [, *list2* , *list3* ...]
error code: set

This command sorts one or more lists. The list *list1* is sorted in ascending order by ASCII code (except that lower case is converted to upper case before doing the comparison). If more than one list is specified in the command the members of the subsequent lists will be reordered in the same way as the first list, thus allowing the sorting of several associated lists using the first list as the key.

If a list has an associated list of file variables then these will also be reordered if the list name is followed by /f or /F. A list of alias variables can be reordered by adding /a or /A to the list name. (This cannot be done for the first list because there is no associated list of names to sort by.)

The error code is set to 1 if the value of the variable *list1* was not set to indicate the number of items in the list. The error code is set to 2 if any of the items in *list1* is undefined. The error code is set to 3 for an invalid character following the / following the list name.

SORTD

syntax: SORTD *list1* [, *list2* , *list3* ...]
error code: set

This is the same as sort except that the sorting is done in descending order.

SOUND

syntax: SOUND *expression*
error code: unchanged

This command plays the sound from a ‘snd’ resource. If the *expression* parameter has an integer value the program will look for a ‘snd’ resource with this resource number. If it can’t find it or the *expression* parameter is not an integer the program will treat field as the name of a ‘snd’ resource. If the program still fails to find a resource it will treat *expression* as the name of a file (either a partial path name relative to the current default directory or a full path name) and plays the first ‘snd’ resource found in this file.

If a ‘snd’ resource can not be found as specified the System Beep sound is played twice. The simplest use is to play a sound in the Sytem File by name, for example

SOUND ‘Clink-Klank’

SRAND

syntax: SRAND *num*
error code: unchanged

This command sets the seed for the sequence of random numbers generated by the #RAND function.

TAILOR

syntax: TAILOR *num*
error code: unchanged

This command defines the unit number of the file that will be used for the output of file tailoring. The error code is set to one if the file was not previously opened.

TEST

syntax: TEST (*expression*) *command*
error code: depends on COMMAND

This command evaluates *expression*. If the result is non-zero then *command* is executed. *command* is any valid command except for the control commands DO, IF, ELSE and END.

UNMOUNT

syntax: UNMOUNT *string*
error code: set

This command unmounts the volume named in the expression *string*.

VALUES

syntax: VALUES *source* , *var1* [, *var2* ...]
error code: unchanged

This command is similar to LVALUES except that when the source string, contained in the variable *source*, is broken into its component fields they are assigned in turn to the variable *var1*, *var2* etc. Decoding stops when all the variables in the list have been assigned values. If there are fewer fields than variables then the remaining variables will be assigned null values.

VLOAD

syntax: VLOAD *file* [, *num*]
error code: set

This command loads the variables from the file named in the string expression *file* to one of the eleven blocks of global variables, depending on the value of *num*.. All previously defined global variables in the block are erased. If there is an error reading the file the error code is set. See the section ‘Saving Global Variables’ for further explanation.

VSAVE

syntax: VSAVE *string* [, *num*]
error code: set

This command saves the variables from one of the eleven blocks of global variables to the file named in the string expression *file*. This file can be subsequently used in a VLOAD command.

Copyright © 1991-1993, Richard G. Gibbs. All rights reserved.

If there is an error writing to the file the error code is set. See the section ‘Saving Global Variables’ for further explanation.

WAIT

syntax:	WAIT [NONE] <i>expression</i>
or:	WAIT [NONE] UNTIL hour:min[: sec] [ON month/day]
error code:	unchanged

In the first format this command suspends the execution of the script for the number of seconds specified by *expression*.

In the second format the command suspends execution until the specified time. The seconds part of the time (which must be specified in 24 hour format) is optional. The date is optional and defaults to the current date. If the date is specified and the month is less than the current month then the date is assumed to be in the next year. Each numerical part of the time or date is an expression. If the specified time has already passed the program does not wait. No error is generated.

This command could be used when running under MultiFinder or System 7 to delay execution of the application to enable some other application to finish. This would be used when the Macintosh is left unattended. During the wait the status message specifies when the wait will terminate unless the NONE keyword was specified. During the wait period the program will respond to menu selection and mouse events or other scripts executed at the interrupt level.

WINDOW

syntax: `WINDOW top , left , bottom , right`
error code: unchanged

This command changes the location and size of the display window. The window's rectangle is reset to the specified values.

WRITE

syntax: `WRITE num , string`
error code: set

This command is used to write text to a previously opened file. The expression *string* is written to the file with unit number *num*. A null string causes a blank line to be written. The error code is set to 1 if the file was not previously opened.

Resources

OmniScript contains a number of resources, some of which the user may like to modify (probably by using either ResEdit or RMaker). The following table lists these resources and their purposes.

type	id	purpose
dctb	140	Allows use of default colors in dialogs with same ids
DITL	140	Dialog template for directory function
DLOG	140	Directory function dialog
EPFI	129	Font information for displays. This resource consists of 9 words: respectively the font number, face and size for the status message, the display window text and the list window.
SFLN	129	Default location of standard file package dialogs. The resource consists of two long words (treated as points). The points are respectively the location of the top left hand corner of the following dialogs 1 Get File or Directory 2 Put File
WIND	129	Main display window
WIND	131	List selection window

The SFLN resource is omitted from the OmniScript supplied. The default is to center these dialogs on the screen.

The folder ‘Change Resources’ describes how the application RMaker can be used to easily change the EPFI resource and add the SFLN resource.