MPW: Macintosh Programmer's Workshop Development Environment, Volume 2

MPW: Macintosh Programmer's Workshop Development Environment, Volume 2

Version 3.0

Part II **Command Reference**

Part II **Command Reference**

Part II is a command dictionary that describes each of the tools, scripts, and built-in commands of the Macintosh Programmer's Workshop 3.0. When you have become sufficiently familiar with the material in Part I, you can move Part II to a smaller separate binder for convenient desktop reference. (You may also want to include frequently used appendixes or tables in the separate binder.) Please be sure to read the next section, "Command Prototype," which explains the format for all command descriptions and defines the basic behavior of all commands. ■

Contents

Command prototype

The following command prototype illustrates the conventions that we've used to describe MPW commands. Most commands behave roughly as specified at the end of the introduction.

Syntax

Command [*option...*] [*file...*]

- *Note:* Filenames, command names, and options are not sensitive to case. The syntax notation itself is described at the end of the introduction.

Description

The first word of the command is the filename of the program to execute or the name of a built-in command. The subsequent words are passed as additional parameters to the command (or recognized by the Shell in the case of I/O redirection).

Most commands recognize two distinct types of parameters: options and filenames. Options begin with a hyphen (-) to distinguish them from filenames. Although the syntax descriptions list the options first, options and files may appear in any order. All options apply to the processing of all the files, regardless of the ordering of options and files.

For commands that read and write text files, you can specify a file, a window, or a selection within a window, as follows:

name	Named window or file
§	The selection in the target window (the second window from the top)
name.§	The selection in the named window

Type

Commands may fall into one of three categories: Tool, Script, or Built-In. This information is useful when you need to figure out why a command isn't working. For example, if you know that the command is a tool or a script, you can deduce that the file might be missing or that there might be a file of the same name in the current directory.

Input

Standard input is often processed if no filenames are specified.

- *Note:* If a program is reading from standard input, you can press Command-Enter (or Command-Shift-Return) to indicate EOF and terminate input. (See "Terminating a Command" in Chapter 4.)

Output

Text processors usually write their output to standard output. The MPW Assembler writes listings to standard output. Link, the MPW linker writes location maps to standard output.

Diagnostics

Errors and warnings are written to diagnostic output. If no errors or warnings are detected, most commands don't write anything to diagnostic output. Assembler and Compiler error messages have the format

message
`File "`*filename*`" ; Line` *linenumber*

This format makes it possible to select and execute the text after "###" because the names "File" and "Line" have been defined as Shell commands—"File" is defined in the Startup file as an alias for the Target command, and "Line" is a short command file that finds a line number.

Several commands write progress and summary information to diagnostic output if you specify the **-p** option.

Status

Status codes are returned in the {Status} variable. A value of 0 indicates that no errors occurred; anything else usually indicates an error. Typical values are

0 Command succeeded.
1 Incorrect options or parameters.
2 Command failed; invalid input.

Positive numbers are returned by tools, scripts, and built-in commands. Negative numbers are returned only by the Shell.

Options

Options specify some variation from the default command behavior. Options begin with a hyphen (-) to distinguish them from files and other parameters.

Options form single words in the command language. Some options require additional parameters, which are separated from the option name with a blank. (An option's parameters also form a single word in the command language.) If more than one option parameter is required, the usual separators between them are commas and equal signs. For example,

```
Asm  -define &debug='on'  -pagesize 84,110  ...
```

Note that spaces are *not* allowed between option parameters and their separating commas. For those options that do have additional parameters, the option parameters are never optional.

Options may appear in any order. *All* options are collected prior to processing files.

Limitations

A few commands may have special cases or warnings that you should know about. Be sure to check for a Limitations heading at the end of the command's reference.

See also

"Structure of a Command" in Chapter 5.

AddMenu—add menu item

Syntax AddMenu [menuName [itemName [command...]]]

Description Associates a list of commands with the menu item itemName in the menu menuName. If the menu menuName already exists, the new item is appended to the bottom of that menu. If the menu menuName doesn't already exist, a new menu is appended to the menu bar, and the new item is appended to that menu. When the new menu item is selected, its associated command list is executed just as though the command text had been selected and executed in the active window.

- *Note:* The command text that you specify for an AddMenu item is processed twice—once when you execute the AddMenu command itself, and again whenever you subsequently select the new menu item. This means that you must be careful to quote items so that they are processed at the proper time. See the "Examples" section below.

You can also use AddMenu to display information for existing user-defined menus by omitting parameters:

- If command is not specified, the command list associated with itemName is written to standard output.
- If itemName and command are both omitted, a list of all user-defined items for menuName is written to standard output.
- If no parameters are specified, a list of all user-defined items is written to standard output.

(This output is in the form of AddMenu commands.)

You can also use AddMenu to change the command list or markings associated with a particular itemName. If both menuName and itemName already exist, the command list associated with itemName will be changed to command. Also, any marking or styles associated with itemName will be changed. The position of itemName in menuName will not be affected.

You can define keyboard equivalents, character styles, and other features for your new menu commands—itemName can contain any of the metacharacters that are used with the AppendMenu( ) procedure documented in the chapter entitled "Menu Manager" of *Inside Macintosh:*

/char	Assign the keyboard equivalent Command-char.
!char	Place char to the left of the menu item.
^n	Item has an icon, where n is the icon number. See *Inside Macintosh.*
(	Item is disabled (dimmed).
<style	Item has a special character style; this style can be any of the following capital letters:

B	Bold
I	Italic
U	Underline
O	Outline
S	Shadow

Multiple styles may be specified by preceding each with "<". Be sure to quote menu items containing these special characters. (See the "Examples" section below.)

◆ *Note:* Semicolons (;) cannot be used within an itemName.

Menu items can't be appended to the Window, Mark, or Apple menus.

Type Built-in.

Input None.

Output If any of the optional parameters is omitted, a list of user-defined menu items and their associated commands is written to standard output.

Diagnostics Errors and warnings are written to diagnostic output.

Status AddMenu may return the following status codes:

0	No errors.
1	Syntax error.
2	An item can't be redefined.
3	System error.

Options None.

Examples

```
AddMenu
```

Lists all user-defined menu items.

```
AddMenu Extras "TimeStamp/P"  'Echo `Date`'
```

Adds an "Extras" menu with a "TimeStamp" item, which writes the current time and date to the active window. This item has the Command-key equivalent Command-P.

```
AddMenu File 'Format<B' 'Erase 1'
```

Adds a "Format" item to the File menu (as discussed under the Erase command) and makes the item bold.

```
AddMenu Find Top 'Find • "{Active}"'
```

Adds the menu item "Top" to the Find menu, and defines it as the Find command enclosed in single quotation marks. This command places the insertion point at the beginning of the active window.

Note: The following attempt to do the same thing will not work:

```
AddMenu Find Top "Find • {Active}"
```

This command won't work because the {Active} variable will be expanded when the menu is added. (It should be expanded when the menu item is executed.) In the first (correct) example, the single quotes defeat variable expansion when the AddMenu command is executed; they are then stripped before the item is actually added. The double quotation marks remain, in case the pathname of the active window happens to contain any special characters.

You may want to add some or all of the following commands to your UserStartup file:

```
AddMenu Find '(-'                '
AddMenu Find 'Top/6'             'Find • "{Active}"'
AddMenu Find 'Bottom/5'          'Find ∞ "{Active}"'
```

These commands create several new items in the Find menu. The first is a disabled separator that creates a new section at the bottom of the menu. The Top and Bottom items position the insertion point at the top and bottom of the active window. Both menu items have Command-key equivalents.

```
AddMenu Directory 'Work'     'Directory HD:MPW:Work'
AddMenu Directory 'Work!•'   'Directory HD:MPW:Work'
```

The first command creates a command to move to the directory HD:MPW:Work. The second command marks the Work item with a bullet without changing the position of the item in the menu.

See also DeleteMenu command.

"Quoting Special Characters," "How Commands Are Interpreted," and "Defining Your Own Menu Commands" in Chapter 5.

"Creating a Menu in Your Program" in chapter "Menu Manager" of *Inside Macintosh.*

Adjust—adjust lines

Syntax Adjust [-c *count*] [-l *spaces*] *selection* [*window*]

Description Finds and selects the given selection and shifts all lines within the selection to the right by one tab, without changing the indentation.

If a count is specified, *count* instances of *selection* are affected. The **-l** option lets you move lines by any number of spaces to the left or right.

If you specify the *window* parameter, the command operates on *window*. It's an error to specify a window that doesn't exist. If no window is specified, the command operates on the target window (the second window from the front).

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status Adjust may return the following status codes:

0 At least one instance of the selection was found.
1 Syntax error.
2 Another error.

Options

-c *count* Repeat the select-and-adjust operation *count* times.

-l *spaces* Every line within the selection will be shifted *spaces* to the right. You can shift a selection left by specifying a negative value for *spaces*.

Examples

```
Adjust -l 4 §
```

Shifts the lines containing the target selection to the right by four spaces.

```
Adjust -l -8 /if/Δ:Δ/else/
```

Selects everything after the next "if" and before the following "else", and shifts all lines within the selection to the left by eight spaces.

See also Align command.
"Selections" in Chapter 6.

Alert—display an alert box

Syntax Alert [-s] [*message...*]

Description Displays an alert box containing the prompt *message*. The alert is displayed until its OK button is clicked. If the message contains any special characters, you'll need to quote it, as explained in Chapter 5.

Type Built-in.

Input Reads standard input for the message if no parameters are specified.

Output None.

Diagnostics None.

Status Alert may return the following status codes:

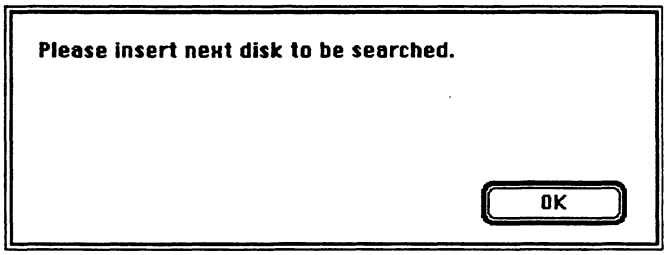
0 No errors.
1 Syntax error.

Options **-s** Run silently. Do not beep when the dialog box is displayed.

Example

```
Alert Please insert next disk to be searched.
```

Displays the following alert box and waits for the user to click "OK" before returning.



See also Confirm and Request commands.

Alias—define or write command aliases

Syntax

Alias [*name* [*word...*]]

Description

Name becomes an alias for the list of words. Subsequently, when *name* is used as a command name, *word...* will be substituted in its place.

If only *name* is specified, any alias definition associated with *name* is written to standard output. If *name* and *word* are both omitted, a list of all aliases and their values is written to standard output. (This output is in the form of Alias commands.)

Aliases are local to the script in which they are defined. An initial list of aliases is inherited from the enclosing script. Inherited aliases may be overridden locally. You can make an alias definition available to all scripts by placing the definition in the UserStartup file.

You can remove aliases with the Unalias command.

Type

Built-in.

Input

None.

Output

When parameters are omitted, the Alias command writes aliases and their values to standard output.

Diagnostics

Errors are written to diagnostic output.

Status

Alias may return the following status codes:

0 No errors.
1 The specified alias could not be found.

Options

None.

Examples

```
Alias Dir Directory
```

Creates an alias "Dir" for the Directory command.

```
Alias Top 'Find •'
```

Creates an alias "Top" for the command "Find •" (which places the insertion point at the beginning of a window). The command takes an optional window parameter and by default acts on the target window. The Top command could now be used as follows:

```
Top             # find top of target window
Top Sample.a    # find top of window Sample.a
                # (equivalent to "Find • Sample.a")
```

See also

Unalias command.

"Command Aliases" in Chapter 5.

Align—align text to left margin

Syntax Align [-c *count*] *selection* [*window*]

Description All lines within each instance of the selection are positioned to the same distance from the left margin as the first line in the selection.

If you specify the *window* parameter, the Align command will act on *window*. It's an error to specify a window that doesn't exist. If no window is specified, the command operates on the target window (the second window from the front).

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status Align may return the following status codes:

0 At least one instance of the selection was found.
1 Syntax error.
2 Any other error.

Option **-c** *count* Repeat the select-and-align operation *count* times.

Examples

```
Align §
```

Same as the Align menu item; that is, it aligns all lines in the default selection with the first line of the selection.

```
Align /Begin/:/End/
```

Selects everything from the next "Begin" through the following "End", and aligns all lines within the selection to the same margin position as the line that contains the "Begin".

See also Adjust command.

"Selections" in Chapter 6.

Asm—MC68xxx Macro Assembler

Syntax Asm [*option*...] [*file*...]

Description Assembles the specified assembly-language source files. One or more filenames may be specified. If no filenames are specified, standard input is assembled and the file "a.o" is created. By convention, assembly-language source filenames end in the suffix ".a". Each file is assembled separately—assembling file *name*.a creates object file *name*.a.o. The object filename can be changed with the **-o** option.

See the *MPW 3.0 Assembler Reference* for more information about the assembly language. The first Commando dialog box for this command is reproduced here for convenience.

Type Tool.

Input If no filenames are specified, standard input is assembled. (You can terminate input by pressing Command-Enter, or you can enter an END directive, preceded by a blank space.)

Output If either the **-l** or the **-s** option is specified, an assembler listing is generated. If standard input is used for the source file, the listing is written to standard output. If the input is taken from file *name*.a, the listing is written to *name*.a.lst. The listing filename can be changed with the **-lo** option. The option **-lo** must be preceded by the **-l** option and must be immediately followed by the listing filename.

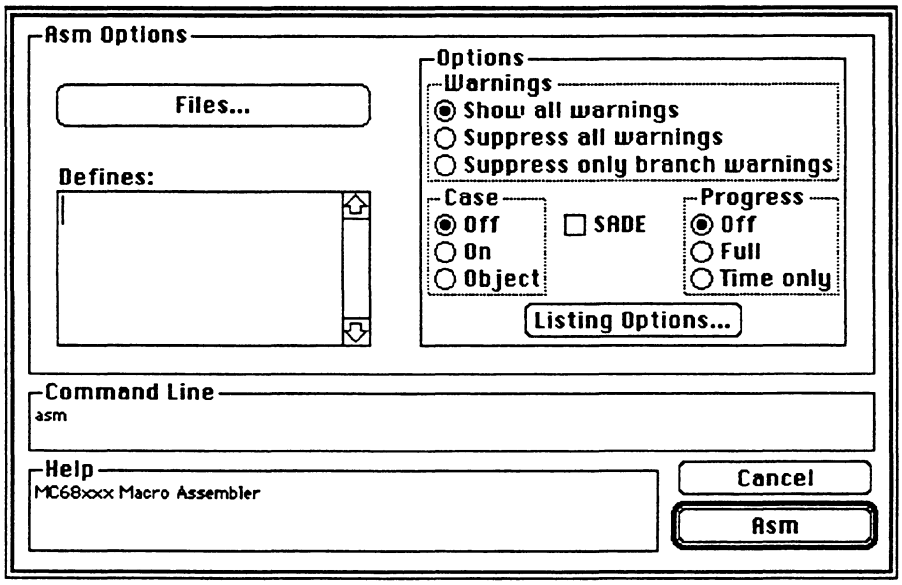
Diagnostics Errors and warnings are written to diagnostic output. If the **-p** option is specified, progress and summary information is also written to diagnostic output.

Status Asm may return the following status codes:

0 No errors detected in any of the files assembled.
1 Parameter or option errors.
2 Errors detected.

Options

Except for the **-case on** option, options may appear in any order.



-addrsize *size*

Set address displays in the listing to *size* digits (values 4 through 8 are allowed). The default is 5 digits.

-blksize *blocks*

Set the assembler's text file I/O buffer size to *blocks* 512 bytes. Values 6 through 62 are allowed. Odd values are made even by reducing the value by 1. The default value is 16 (8192 bytes) if the assembler determines it has the memory space for the I/O buffers, and 6 (3072 bytes) otherwise. This option permits optimization of I/O performance (transfer rate for text file input, load/dump files, and listing output) as a function of the disk device being used. Note that increasing the *blocks* value reduces the amount of memory available for other Assembler structures (such as symbol tables).

-case on

Distinguish between uppercase and lowercase letters in nonmacro names (same as CASE ON). (Case is always ignored in macro names.) If you intend to preserve the case of names declared by the **-define** option, the **-case on** option must *precede* the **-define** option(s) in the command line.

-case obj[ect]

Preserve the case of module, EXPORT, IMPORT, and ENTRY names *only in the generated object file.* In all other respects, case is ignored within the assembly, and the behavior is the same as the preset CASE OFF situation.

-case off Ignore the case of letters. All identifiers are case insensitive. This is the preset mode of the assembler, but it may be used in the command line to reverse the effect of one of the other **-case** modes.

-c[heck] Syntax check only. No object file is generated.

-d[efine] *name*[=*value*] [,*name*[=*value*]]

Define the name as having the specified value. The value is a decimal integer. If *value* is omitted, a value of 1 is assumed. This option is equivalent to placing the directive

name `EQU` *value*

at the beginning of your source file. To test whether the name is defined, use the function `&Type`. You can define more than one name by specifying multiple **-d** options or multiple *name*[=*value*] parameters separated by commas. For example,

```
Asm -d debug1,&debug="'on'" ...
```

-d[efine] *&name*[=[*value*]] [,*&name*[=[*value*]]]...

Define the macro *name* as having the specified value. The value is a decimal integer or a string constant. If the *=value* is omitted, the decimal value 1 is assumed. If only the *value* is omitted, the null string is assumed. **-define** is equivalent to declaring the name as a global arithmetic symbol (GBLA for an integer value) or global character macro symbol (GBLC for a string value) and placing one of the following directives at the beginning of the source file:

```
            GBLA  &name
&name       SETA  value
```

or

```
            GBLC  &name
&name       SETC  value
```

To test whether the name is defined, use the function `&Type`. You can define more than one macro name by specifying multiple **-d** options or multiple *&name*[=*value*] parameters separated by commas.

-e[rrlog] *filename*

Write all errors and warnings to the error log file with the specified filename (same as ERRLOG '*filename*').

Note: If only warnings are generated, no error file is created.

-f Suppress page ejects (same as PRINT NOPAGE).

-font [*fontname*] [,*fontsize*]

Set the listing font to *fontname* (for example, Courier), and the size to *fontsize*. This option is meaningful only if the **-s** or the **-l** option is used; you cannot omit both. The default listing font is Monaco 7. Note that listings are formatted correctly only if a monospaced font is used.

-h Suppress page headers (same as PRINT NOHDR).

-i *pathname*[,*pathname*]...

Search for include and load files in the specified directories. Multiple **-i** options may be specified. At most, 15 directories are searched. The search order is:

1. The include or load filename is used as specified. If a *full pathname* is given, no other searching is applied.

   If the file isn't found, and the pathname used to specify the file is a *partial pathname* (no colons in the name or a leading colon), the following directories are searched.

2. The directory containing the current input file.
3. The directories specified in **-i** options, in the order listed.
4. The directories specified in the Shell variable {AIncludes}.

-l Generate full listing. If file *name*.a is assembled, the listing is written to *name*.a.lst.

-lo *listingname*

Pathname for the listing file and directory for the listing scratch file. If *listingname* ends with a colon (:), it indicates a directory for the listing file, whose name is then formed by the normal rules (that is, *inputFilename*.a.lst). If *listingname* does not end with a colon, the listing file is written to the file *listingname*. In this case, listings for multiple source files are appended to the listing file. In either case, the directory implied by the listing name is used for the assembler's listing scratch file. The **-lo** option is meaningful only if the **-s** or the **-l** option is used.

-o *objname* Pathname for the generated object file. If *objname* ends with a colon (:), it indicates a directory for the output file, whose name is then formed by the normal rules (that is, inputFilename.o). If *objname* does not end with a colon, the object file is written to the file *objname*. (In this case, only one source file should be specified to the Aassembler.)

-pagesize [*l*] [,*w*]

Set the listing page size. (This option is meaningful only if the **-s** or **-l** option is specified; you cannot omit both.) The *l* and *w* parameters are integers: *l* is the page length (default = 75) and *w* is the page width (default = 126). (These settings assume that Monaco 7 is being used with the MPW Print command to the LaserWriter.)

-print *mode*[,*mode*]...

Set a print option mode. *Mode* may be any one of the following PRINT directive options:

`[NO]GEN`	Macro expansions
`[NO]PAGE`	Page ejects
`[NO]WARN`	Warnings
`[NO]MCALL`	Macro calls
`[NO]OBJ`	Object code
`[NO]DATA`	Data
`[NO]MDIR`	Macro directives
`[NO]HDR`	Page headings
`[NO]LITS`	Literals
`[NO]STAT`	Progress information
`[NO]SYM`	Symbol table display

See the *MPW 3.0 Assembler Reference* for a discussion of these PRINT settings. You can specify more than one print option by specifying multiple **-print** options or multiple *mode* parameters separated by commas. For example,

```
Asm -print nowarn,noobj,nopage  ...
```

Note that single-letter options are provided for some of the settings: **-f** (NOPAGE), **-h** (NOHDR), **-p** (STAT), and **-w** (NOWARN).

-p Write assembly progress information (module names, includes, loads, and dumps) and summary information (number of errors, warnings, and compilation time) to the diagnostic output file. (This option is the same as PRINT STAT.)

-s Set PRINT NOOBJ to generate a shortened form of the listing file. If the **-l** option is also specified, the rightmost option takes precedence.

-sym off Do not write object file records containing information for SADE, the MPW symbolic debugger. This is the default and will be in effect if no **-sym** option is specified.

-sym [on | full]

Write complete object file records containing information for use by SADE. The options **on** and **full** are equivalent. The symbolic information generated by the assembler consists of Module Begin (entry) OMF records for identifiers defined by the PROC, FUNC, and MAIN directives; Local Identifier OMF records for all EQU and SET identifiers except for those identifiers defined in the files included from the {AIncludes} folder; and Local Label OMF records for the local code labels.

-t Display the assembly time and the number of lines to the diagnostic file even if progress information (**-p**) is not currently displayed.

-w Suppress warning messages (same as PRINT NOWARN).

-wb Suppress branch warning messages only.

Example

```
Asm -w -l Sample.a Memory.a -d Debug
```

Assembles Sample.a and Memory.a, producing object files Sample.a.o and Memory.a.o. Suppresses warnings and defines the name Debug as having the value 1. Two listing files are generated: Sample.a.lst and Memory.a.lst. (Sample.a and Memory.a are located in the AExamples directory.)

See also *MPW 3.0 Assembler Reference*.

Backup—folder file backup

Syntax backup [*option* ...] -from *folder* -to *folder* [*file* ...]

Description Files in a source ("from") folder are copied to a destination ("to") folder based on the modification date. By default, only files that already exist in both the source and destination folders are candidates for copying. (The **-a** option can override this default.) Backup does not actually make the copies. Instead, it generates a script of MPW Shell duplicate commands.

Backup's default operation is based on the premise that you already have an existing folder on two sets of disks (generally a hard disk and a set of 3.5-inch disks—drive numbers may be specified as folder "names") and that you want to make sure that the files on one of the disks are the same as the files on the other disk. Thus, it is the files on the destination ("to") disk that determine which files can be copied from the source ("from") disk.

A Shell duplicate command is generated to the standard output file if

- a file on a source disk also exists on the destination disk, and
- the modification date of the source is newer than that of the destination.

In addition to the basic function of generating Shell duplicate commands, Backup also provides these services:

- Folders can be recursively processed, allowing processing of all folders and subfolders contained within folders (**-r**).
- Compare commands can be generated for out-of-date files of type TEXT to discover why the files are different (**-compare**).
- Filenames that exist on one disk and not on the other can be displayed (**-check from,to**).
- File folder names that don't exist on the destination can be displayed (**-check folders**).
- Filenames in the destination that are *newer* than the source can be displayed (**-check newer**).

Type Tool.

Input None.

Output

For each file to be copied, a Shell Duplicate command is generated to the standard output file as follows:

```
Duplicate -y  FromFile ToFile
```

Duplicate's **-y** option may be suppressed by using Backup's **-y** option. If you are using the **-e** option, then the Shell's Eject commands are generated at the end of the list of Duplicates. These commands cause the source and/or destination disks to eject if the **-from** and **-to** options specify as parameters either or both disks as disk drive numbers 1 or 2.

If you use the **-compare** option, a Compare command is written to the standard output file if the files are of type TEXT. Note that only the Compare is generated if you specify **-compare only**. You can also specify all additional Compare command options with the Backup **-compare** option.

Diagnostics

Errors and warnings are written to the diagnostic output file. If you specify the **-p** option, and the diagnostic file is not the same as the standard output file, then a summary of all duplicate commands generated is written to the diagnostic output file. The summary shows the modification dates of both the source and destination files. If you use the **-check** option, a report is written to the diagnostic output file that includes any files in one folder that don't exist on the other folder, and any files in the destination folder that are newer than the source. You can redirect this report to a file by using the **-co** option.

Status

Backup may return the following status codes:

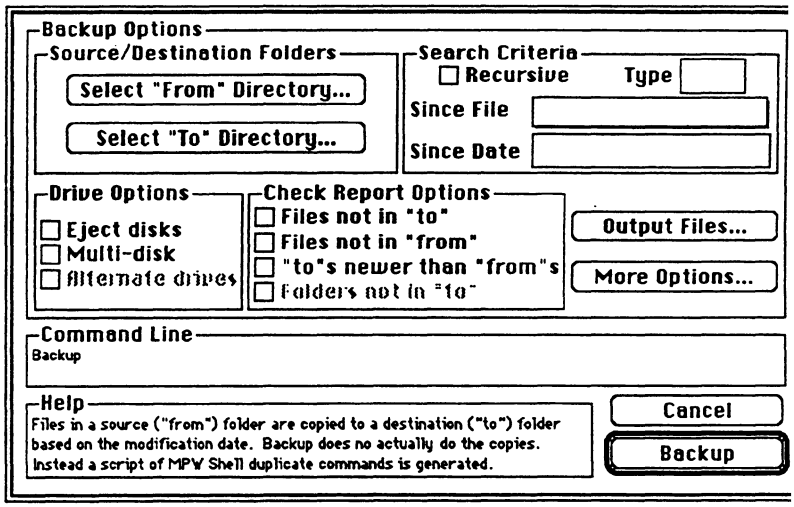
0 No errors; Shell duplicate commands have been generated or filenames were listed.
1 Parameter or option errors.
3 No errors and no files to duplicate or list.

- *Note:* Backup returns a status code of 3 when no files need copying. If no files are copied because *none* of the files in the source folder exists in the destination folder, Backup also reports a warning to the diagnostic output file. If there are no name matches, it is possible that your from/to pathnames were specified incorrectly. Hence, Backup lets you know of the possible error. Backup does not report this as an error if you use the **-l**, **-a**, or **-since** option.

Options

-a Normally, a file in the source ("from") is ignored if it does not exist in the destination ("to"). Using the **-a** option forces Backup to generate a Shell Duplicate command for all files in the source that don't exist in the destination.

-alt If you use this option with the **-m** ("multidisk") option, Backup will alternate the drive numbers when it asks for additional disks. This option has meaning only if either **-from** or **-to,** but not both, specifies a disk drive (1 or 2).



-c Create folders. When a folder name doesn't exist in the destination disk and there are files in the source to copy, **-c** generates a Shell Newfolder command to create the folder on the destination disk. Note that this option makes sense only if you are using the **-a** option.

-check *checkopt* [,*checkopt*]...

Produce reports on the source and destination based on the *checkopt* parameters. *Checkopt* may be any one of the following parameter words:

from	Report all files in the source ("from") folder that don't exist in the destination ("to").
to	Report all files in the destination ("to") folder that don't exist in the source ("from").
allfroms	Same as **from,** but report *all* folders processed, even if there are no files in that folder to report.
alltos	Same as **to,** but report *all* folders processed, even if there are no files in that folder to report.
folders	Report all source ("from") *folders* that don't exist as destination ("to") folders when recursively (**-r**) processing folders. Note that only the outermost folder names are reported.
newer	Report all files in the destination ("to") that are *newer* than the source ("from").

Note: The **-check** option is ignored if the **-since** option is used.

-co *filename* Normally the **-check** report is written to the diagnostic output file. The **-co** option allows you to redirect the report to the specified *filename.*

-compare [only] [,'*option* ...'] | '*option* ...'

Generate Compare commands for all files of type TEXT that are to be duplicated. If **only** is specified, then *only* the Compares are generated, not the Duplicates. Additional Compare command options and output redirection can be specified. Make sure that the Compare options you include are correct, because Backup does not check for you. A period (.) may be used to indicate that there are no Compare options. Note that, in general, the Compare options must be enclosed in quotation marks to ensure that they are not used as Backup options.

-d Generate Delete commands for all files in the destination ("to") folder that don't exist in the source ("from"). If this option is specified, the options **-check to**, **-check alltos**, **-m**, **-l**, and **-since** cannot be used.

-do [only],'*command...*' |'*command...*'

Generate the command string specified by *command...* for all files that are to be duplicated. If **only** is specified then *only* the command string is generated, not the duplicates. The **-do only** option may not be specified when the **-compare only** option is specified. When the command string is generated, the source ("from") and destination ("to") pathnames are added to the command string as the last two (or only) parameters like this:

command... fromFilename toFilename

If **-sync** is specified, the same command is generated but with one additional parameter to indicate the direction. If the source has a newer modification date than the destination (the standard mode of copying the source to the destination), a command string like this is generated:

command... fromFilename toFilename '-->'

If the destination has a newer modification date than the source, the following command string is generated:

command... fromFilename toFilename '<--'

If **-l** is specified then **-do only** is implied, and the command string, rather than a directory listing, is generated for each source ("from") file, like this:

command... fromFilename

-e Eject the disk from drive 1 and/or drive 2 if **-from** or **-to** specify drive number 1 or 2. Disks are ejected when Backup terminates if there are no files to duplicate. If Duplicate commands are generated, then Shell eject commands are generated to eject the specified disk(s).

-from *folder* | *drive*

Specify the folder *or* drive number (1 or 2) from which to get the source list of files. If this option is omitted, the list may be specified as a sequence of filename parameters to Backup (for example, folder:≈). If both **-from** and a list of files are omitted, then drive 1 is assumed (that is, **-from 1**) if the **-to** parameter is explicitly specified. The **-from** option must be specified if **-to** is omitted or the **-l** option is used. You can use the Shell wildcard character, "≈", to do limited pattern matching when specifying a **-from** folder. However, you must quote such folder specifications to allow Backup (rather than the Shell) to process the pattern. The difference between specifying **-from** and supplying a list of filenames is that **-from** *always* implies that the files belong to the specified folder, whereas a list of files explicitly specifies those files. Using **-from** is more efficient than using the list, but the list allows more complicated patterns.

-l

Generate a list of all files in the source ("from") folder. The **-to** option cannot be specified when **-l** is used. If the **-do** option is specified, the **-do** command string, rather than a file listing, is generated for each file.

-lastcmd '*cmd*'

Generate the specified command as the last command. For example, a Beep command could be generated to signal that all the duplicates have been completed.

-level *nestinglevel*

Used to qualify the **-a** option. **-level** restricts the copying of all files in the source that don't exist in the destination to those contained in folders nested at a level greater than or equal to the specified nesting level. This minimum nesting level is relative to the folder specified by the **-from** option. The **-from** folder is considered level 0. Folders contained in it are level 1, and so on. The preset **-a** qualifying nesting level is 0; that is, all files in the source that don't exist in the destination are copied as specified for **-a**. Because the value of the nesting level is relative, it may take some experimentation to produce the desired effects.

-m

Multidisk operation. Backup will display a dialog box asking for additional disks to be mounted in drive 1 or 2 (depending on whether **-from** or **-to** specifies drive 1 or 2). This option is ignored if *both* **-from** and **-to** specify disk drives.

-n When recursion (**-r**) is specified, generate the Duplicate commands for files nested in inner folders with leading spaces to show the nesting structure.

-p Write Backup's version number and a report of all Duplicate commands generated to the diagnostic output file. The report is not produced if the diagnostic output file and the standard output file are the same.

-r Recursively process subfolders encountered.

-revert Revert all newer files in the destination ("to") folder to their state in the source ("from") folder. The default mode for Backup is to copy only those files in the source folder that are newer than files of the same name in the destination folder. By specifying **-revert,** the copy criteria are reversed. *Only* files in the destination that are newer than those in the source are copied. This option is useful for reverting a newer file to its previous state in an older backup copy.

-since *date*[,*time*] | ,[*time*] | *filename*

Generate Duplicate commands to the destination ("to") folder for *all* files in the source ("from") folder(s) that have a modification date greater than or equal to the specified *date* and *time*, or a date and time determined from the modification date of the specified *filename*. This is a special option that unconditionally copies files that satisfy the date/time requirements. Files and folders in the destination folder are ignored. This option is useful, for example, for copying to a single disk all files changed since a certain time. The *date* is specified in the form mm/dd/yy. The day ("dd") and/or year ("yy") may be omitted. The *time* is specified as hh:mm:ss. The minutes ("mm") and/or seconds ("ss") may be omitted. An entire *date* or the *time* may be omitted. If both are omitted, the comma is still required. If the *date* is omitted, the current date is used. If the time is omitted, time 00:00:00 is used.

As an alternative to specifying an explicit date and/or time, you can supply a filename. The modification date and time of that file will be used as the **-since** date and time.

Note: Because the structure of the destination folder is ignored when you use the **-since** option, Duplicate commands may be generated for the same filename from different source folders. It is recommended that you use the **-y** option to suppress the Duplicate **-y** options when using **-since.**

-sync Synchronize both source ("from") and destination ("to") folders. Files are copied in both directions; source files newer than those in the destination are copied to the destination, and destination files newer than those in the source are copied to the source. The **-sync** option may not be specified when any of the options **-revert, -since,** or **-a** is specified.

Note: Use this option with caution, because it can cause copies in the opposite direction from that specified as **-from** and **-to**. An example of its safe use is with the **-compare only** option. This option will generate compare commands for all TEXT files that differ in their modification dates in either direction.

-t *type* Consider only files of the specified *type* as candidates for Backup.

-to *folder* | *drive*

Specify the folder or drive number (1 or 2) from which to get the destination list of files. If the **-to** option is omitted and the **-from** parameter is explicitly specified, then drive 1 is assumed (that is, **-to** 1). You must specify the **-to** option if you omit the **-from** option.

-y Suppress generation of the Shell Duplicate command's **-y** option.

Examples

```
backup  -from :HDfolder:  -e
```

Check that all files on the disk in drive 1 (**-to** is omitted, so "**-to** 1" is implied) are up to date with respect to the files in :HDfolder:. If they are, the disk in drive 1 is ejected. If not, the appropriate Duplicate commands are generated to update the out-of-date files on the disk in drive 1. An Eject 1 command is generated to eject the disk after the Duplicate commands are processed.

```
backup  -r  -from FServer:MPW:  -to HD:MPW:  -check folders
```

Recursively process (**-r**) all the files in all the folders on FServer:MPW: to make sure that the files on HD:MPW: are up-to-date. Appropriate Duplicate commands are generated to copy the out-of-date files from the folders in FServer:MPW: to the folders in HD:MPW:. It is assumed that the folder names in HD:MPW: are the same as the folder names in FServer:MPW:. Any folders in FServer:MPW: that don't exist in HD:MPW are skipped. Because the **-check** option is specified, a list of all the skipped folders is written to the diagnostic file.

Limitations

Multi-disk operation (**-m**) is not supported with recursion (**-r**).

The **-e** option is ignored when **-m** is specified.

Only drive numbers 1 and 2 are supported, and they are assumed to be ejectable 3.5-inch disk drives.

Beep—generate tones

Syntax Beep [*note*[,*duration*[,*level*]]]...

Description For each parameter, Beep produces the given note for the specified duration and sound level on the Macintosh speaker. If no parameters are given, a simple beep is produced.

Note is one of the following:

- A number indicating the count field for the square wave generator, as described in chapter "Summary of the Sound Driver" of *Inside Macintosh.*
- A string in the following format:

  [*n*] *letter*[# | b]

  n is an optional number between –3 and 3 indicating the octaves below or above middle C, followed by a letter indicating the note (A–G) and an optional sharp (#) or flat (b) sign. Note that any sharps (#) must be enclosed in quotation marks—otherwise they will be interpreted as comment delimiters.

The optional *duration* is given in sixtieths of a second. The default duration is 15 (one-quarter second).

The optional sound *level* is given as a number from 0 to 255. The default level is 128.

Type Built-in.

Input None.

Output None.

Diagnostics None.

Status A status code of 0 is always returned.

Options None.

Examples

```
Beep
```

Produce a simple beep on the speaker.

```
Beep 2C,20 '2C#,40' 2D,60
```

Play the three notes specified: C , C sharp, and D—all two octaves above middle C—for one-third, two-thirds, and one full second, respectively. Notice that the second parameter must be quoted; otherwise the sharp character (#) would indicate a comment.

Begin...End—group commands

Syntax

Begin
command...
End

Description

Groups commands for pipe specifications, conditional execution, and input/output specifications. Carriage returns must appear at the end of each line as shown above, or be replaced with semicolons (;). If the pipe symbol (|), conditional execution operators (&& and ||), or input/output specifications (<, >, >>, ≥, ≥≥, Σ, ΣΣ) are used, the operator must appear after the End command and applies to all of the enclosed commands.

- *Note:* Begin and End behave like left and right parentheses. Once the Begin command has been executed, the Shell will not execute any of the subsequent commands until it encounters the End command, so that input/output specifications can be processed.

Type

Built-in.

Input

None.

Output

None.

Diagnostics

None.

Status

The status code of the last command executed is returned. (If no commands appear between Begin and End, 0 is returned.)

Options

None.

Examples

The following commands save the current variables, exports, aliases, and menus in the file SavedState.

```
Begin
    Set
    Export
    Alias
    AddMenu
End > SavedState
```

Notice that the output specification following "End" applies to all of the commands within the Begin...End control command. This command is identical to the following:

```
(Set; Export; Alias; AddMenu) > SavedState
```

The commands Set, Export, Alias, and AddMenu write their output in the form of commands; these commands can be executed to redefine variables, exports, aliases, and menus, respectively. Therefore, after executing the above commands, the command

```
Execute SavedState
```

will restore all of these definitions. You must "execute" the script so that the variables and aliases are applied to the current scope.

- *Note:* This technique is used in the Suspend script to save state information. (You might want to take a look at Suspend, which also saves the list of open windows and the current directory.) The Resume file runs the file that Suspend creates, restoring the various definitions, reopening the windows, and resetting the current directory.

Break—break from For or Loop

Syntax Break [If *expression*]

Description If *expression* is nonzero, Break terminates execution of the immediately enclosing For or Loop command. (Null strings are considered zero.) If the "If *expression*" is omitted, the break is unconditional. (For a definition of *expression,* see the Evaluate command.)

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status Break may return these status codes:

0 No errors detected.
−3 Break is found outside a For...End or Loop...End, or the parameters to Break are incorrect.
−5 Invalid expression.

Options None.

Examples

```
Set Exit 0
For file in Startup UserStartup Suspend Resume Quit
    EnTab "{file}" > temp
    Break If {Status} != 0
    Rename -y temp "{file}"
    Print -h "{file}"
    Echo "{file}"
End
```

This For loop entabs and prints each of the special MPW scripts; the Break command terminates the loop if a nonzero status value is returned. (See the For command for further explanation of this example.)

```
Set loopcount 1
Loop
    Break if {loopcount} > 10
    Echo "Loop Number {loopcount}"
    Evaluate loopcount +=1
End
```

This example loops until the variable {loopcount} is greater than 10. Use of the Evaluate command is also demonstrated.

See also

For, Loop, and If commands.

Evaluate command (for a description of expressions).

"Structured Commands" in Chapter 5.

BuildCommands—show build commands

Syntax BuildCommands program [*options...*]

Description BuildCommands writes to standard output the commands needed to build the specified program.

Make is used to generate the build commands. If file *program*.make exists, it is used as the makefile. If not, file MakeFile is used.

The specified options control the generation of the build commands. The options are passed directly to Make. BuildCommands is used to implement the Show Build Commands and Show Full Build Commands menu items in the Build Menu.

Type Script.

Input None.

Output The commands needed to build the specified program are written to standard output.

Diagnostics Errors and warnings are written to diagnostic output. They may be written either by BuildCommands or by Make.

Status Status code 0 is returned if the build commands are generated without error. If an error occurs, the status code returned by Make is returned.

Option The options specified are passed directly to Make and control the generation of the build commands. Although other Make options may be used, the most useful is **-e.**

-e Generate complete build commands, regardless of file dates. Ignore any up-to-date object files or other temporary files that may already exist.

Example

```
Open "{Worksheet}"
BuildCommands Count >> "{Worksheet}" ≥≥ Dev:StdOut
```

Generates the build commands for Count. The Worksheet window is brought to the front. The build commands, or any errors generated by Make are written at the end of the Worksheet. The Show BuildCommands menu item is implemented using similar commands.

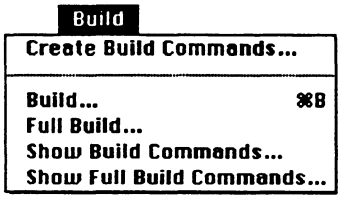
See also

"Building a Program: An Introduction" in Chapter 2.

BuildMenu—create the Build menu

Syntax BuildMenu

Description Creates the Build menu shown below. Each of the items in the menu is described in Chapter 3.



Type Script.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status A status code of 0 is always returned.

Options None.

Example `BuildMenu`

Creates the Build menu. This command should appear in the UserStartup file to create the Build menu.

See also "Building a Program: An Introduction" in Chapter 2.

BuildProgram—build the specified program

Syntax BuildProgram program [*option...*]

Description Builds the specified program. A simple transcript of the build, including timing information and commands used to do the build, is written to standard output.

Make is used to determine the commands needed to do the build. If file *program*.make exists, it is used as the makefile. If not, the file MakeFile is used.

The options specified are passed directly to Make; they control the generation of the build commands. BuildProgram is used to implement the Build and Full Build menu items in the Build menu.

Type Script.

Input None.

Output A transcript of the build, including timing information and the commands used to do the build, is written to standard output.

Diagnostics Errors that occur during the generation of the build commands or during the build are written to diagnostic output.

Status Status code 0 is returned if the build is completed without error. If an error occurs during the generation of the build commands, the status value returned by Make is returned. If an error occurs during the build, the status value returned by the build step that detected the error (such as Asm or Link) is returned.

Option The options specified are passed directly to Make, and control the generation of the build commands. Although other Make options may be used, the most useful is **-e.**

-e Rebuild everything, regardless of dates. The specified program is completely rebuilt, ignoring any up-to-date object files or other temporary files that may already exist.

Example

```
Open "{Worksheet}"
BuildProgram -e Count >> "{Worksheet}" ≥≥ Dev:StdOut
```

Completely rebuilds Count. The Worksheet window is brought to the front. The transcript of the build and any errors are written at the end of the Worksheet. The Full Build menu command is implemented using similiar commands.

See also "Building a Program: An Introduction" in Chapter 2.

C—C compiler

Syntax

C [*option*...] [*file*]

Description

Compiles the specified C source file. Compiling file *Name*.c creates object file *Name*.c.o. (By convention, C source filenames end in a ".c" suffix.) If no filenames are specified, standard input is compiled and the object file "c.o" is created.

(Note that SADE object file information cannot be generated for standard input source files.)

See the *MPW 3.0 C Reference Manual* for details of the C language definition.

Type

Tool.

Input

If no filenames are specified, standard input is compiled. You can terminate input by pressing Command-Enter.

Output

If you specify the **-e** or **-e2** options, preprocessor output is written to standard output, and no object file is produced.

Diagnostics

Errors and warnings are written to diagnostic output. If the **-p** option is specified, progress and summary information is also written to the diagnostic output.

Status

The following status codes may be returned:

0 Successful completion.
1 Errors occurred.

Options

-b Generate PC-relative references for functions in the same segment and for string constants (which are kept at the end of the function code module). Useful for writing DAs, WDEFs, and so on.

-b2 Same as the **-b** option, but also allows the code generator to reduce code size by overlaying string constants where possible.

-b3 Allow the code generator to keep string constants in the code segment and overlay them when possible (but always generate A5 relative references for function addresses).

-c Don't call the code generator.

-d *name* Define *name* to the preprocessor with value one. This is the same as writing

```
#define name 1
```

at the beginning of the source file. (The **-d** option does not override `#define` statements in the source file.)

-d *name=string*

Define *name* to the preprocessor with value *string*. This is the same as writing

```
#define name  string
```

at the beginning of the source file.

-e Do not compile the program. Instead, write the output of the preprocessor to standard output. This option is useful for debugging preprocessor macros.

-e2 Same as the **-e** option, but also suppresses comments.

elems881 Use in-line MC68881 instructions for all transcendental functions available on the MC68881 processor. See the *MPW 3.0 C Reference* for a complete list of these functions. This option implies the **-mc68881** option.

-i *pathname*[,*pathname*]...

Search for `include` files in the specified directories. Multiple **-i** options may be specified. A maximum of 15 directories can be searched. This is the search order:

1. The `include` filename is used as specified. If a *full pathname* is given, no other searching is applied. If the file wasn't found, and the pathname used to specify the file is a *partial pathname* (no colons in the name or a leading colon), the following directories are searched:
2. The directory containing the current input file.
3. The directories specified in the **-i** options, in the order listed.
4. The directories specified in the Shell variable {Cincludes}.

-m Generate 32-bit data references. More than 32K of global data is assumed. The object code is less efficient.

-mbg off Don't include symbols for the MacsBug debugger.

-mbg full Include full (untruncated) symbols for MacsBug.

-mbg ch8 Include MPW 2.0-compatible MacsBug symbols (eight characters only, in a special format).

-mbg *number*
Include MacsBug symbols truncated to length *number*.

-mc68020 Generate MC68020 instructions whenever doing so would provide faster and/or smaller object code.

-mc68881 Generate MC68881 instructions for all basic floating-point operations.

-n Change errors of pointer assignment incompatibility into warnings.

-o *objname* Pathname for the generated object file. If *objname* ends with a colon, it indicates a directory for the output file, whose name is then formed by the normal rules (that is, *inputfilename*.o). If *objname* does not end with a colon, the object file is written to the file *objname*.

-p Write progress information (include file names, function names, and sizes) and summary information (number of errors and warnings, code size, global data size, and compilation time) to diagnostic output.

-r Emit a warning when calling a function that doesn't have a definition.

-s *name* Name the object code segment. (The default segment name is "Main".)

-sym off Do not emit SADE object file information.

-sym on | full
Write complete object file records containing information for SADE, the MPW symbolic debugger. This option can be limited by also specifying one or more of **nolines**, **notypes**, and **novars**, which causes omission of line, type, and variable information, respectively, from the object file.

-t Write compilation time to diagnostic output.

-u *name* Undefine the predefined preprocessor symbol *name*. This is the same as writing

```
#undef name
```

at the beginning of the source file.

-w Suppress compiler warning messages. (By default, warnings are written to diagnostic output.)

-w2 Emit even more warnings about constructs that the compiler has reason to suspect.

-y *pathname*
Put the compiler's temporary intermediate (".o.i") files in the directory specified by *pathname*.

Example

```
C -p Sample.c
```

Compiles Sample.c, producing the object file Sample.c.o. Writes progress information to diagnostic output. (Sample.c is found in Examples:CExamples.)

See Also *MPW3.0C Reference.*

Canon—canonical spelling tool

Syntax Canon [-s] [-a] [-c *n*] *dictionaryFile* [*inputFile*...]

Description Canon copies the specified files to standard output, replacing identifiers with the canonical spellings given in *dictionaryFile*. If no files are specified, standard input is processed.

DictionaryFile is a text file that specifies the identifiers to be replaced and their new (or canonical) spellings. Identifiers are defined as a letter followed by any number of letters or digits. (The underscore character (_) is also considered a letter.) Each line in the dictionary contains either a pair of identifiers or a single identifier:

- If *two identifiers* appear, the first is the identifier to replace, and the second is its canonical spelling. For example, the dictionary entry

  ```
  NIL  NULL     # change NIL to NULL
  ```

  changes each occurrence of "NIL" to "NULL".

- A *single identifier* specifies both the identifier to match and its canonical spelling. This feature is useful because the matching may not be case sensitive or restricted to a fixed number of characters. (See the "Options" section on the next page.) For example, the dictionary entry

  ```
  true
  ```

  changes all occurrences of "TRUE", "True", "tRUE", and so on to "true".

You can specify a left context for the first identifier on each line of the dictionary by preceding it with a sequence of nonidentifier characters. Replacement will then occur only if the left context in the input file exactly matches the left context in the dictionary. For example, if C structure component upperLeft should be replaced with topLeft, the dictionary might include the following:

```
.upperLeft  topLeft
->upperLeft  topLeft
```

You can include comments in the dictionary file by using the # symbol: everything from the # to the end of the line is ignored.

◆ *Note:* The file Canon.Dict is a sample dictionary file that's included with MPW. (See the "Examples" section below.)

Type Tool.

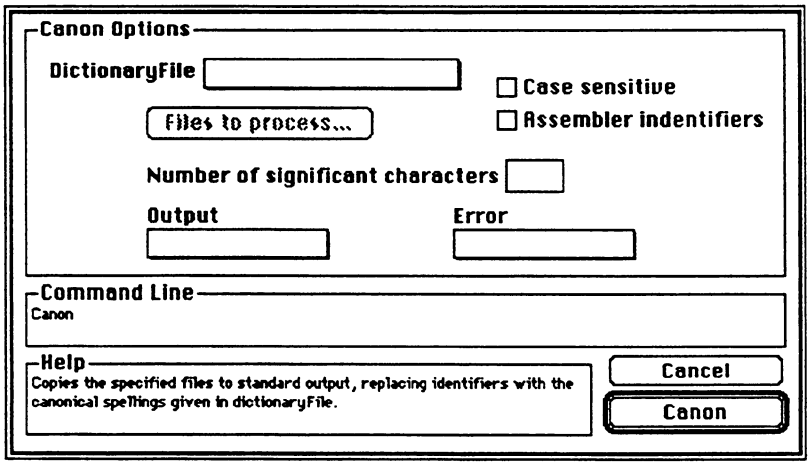
Input Standard input is read if no files are specified.

Output The specified files are written to standard output with the identifiers replaced.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

0 All files processed successfully.
1 Error in command line.
2 Other errors.



Options

-s Use case-sensitive matching. (Pattern matching is normally not case sensitive.

-a Causes the characters $, %, and @ to be considered letters (for defining identifiers). This option is useful when processing an assembly language source.

-c *n* Take only the first *n* characters as significant. (Normally all characters in identifiers are significant.)

Examples

The file Canon.Dict, in the Tools folder, contains a list of all of the identifiers used in the Standard C library and the *Inside Macintosh* C interfaces. This list was made from the Library Index in the *MPW 2.0 C Reference*. The entries in Canon.Dict look like the following:

```
abbrevDate
ABCallType
abortErr
ABProtoType
abs
acos
activateEvt
...
```

The following command copies the file Source.c to the file Temp; identifiers whose first eight characters match a dictionary entry are replaced with that entry.

```
Canon -c 8 "{MPW}"Tools:Canon.Dict Source.c > Temp
```

The `-c 8` option is useful when porting source code from other systems where only eight characters are significant.

- *Note:* The list of Pascal identifiers used in the *Inside Macintosh* interface is almost identical to the list used in C. The dictionary Canon.Dict can also be used to port Pascal programs from other systems, as long as you use the canonical capitalizations for the various Standard C library identifiers.

Limitations

The maximum line length in the dictionary file is 256 characters. Longer lines are considered an error. Identifiers and words in comment sections are replaced.

Catenate—concatenate files

Syntax Catenate [*file...*]

Description Catenate reads the data fork of each file in sequence and writes it to standard output. If no input file is given, Catenate reads from standard input. None of the input files may be the same as the output file.

Type Built-in.

Input Standard input is processed if no input files are specified.

Output All files are written to standard output.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

0 All files were processed successfully.
1 One or more files were not found.
2 An error occurred in reading or writing.

Options None.

Examples

```
Catenate Makefile.a
```

Writes Makefile.a to the active window immediately following the command.

```
Catenate File1 File2 > CombinedFile
```

Concatenates the first two files and places the result in the third. If CombinedFile doesn't exist, it will be created; if it exists, it will be overwritten.

```
Set selection  "`Catenate §`"
```

Captures the selection from the target window in the Shell variable {selection}.

```
Catenate >> {Worksheet}
```

Appends all subsequently entered text to the Worksheet window (until you indicate end-of-file by pressing Command-Enter).

Warning Beware of commands such as

```
Catenate File1 File2 > File1
```

The above command will cause the original data in File1 to be lost. To append one file to another, use the form

```
Catenate File2 >> File1
```

See also Duplicate command.

"Redirecting Input and Output" in Chapter 5.

CheckIn—check in files to a project

Syntax

CheckIn -w | -close | [-u *user*] [-project *project*] [-t *task*] [-p] [-cs *comment* | -cf *file*] [-new | -b] [-m | -delete] [-touch] [-y | -n | -c] [-a] *file...*

Description

Return ownership of the specified files to Projector and save all changes as new revisions. The default is to leave you with a read-only copy of the file. *File* must be an HFS pathname. Projector determines the project each file belongs to by inspecting the file's resource fork. Since Projector puts the name of the project in the resource fork of checked-out files, files belonging to different projects can be checked in with a single command.

If the **-a** (all) option is used instead of *file...*, Projector examines all files in the current directory and checks in all files in the current directory that have been checked out for modification. The files are checked into their respective projects.

To add a new file to the project, use the **-new** option.

When the file is checked in, Projector automatically increments the revision number by one. For example, if revision 2.17 was checked out, the new revision will be 2.18. To override this, use the ProjectInfo command to find the revision number, increase it by the amount desired, and then check the file in, using the "filename,rev" notation. For example, if file.c revision 2.17 was checked out, you could check it in as file.c,3.0 to jump to the next major revision level.

See Chapter 7 for complete definitions of the terms and symbols used in Projector commands.

Type

Built-in. Option **-u** 'user' is required if the Shell variable {user} is null.

Input

None.

Output

Progress is written to standard output if the **-p** option is specified.

Diagnostics

Errors and warnings are written to diagnostic output.

Status

The following status codes may be returned:

0 No errors.
1 Syntax error.
2 Error in processing.
3 System error.

Options

-w Open the Check In window.

-close Close the Check In window.

-u *user* Name of the current user. This option overrides the {User} shell variable.

-project *project*
Name of the project that contains the files. This becomes the current project for this command.

-new Add a new file to the project.

-t *task* A very short description of the task that was accomplished by the changes made to the file(s). This task overrides the task found in the `'ckid'` resource of each file specified.

-cs *comment*
A short description of what changes have been made to the file(s) being checked in. This comment overrides the comment found in the `'ckid'` resource of each file specified.

-cf *filename* The comment is contained in the file *filename*. This comment overrides the comment found in the `'ckid'` resource of each file specified.

-a Check in all the files in the current directory.

-b Check in the file as a branch off the revision that was checked out.

-m Keep a write-privileged copy of the file(s) for further modification. This basically does a check-in followed by a check-out for modification of the new revision.

-delete Delete the file after checking it in.

-p Write progress information to standard output.

-touch Touch the modification date of the file before checking it in.

-y Answer "Yes" to all dialogs (doing so avoids the dialogs).

-n Answer "No" to all dialogs (doing so avoids the dialogs).

-c Cancel the dialog if a conflict occurs (doing so avoids the dialogs).

Examples

```
CheckIn file.c -cs "added some comments"
```

Check in file.c to the current project. A new revision of file.c is created and the user is left with a read-only copy of the file. The comment is saved with the new revision. Because no revision number is specified, Projector simply increases the revision number by one.

```
CheckIn file.c interface.c,5 -t "Added -x option" ∂
    -cf commentFile
```

This command checks in two files reading the comment from the file commentFile. The task is also saved with the new revisions. The user is left with read-only copies of the files. The new revision for interface.c is revision 5.

```
CheckIn hd:work:file.c hd:work:main.c -m
```

The files to be checked in are hd:work:file.c and :main.c. After the command executes, the user still has modifiable copies of the files.

```
CheckIn -new file.c
```

To check a new file into the project use the **-new** option. The above command adds file.c to the current project.

```
CheckOut -project Zoom∫Utilities∫MyProject file.c -m
...edit the file...
CheckIn -project Zoom∫Utilities∫MyProject file.c -b
```

The preceding command sequence illustrates the usefulness of the **-b** option. In this case, the user checked out a write-privileged copy of the latest revision of file.c from the current project, edited the file, and then, using the branch option, checked in the file on a branch.

See Also CheckOut and CheckOutDir.

CheckOut—check out files from a project

Syntax

CheckOut -w | -close | [-u *user*] [-project *project*] [-m | -b [-t *task*]] [-cs *comment* | -cf *file*] [-d *directory*] [-r] [-open] [-y | -n | -c] [-p] [-noTouch] [-cancel | -update | -a | -newer | *file...*]

Description

Under Projector, CheckOut obtains copies of file revisions from a project. The default is to check out read-only copies. Unless otherwise specified, copies are placed in the checkout directory associated with the project.

If *file* is a leafname (that is, file.c), Projector checks out the latest revision of the file from the current project. If *file* specifies a revision (for example, file.c,22), that revision is checked out.

If *file* is a partial or full HFS pathname (that is, :work:file.c or HD:work:file.c), the file does not go into the checkout directory. Instead, Projector checks out the file (that is, file.c) in the current project and places the copy in the specified HFS location (that is, in the :work: or HD:work: directory, respectively).

Finally, *file* may be a Name. See the NameRevisions command for more information about Names. The Name is expanded and the corresponding revisions are checked out.

To check out an old revision for modification, you must specify the **-b** (branch) option.

If you are checking out revision 5 of file.c into hd:work and Projector determines that you already have that revision in the work directory, Projector will not recopy the data of revision 5. This is especially nice when you are checking out a revision for modification, and you already have a read-only copy of that revision.

See Chapter 7 for complete definitions of the terms and symbols used in Projector commands.

Type

Built-in.

Input

None.

Output

Progress is written to standard output if the **-p** option is specified.

Diagnostics

Errors and warnings are written to diagnostic output.

Status

The following status codes may be returned:

0 No errors.
1 Syntax error.
2 Error in processing.
3 System error.

Options

-w Open the Check Out window.

-close Close the Check Out window.

-u *user* Name of the current user. This overrides the {User} shell variable.

-project *project*
Name of the project that contains the files. This becomes the current project for this command.

-d *directory* The directory to which the checked-out files should go. This overrides the checkout directory for the current project. See the CheckOutDir command.

-t *task* A very short description of the task to be accomplished by checking out files for modification.

-cs *comment*
A short description of what changes will be made to the file(s) being checked out.

-cf *filename* The comment is contained in the file *filename.*

-a Check out all the files in the Project.

-m Check out a write-privileged copy of the file for modification. This locks the revision, preventing other users from inadvertently changing the revision.

-open Open the files after checking them out. This only works on files of type `text`.

-b Create a branch. A write-privileged copy of the file is checked out. When the file is checked back in, it will become a branch of the revision that was checked out.

-update Find all read-only copies of the project in the checkout directory (or the **-d** directory) and update them to the latest revision if they are older revisions. Files that have been checked out for modification, or that are on branches, are not affected. This option cannot be used when checking out files for modification.

-p	Write progress information to standard output.
-r	Recursively execute the CheckOut command on the current project and all of its subprojects.
-newer	Check out the latest copy of all files in the project. Files that have been checked out for modification, or that are on branches, are not affected. This option cannot be used when checking out files for modification.
-cancel	Cancel the check out of the specified files.
-noTouch	Don't touch the modification date of the checked out files.
-y	Answer "Yes" to all dialogs (avoids the dialogs).
-n	Answer "No" to all dialogs (avoids the dialogs).
-c	Cancel the dialog if a conflict occurs (avoids the dialogs).

Examples

```
CheckOut -m -project Zoom∫Utilities∫MyProject file.c
```

Checks out a write-privileged copy of the latest revision of file.c from the Zoom∫Utilities∫MyProject project. The file is placed in the checkout directory for the project.

```
CheckOut -m file.c
```

The above command checks out the latest revision of file.c for modification. The file is placed in the checkout directory for the project. If you already happen to have the latest revision of file.c in the checkout directory, then Projector only updates the `'ckid'` resource of file.c to indicate that it is now a modifiable file.

```
CheckOut -project Zoom∫Utilities∫Kerfroodi file.c,22
```

The above command checks out a read-only copy of revision 22 of file.c from the Zoom∫Utilities∫Kerfroodi project. The file is placed in the checkout directory for the project.

```
Project Zoom∫Utilities∫Kerfroodi
CheckOut file.c -t "Fix Bug 7" -m -d∂
"{Zoom}UtilitiesSrc:Kerfroodi"
```

By setting the current project with the Project command you don't need to specify a project on subsequent Projector commands. By setting the task other users will be able to see why you have checked out file.c. The files are placed in {Zoom}UtilitiesSrc:Kerfroodi.

```
CheckOut -a -d HD:Work:Test
```

The above example checks out read-only copies of all of the files in the current project and places the copies in the directory HD:work:Test.

```
CheckOut -a -project Zoom∫ -r
```

Checks out read-only copies of all files in the Zoom project and all of its subprojects. Its behavior is the same as if you had executed these commands individually:

```
CheckOut -a -project Zoom∫
CheckOut -a -project Zoom∫Voom
CheckOut -a -project Zoom∫Utilities
CheckOut -a -project Zoom∫Utilities∫MyProject
...
```

You can conveniently update the read-only files (from the current project) in the current directory without affecting any files checked out for modification. To do this, use the **-update** option:

```
CheckOut -update -d :
```

See Also CheckIn and CheckOutDir.

CheckOutDir—set checkout directory

Syntax

CheckOutDir [-project *project*] [-m] [-r] [-x | *directory*]

Description

Under Projector, CheckOutDir changes the checkout directory associated with the current project to the HFS pathname *directory*. From this point on, files checked out of the named project are placed, by default, into this directory. The directory is created if it does not exist. When a project is mounted, the checkout directory is initially set to ":"—that is, the current directory.

It is recommended that you put CheckOutDir commands immediately following the corresponding MountProject commands you place in your UserStartup file, script, or AddMenu for project initialization.

If *directory* is missing, the checkout directory of the current project is written to standard output in the form of a CheckOutDir command.

See Chapter 7 for complete definitions of the terms and symbols used in Projector commands.

Type

Built-in.

Input

None.

Output

If *directory* is missing, the checkout directory of the current project is listed in the form of a CheckOutDir command.

Diagnostics

Errors and warnings are written to diagnostic output.

Status

CheckOutDir may return these status codes:

0 No errors.
1 Syntax error.
2 Error in processing.
3 System error.

Options

-project *project*
Name of the project with which to associate the checkout directory. This becomes the current project for this command.

-m Display or set the checkout directories for all "mounted" root projects.

-r Recursively display or set checkout directories.

-x Reset the checkout directory back to the default—that is, the current directory ":".

Examples

```
CheckOutDir HD:work:Test
```

This command causes subsequent files in the current project to be checked out to the HD:work:Test folder.

```
CheckOutDir
CheckOutDir -project Zoom∫Utilities∫Test  HD:work:Test
```

The above command outputs the checkout directory of the current project in the form of a CheckOutDir command.

```
CheckOutDir -project Zoom∫ -r
CheckOutDir -project Zoom∫ :
CheckOutDir -project Zoom∫Vroom :
CheckOutDir -project Zoom∫Utilities :
CheckOutDir -project Zoom∫Utilities∫Test HD:work:Test
```

The **-r** option lets you display the checkout directory for the current project and all subprojects. In this case, only the sort project has a checkout directory setting that differs from the default.

The **-r** option can also be used to set the checkout directories of a complex project to mirror the projects own hierarchical structure. For example:

```
CheckOutDir -project Zoom∫ -r HD:Work:
```

After executing the above command, listing the checkout directories for the projects under Zoom yields

```
CheckOutDir -project Zoom∫ -r
CheckOutDir -project Zoom∫ HD:work:
CheckOutDir -project Zoom∫Vroom HD:Work:Vroom
CheckOutDir -project Zoom∫Utilities HD:Work:Utilities
CheckOutDir -project Zoom∫Utilities∫Test
HD:Work:Utilities:Test
```

Notice how the directory structure is similar to the project structure. The directories are created if they do not exist.

The **-m** option lists the checkout directories of the root projects. For example

```
CheckOutDir -m
CheckOutDir -project Zoom∫  HD:Work:Zoom
CheckOutDir -project Test∫  HD:Test
```

See Also MountProject, CheckIn, and CheckOut.

Choose—choose or list network volumes and printers

Syntax Choose [*options...*] *name...*

Description Choose noninteractively mounts or lists the specified AppleShare volumes or printers. Each name takes the form

```
[zone]:[server[:volume]]
```

("Server" means any file or printer server.) The zone name is always optional and defaults to the current zone. A server name must be preceded by (at least) a colon. Volume names are only applicable to file servers.

When mounting file server volumes, a server name is required. If a volume name is specified, only that volume is mounted. If the volume name is omitted, or if it is the wildcard character "≈", all volumes on the server are mounted:

```
[zone]:server:volume
[zone]:server[:≈]
```

When **-list** is specified, the wildcard character "≈" may be used in place of names in all of the fields: "≈" in the zone field expands to all zones; "≈" in the server field expands to all servers in the specified zones; "≈" in the volume-name field expands to volumes on the specified servers (listing volumes on a server requires a server login—that is, as a user with a valid password or as a guest). If the wildcard character "≈" is used, it must be quoted so that the Shell will not expand it.

The **-list** option also expands the next unspecified item in a name. A zone name followed by nothing else expands to a list of servers in that zone, and a server name followed by nothing else expands to a list of volumes on the server.

If a "≈", ":" or "∂" character appears in a server, volume, or zone name, it may be quoted with the character "∂". This quoting mechanism supplements quoting already performed by the Shell.

Any number of volumes may be mounted (though a system-dependent limit exists on the number of active server connections). Only one printer may be chosen at a time, since only one printer can be active.

Server and volume passwords are case sensitive. More than one server and volume may be mounted with a single command, but the server and volume passwords must be the same for each, since at most one password of each type may be specified on the command line.

Input None.

Output If **-list** is specified, the names of zones, servers, and volumes on file servers are printed in a form suitable for reinput to Choose command lines. If **-c** is specified, the name of the tool (plus appropriate options) appears on each output line.

If **-v** is specified, the names of volumes that were mounted are printed.

If **-cp** is specified, the name, type, and driver of the currently chosen printer are printed.

Diagnostics Errors are written to diagnostic output.

Various confusing messages (such as "No AFPLogin call has been successfully made for this session") are usually the result of a missing or mistyped password.

Status The following status codes may be returned:

0 No errors.
1 Syntax error on command line.
3 Any other error.

Options

-list Print information about the specified network entities.

-c Precede each line of **-list** output with the name of the Choose tool (that is, output Choose commands).

-type *typename*
This option sets the type of the network object to choose or list. The type name is not case sensitive. For mounting or listing volumes, the type name defaults to 'AFPServer'; for choosing or listing printers, it defaults to the name of the current printer driver (such as 'LaserWriter'). Use this option to choose or list network entities of other types.

A type name of "≈" or "=" matches all network entity types. You can list or attempt to mount network entities that are not chooseable. For instance, it is not possible to mount or list volumes on servers of types other than 'AFPServer'.

-p Writes Choose's version number and step-by-step progress information to standard output. This is reassuring when you are doing listings that can take several minutes (for example, every server on the internet).

The following options are applicable to file servers only and may not be specified in conjunction with any printer options:

-u *name* Specify the user name for the server log-in. This option has precedence over the Shell variable "{User}", which in turn has precedence over the user name string in the system resource file ('STR ' -16096). If no valid user name is found in any of the above locations, **-guest** is assumed.

-guest Login as a guest instead of with a user name.

-pw*password*
Specify the server log-in password. The server password defaults to the value of the Shell variable "{ServerPassword}".

-vp *password*
Specify the volume log-in password. The volume password defaults to the value of the Shell variable "{VolumePassword}".

-v Print the volume names (only) of any volumes mounted. Colons are appended to each volume name. This is useful in Shell scripts when volume names are not known ahead of time.

The following options are applicable to printers only and may not be specified in conjunction with any file server options:

-pr Specify that a printer is being chosen or listed.

-cp Print the name and type of the currently chosen printer to standard output. This occurs before any new printer is chosen.

-dr *drivername*
Specify the driver name of the printer to choose. This is the name of a printer driver in the system folder (such as "ImageWriter").

Examples

```
Choose :Linker:Sources
```

Mount the volume Sources on the server Linker, located in the current zone, using the default user name, server password, and volume password.

```
Choose -v -guest 'Systems:Sources:Doc' 'Systems:Games:≈'
```

Mount the volume Doc on the server Sources and every volume on the server Games in the zone Systems as a guest. Print the names of the volumes that are mounted by the command. List the names of all zones. Notice that the wildcard character "≈" is quoted.

```
Choose -list 'Whale Zone:≈' 'Whale Zone:Moby Dick:≈' '≈:'
```

List all file servers in the zone Whale Zone, all volumes on the file server Moby Dick in that zone (after logging in with the default user name and server password) and all zones (with their servers).

```
Choose -pr -list ':≈'
```

```
Choose -cp -pr "Zarf:Kitchen Sink"
```

List all printers of the current type in the current zone. Print the name of the currently selected printer, then select the printer called Kitchen Sink in the zone Zarf.

```
Choose -list -type "Fortune Cookie Server" '≈:≈'
```

List all network entities of type Fortune Cookie Server in all zones.

See also Unmount and Volumes commands.

Clear—clear the selection

Syntax Clear [-c *count*] *selection* [*window*]

Description Finds *selection* and deletes its contents. The selection is not copied to the Clipboard. (For a definition of *selection,* see Chapter 6.)

If *window* is specified, the Clear command acts on that window. It's an error to specify a window that doesn't exist. If no window is specified, the command operates on the target window (the second window from the front).

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status Clear may return the following status codes:

0 At least one instance of *selection* was found.
1 Syntax error.
2 Any other errors.

Option **-c** *count* Repeat the select-and-delete operation *count* times.

Examples

```
Clear §
```

Deletes the current selection. This is like the Clear command in the menu bar, except that the action occurs in the target window rather than the active window.

```
Clear /BEGIN/:/END/
```

Selects everything from the next BEGIN through the following END, and deletes the selection.

See also Cut and Replace commands.

"Selections" in Chapter 6 (see Appendix B for a summary).

Close—close specified windows

Syntax Close [-y | -n | -c] [-a | *window...*]

Description Close the window or windows specified by *window*. If no window is specified, the target window is closed. If changes to the window have not been saved, a dialog box requests confirmation of the Close command. In scripts you can use the **-y**, **-n,** or **-c** option to avoid this interaction. Use the **-a** option instead of *window* to close all of the open windows (other than the Worksheet).

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status Close may return the following status codes:

0	No errors.
1	Syntax error
2	Any other error, such as "Window not found."
4	Cancelled from dialog.

Options

-a Close all open Shell windows (except for the Worksheet, which cannot be closed). This option cannot be specified when any *windows* are specified.

-n Answer "No" to any confirmation dialogs, causing all of the specified windows to be closed without saving any changes.

-y Answer "Yes" to any confirmation dialogs, causing all of the specified windows to be saved before closing them.

-c Answer "Cancel" to any confirmation dialogs, causing any modified windows to be left open.

Examples

```
Close
```

Closes the target window, prompting the user with a confirmation dialog box if needed.

```
Close -a -y
```

Saves and closes all open windows.

```
Close -n Test.a Test.r
```

Closes the windows Test.a and Test.r without saving any of the changes.

See also "File Menu" in Chapter 3.

Commando—display dialog for a command

Syntax Commando [*commandname*] -modify

Description The Commando interface lets you operate any properly configured MPW tool or script using specialized Macintosh dialog boxes instead of the ordinary command line method. The dialogs make it easy to find options and build up complex command lines.

Commands with many options and parameters may employ one or more nested dialog boxes. See "Commando Dialogs" in Chapter 4 for more information on the basics of using the Commando dialogs. Chapter 13 describes the structure of the Commando resource and shows how to create Commando dialogs for your own tools and scripts.

The controls of a Commando dialog box, including text fields, buttons, titles, and so on, can be sized and moved within the dialog box by using the mouse, exactly as you would drag an object in the Finder. See "Editing Commando Dialogs" in Chapter 13 for information on moving and sizing controls.

Type Tool.

Input None.

Output If Commando is invoked by typing Commando [*commandname*], the command line is simply written to standard output. However, the command line is intercepted by the Shell and executed if Commando is invoked by typing [*commandname*] ... (the ellipsis generated by Command-semicolon), or if Commando is invoked by typing [*commandname*] Option-Enter.

Diagnostics Errors are written to diagnostic output.

Status Commando may return the following status codes:

0 The Do It button was selected.
1 The Cancel button was selected.
2 Error occurred while parsing the cmdo resource.
3 I/O or program error.

Option **-modify** Enables Commando's built-in dialog editor.

Examples

```
Commando Rez
```

Displays the frontmost Rez dialog box shown under "Rez" in Part II.

```
Rez...
```

Displays the frontmost Rez dialog box shown under "Rez" in Part II, exactly as in the previous example.

See also

"Invoking Commando" in Chapter 4.
Chapter 13.

Compare—compare text files

Syntax Compare [*option* ...] *file1* [*file2*]

Description Compares the lines of two text files and writes their differences to standard output. Options are provided to compare a specific column range in each file (**-c**), to ignore blanks (**-b**), and to ignore case (**-l**).

Both files are read and compared line for line. As soon as a mismatch is found, the two mismatched lines are stored in two stacks, one for each file. Lines are then read alternately (starting from the next input line in *file2*) until a match is found to put the files back in synchronization. If such a match is found, Compare writes the mismatched lines to standard output.

Files are considered resynchronized when a certain number of lines in the two stacks exactly match. By default, the number of lines, called the *grouping factor,* is defined by the formula

$G = \text{Trunc}((2.0 * \text{Log}_{10}(M)) + 2.0)$

where *G* is the grouping factor and *M* is the number of lines saved in each stack so far. This definition requires more lines to be the same after larger mismatches. Using this formula, the following table shows the grouping factor *G* as a function of the number of mismatched lines:

M: Number of mismatched lines			*G:* Grouping factor
1	to	3	2
4	to	9	3
10	to	31	4
32	to	99	5
100	to	315	6
316	to	999	7
1000	to	3161	8
3162	to	9999	9

With the default dynamic grouping, the **-g** option sets the lower limit for *G* (which must be at least 2, because the formula is always applied). The **-s** option lets you fix *G* as a static constant. A static *G* may be desirable under some circumstances, but may also resynchronize the files at undesirable points, especially if *G* is too small. It's recommended that you use the default (dynamic *G*) first; if the results aren't satisfactory, try a higher minimum value of dynamic G (such as 3 or 4). If that is still unsatisfactory, try the static *G* option.

With either option, there's a limit on the depth of the stacks— that is, on how far out of synchronization the two files can get before they're no longer worth comparing. For a dynamic *G*, the limit on the number of mismatched lines is 1000, but you can choose a lower limit with the **-d** option. For the static *G* option, typical values for *G* are 1 to 5, and the stack depth should be between about 10 and 50 (the default limit is 25).

Type

Tool.

Input

The *file1* and *file2* parameters specify the two files to be compared. If *file2* is omitted, *file1* is compared to standard input.

Output

Mismatched lines, optionally shown with context (**-e**) or suppressed entirely (**-m**), descriptive messages, and Shell editor commands to select the mismatches are written to standard output. With the **-h** option, some of each file's output lines are displayed side by side; otherwise, the first stack's lines are displayed before the second stack's. In either case, lines are shown with their line numbers.

The following messages appear when showing mismatches:

`Nonmatching lines` (*Shell editor commands*)
...both stacks are displayed...

`Extra lines in 1st before` <*line*> `in 2nd` (*Shell editor commands*)
...lines in file1's stack are displayed...

`Extra lines in 2nd before` <*line*> `in 1st` (*Shell editor commands*)
...lines in file2's stack are displayed...

`Extra lines in 1st file` (*Shell editor commands*)
...lines in file1's stack are displayed...

`Extra lines in 2nd file` (*Shell editor commands*)
...lines in file2's stack are displayed...

The Shell editor commands consist of File and Line (Line is provided in the MPW Scripts folder) commands to select the mismatched lines. In the case of extra lines in one file and not the other, the selection for the missing lines is generated as an insert point.

The lines displayed may be suppressed with the **-m** option. If you use **-m** the messages are formatted slightly differently:

```
### Extra lines in 2nd file

    Shell editor commands
```

When mismatched lines are shown, their context can also be displayed by the using the **-e** option. Up to *n* equal lines (*n* is specified with the **-e** option) in both files preceding and succeeding the mismatches will be displayed like this:

...preceding context lines ...

```
----------------------
```

...mismatched or extra lines...

```
++++++++++++++++++++++
```

...succeeding context lines...

If an end-of-file condition occurs or the maximum stack depth is reached during resynchronization, one of the following messages will also appear:

```
*** Nothing seems to match ***

*** EOF on both files ***

*** EOF on file 1 ***

*** EOF on file 2 ***
```

If both files are in synchronization, and both reach their end-of-file at the same time, the following message will appear if *any* mismatches occurred:

```
*** EOF on both files at the same time ***
```

If both files match, the following message is displayed:

```
*** Files match ***
```

Diagnostics

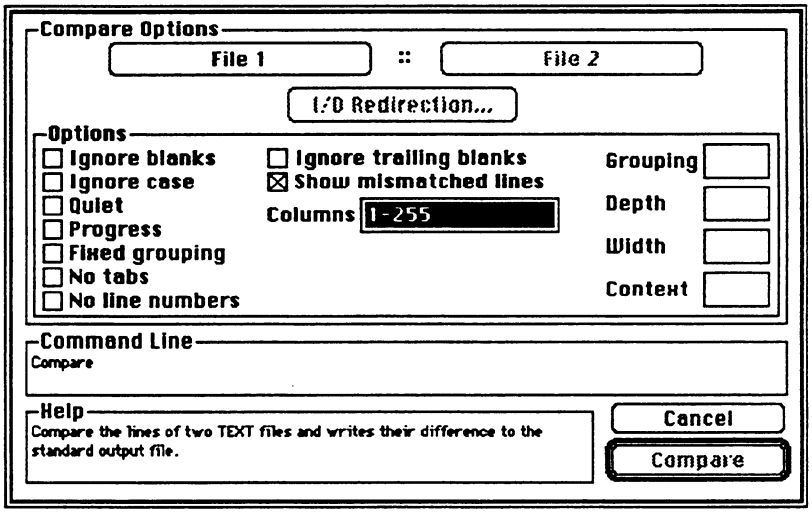
Parameter errors are written to diagnostic output.

Status

The following status codes may be returned to the Shell:

0 Files match.
1 Parameter or option error.
2 Files don't match.

Options **-b** Treat several blanks (spaces or tabs) as a single space, and ignore trailing blanks.



-c *col1-col2*[,*col1-col2*]

Compare only the columns *col1* to *col2* of each file. If the second column range is omitted, then the first range applies to both files; otherwise the first range applies to *file1* and the second range applies to *file2*. If *col1* is omitted, 1 is assumed. If *col2* is omitted, 255 is assumed.

- *Note:* To use the **-c** option, the tabs must be expanded. The tab setting is determined from the file's tab value. (See also the **-x** option below.)

-d *depth* Sets the maximum stack depth (size) for resynchronization—that is, how far out of synchronization the files can get before they're no longer worth comparing. *Depth* is an integer value from 1 to 1000. The default is 1000 if dynamic grouping is in use, and 25 for static (**-s**) grouping.

-e *context* Up to the specified number of context lines are displayed before and after the mismatched or extra lines. Values of 1 to 100 are allowed. Context lines are shown only if they are equal in both files, so fewer than the specified number of lines may be shown. Note that this option is ignored if the **-m** option is specified.

-g *groupingFactor*
: Specifies the grouping factor, *G*. For dynamic grouping, **-g** specifies the ***minimum*** grouping factor, that is, the minimum number of lines that must match for the two files to be considered resynchronized. (This value must be at least 2, which is the default.) If the **-s** (static) option is used, **-g** specifies a fixed grouping factor. (Values are from 1 through 1000; the default is 3.)

-h *width*
: Display mismatches in the horizontal format. Only a portion of each mismatched line is displayed side by side. *Width,* the total display line width, is a number from 70 to 255 that controls the total number of characters displayed in each of the two columns, or portions, of equal width.

-l
: Ignore case differences (convert all lines to lowercase before comparing them). The default is case sensitive.

-m
: Suppress the display of mismatched and extra lines. Only the mismatch messages and Shell editor commands to select the mismatches are displayed. The default is to display the mismatched and extra lines along with the messages. This option is ignored if the **-h** option is specified.

-n
: Do not write any messages to standard output if both files match.

-p
: Write Compare's version information to diagnostic output.

-s
: Static (fixed) grouping factor. The grouping factor is set with the **-g** option.

-t
: Ignore trailing blanks (spaces or tabs). This is a subset of the **-b** option.

-v
: Display differences between two files in a format that allows output lines to be cut and pasted into a source file.

-x
: Suppress tab expansion. Normally, tabs are expanded into spaces except when the **-b** option is used. The tab value is determined from the file's tab setting (a resource); if there is no setting, 4 is used.

▲ **Caution:** This option can cause stacked lines to be displayed incorrectly if the files contain tabs. Also, the **-c** option should not be used with **-x,** because **-c** depends on the true columns as displayed with tabs expanded. ▲

- *Note:* All comparison criteria that affect the individual lines *before* comparison—column range (**-c**), blanks compression (**-b**), and case conversion (**-l**)—are applied to those lines before they are stacked. Thus when the lines are displayed, they'll be shown in their modified form.

Examples

```
Compare File File.bak > Mismatches
```

Compares File and File.bak, writing the results to the file Mismatches. No options are specified, so dynamic grouping is used, blanks are retained, tabs are expanded into spaces, and matching is case sensitive.

```
Compare  File.old.§  File.new.§
```

Compares the selected portions of the two windows and writes out the results.

Limitations

Compare can handle text files with a maximum line length of 255 characters.

The text files compared should be fewer than 9999 lines long, because the displays are formatted based on four-digit line numbers.

See also

Equal command (Equal is a quicker command that tells you whether files are different, but stops at the first byte at which they differ).

CompareFiles—show file differences

Syntax CompareFiles [-9 | -13 | -b x y] *oldFile newFile*

Description CompareFiles compares two text files (using the tool Compare) and, if there are any differences, displays the file in adjacent windows for interactively viewing the differences. A menu will be appended to the menu bar to go through the changes.

When all the changes have been shown, the windows will be closed (if they were closed when CompareFiles started) and the menu will be deleted.

The Compare menu contains four items for viewing and editing the differences. The items perform the following actions:

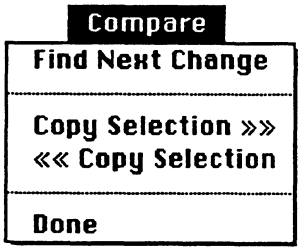
Find Next Change Finds the next difference and highlights the changes in each window. (Notice that the differences are shown from bottom to top. This is so editing changes will not affect the file offsets recorded from the Compare tool.)

Copy Selection » Replaces the changed text in the new file with the old text.

Copy Selection « Replaces the old text with the changed text from the new file.

Done Closes the files (asking if you want to save changes) and deletes the Compare menu. Use this item to close all the windows and delete the menu. (If you close any of the windows yourself, they will not be restored to their previous size and position.)

The figure below shows the CompareFiles menu.



To increase the speed of CompareFiles, there are a few restrictions: the options and parameters must be specified in the order indicated on the Syntax summary above, and the size of the rectangle specified with the **-b** option is not checked for accuracy. Remember, however, that since CompareFiles is a script, you may easily modify the behavior to fit your working style.

Type Script.

Input None.

Output None.

Diagnostics Errors from the script itself are written to standard output. Errors from running Compare are written to diagnostic output.

Status The following status codes may be returned:

0 The files match.
1 Syntax error.
2 The files differ.

Options The options specify the screen size to use for the tiling of the windows. The default screen size is 640 by 480.

-9 Tile the two open windows to fit on a 512 by 342 (9-inch) screen.

-13 Tile the open windows to fit on a 640 by 480 (13-inch) screen. This is the default screen size.

-b x y Tile the open windows to fit within the area specified by **x** and **y**.

Examples

```
CompareFiles Sample.old Sample.c
```

Compares the file Sample.c to Sample.old. If there are some differences, those two files are opened side by side on the screen.

```
CompareFiles -b 1024 1024 Sample.old Sample.c
```

Compares the file Sample.c to Sample.old. If there are differences, the files are opened and tiled into a 1024 by 1024 rectangle.

See Also Compare Tool.

CompareRevisions—compare revisions

Syntax CompareRevisions *file*

Description Compare the revision of the HFS file *file* with another revision of that same file.

CompareRevisions uses the ProjectInfo command to determine what project *file* belongs to and what its revision is. CompareRevisions then displays a list of the other revisions of the file for the user to choose. CompareRevisions checks this other revision out and calls the CompareFiles script to display both revisions on the screen and to highlight the differences between them. CompareFiles puts up an AddMenu named Compare to help you step through the differences between the two revisions.

The file must belong to a currently mounted project. If the project that the file belongs to is not currently mounted, CompareRevisions displays an Alert.

CompareRevisions uses the CompareFiles script.

Type Script.

Input None.

Output None.

Diagnostics Errors and warnings are written to diagnostic output.

Status The following status codes may be returned:

0 No Errors.
1 Syntax Error.
2 Error in Processing.
3 System Error.

Options None.

Examples

```
CompareRevisions file.c
```

This example compares the revision in HFS file "file.c" in the working directory with any other revisions of file.c in the project.

```
AddMenu Project 'Compare Revisions' 'CompareRevisions ∂
"{Active}" ΣΣ "{WorkSheet}"'
```

This example adds CompareRevisions to the Project menu and allows you to compare revisions by opening the file you wish to compare and then selecting the 'Compare Revisions' menu item in the Project menu.

See Also CompareFiles.

Confirm—display confirmation dialog box

Syntax

Confirm [-t] [*message...*]

Description

Confirm displays a confirmation dialog box with OK and Cancel buttons and the prompt *message*. There is no output to this command: the result of the dialog is returned in the {Status} variable.

Note: Because Confirm returns a nonzero status value to indicate that No or Cancel was selected, a script should set the Shell variable {Exit} to zero before executing the Confirm command. (This step is necessary because the Shell aborts script processing when a nonzero status value is returned and {Exit} is nonzero.)

Type

Built-in.

Input

Reads standard input for the message if no parameters are specified.

Output

None.

Diagnostics

Errors are written to diagnostic output.

Status

The Confirm command may return the following status codes:

0 The OK button was selected.
1 Syntax error.
4 The Cancel button was selected or the No button was clicked in a three-way dialog box.
5 The Cancel button was selected in a three-way dialog box; see the **-t** option.

- *Note:* In a two-button dialog box, "Cancel" means the same thing as "No"; OK means "Yes."

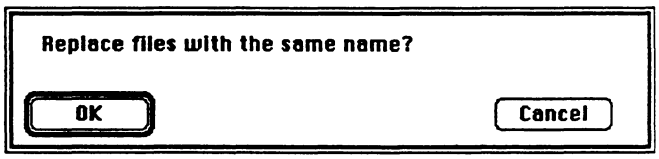
Option

-t Display a three-way confirmation dialog box, which includes Yes, No, and Cancel buttons. In this case, 4 means "No" and 5 means "Cancel."

Examples

```
Set Exit 0
Confirm "Replace files with the same name? "
If {Status} == 0
          Duplicate -y Source:≈ Destination:
End
Set Exit 1
```

The following confirmation dialog box will be displayed:



If you select the OK button, the Duplicate command will be executed.

The following script makes use of a three-way confirmation dialog box:

```
Set Exit 0
Set list ""
For file In `files -t TEXT`
    Confirm -t "Print file {file}?"
    Set SaveStatus {Status}
    Continue If {SaveStatus} == 4     # No
    Break If {SaveStatus} == 5        # Cancel
    Set list "{list} '{file}'"        # Yes
End
If "{list}" ! = ""
    Print {PrintOptions} {list}
End
Set Exit 1
```

This example prints selected TEXT files in the current directory. For each file, it displays a dialog box with three choices (Yes, No, and Cancel). Selecting "Yes" prints the file. If you select "No," the Continue command causes this file to be skipped, but processing continues with the next file in the list. If you select "Cancel," the Break command causes the For loop to be terminated, ending the question-and-answer session. The filenames are saved in the variable {list} and printed following the loop.

See also Alert and Request commands.

Continue—continue with next iteration of For or Loop

Syntax Continue [If *expression*]

Description If *expression* is nonzero, Continue terminates this iteration of the immediately enclosing For or Loop command and continues with the next iteration. (Null strings evaluate to zero.) If the "If *expression*" clause is omitted, the Continue is unconditional. If no further iterations are possible, the For or Loop is terminated. (For a definition of *expression,* see the Evaluate command.)

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status Continue may return the following status codes:

0	No errors.
–3	Error in parameters, or Continue not within For...End or Loop...End.
–5	Invalid expression.

Options None.

Example

```
Set Exit 0
Set list ""
For file In `files -t TEXT`
          Confirm -t "Print file {file}?"
          Set SaveStatus {Status}
          Continue If {SaveStatus} == 4     # No
          Break If {SaveStatus} == 5        # Cancel
          Set list "{list} '{file}'"        # YesEnd
End
Print {PrintOptions} {list}
Set Exit 1
```

In this example, the Continue command is executed if the user selects No (status value 4). The Continue causes the current file to be skipped, but processing continues with the next file in the list.

(For a full explanation of this example, refer to the Confirm command.)

See also

For, Loop, Break, and If commands.

Evaluate command, for a description of expressions.

"Structured Commands" in Chapter 5.

Copy—copy selection to Clipboard

Syntax Copy [-c *count*] *selection* [*window*]

Description Finds *selection* in the specified window and copies it to the Clipboard, replacing the previous contents of the Clipboard. If no window is specified, the command operates on the target window (the second window from the front). It's an error to specify a window that doesn't exist.

For a definition of *selection,* see "Selections" in Chapter 6; a summary of the selection syntax is contained in Appendix B.

◆ *Note:* To copy files, use the Duplicate command.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status Copy may return the following status codes:

0 At least one instance of the selection was found.
1 Syntax error.
2 Any other error.

Option **-c** *count* For a count of *n,* find and copy the *n*th instance of *selection.*

Examples

```
Copy §
```

Copies the current selection to the Clipboard. This command is like the Copy command in the Edit menu, except that the action takes place in the target window.

```
Copy /BEGIN/:/END/
```

Selects everything from the next `BEGIN` through the following `END` and copies this selection to the Clipboard.

See also

Cut and Paste commands.

"Selections" in Chapter 6 and Appendix B.

Count—count lines and characters

Syntax Count [-l] [-c] [*file...*]

Description Counts the lines and characters in its input and writes the results to standard output. If no files are specified, standard input is read. If more than one file is specified, separate counts are printed for each file, one per line and preceded by the filename. A total is printed following the list.

Type Tool.

Input Standard input is read if no files are specified on the command line.

Output Line and character counts are written to standard output.

Diagnostics Errors are written to diagnostic output.

Status Count may return the following status codes:

0 No errors.
1 Error in parameters.
2 Unable to open input file.

Options

-l Write only the line counts.

-c Write only the character counts.

Examples

```
Count MakeFile.c Count.c
```

Displays line counts and character counts in the form

```
MakeFile.c      43    981
Count.c   153   3327
Total     196   4303
```

```
Files | Count -l
```

Displays the total number of files and directories in the current directory.

```
Count -l §
```

Displays the number of lines selected in the target window.

- *Note:* The source code for Count is included in the CExamples folder in the file Count.c, as part of MPW C.

CPlus—C++ compiling system

Syntax CPlus [*option* ...] [*file*]

Description CPlus compiles the specified C++ source file. Compiling file *Name*.cp creates object file *Name*.cp.o. (By convention, C++ source filenames end in a ".cp" suffix.) If no filenames are specified, standard input is compiled and the object file "c.o" is created.

(Note that SADE object file information cannot be generated for standard input source files.)

The CPlus script activates, in turn, *CFront*, and the *MPW C Compiler*. (*CFront* consists of two components: a C preprocessor and a C++ to C translator.)

See the *MPW 3.0 C++ Reference Manual* for details of the MPW C++ language definition.

Type Script.

Input If no filenames are specified, standard input is compiled. You can terminate input by pressing Command-Enter.

Output If you specify the **-e** or **-e2** option, preprocessor output is written to standard output, and no object file is produced. If you specify the **-c** option, the C code produced by *Cfront* is written to standard output.

Diagnostics Errors and warnings are written to diagnostic output. If the **-p** option is specified, progress and summary information is also written to the diagnostic output.

Status The following status codes may be returned:

0 Successful completion.
1 Errors occurred.

Options

-b Generate PC-relative references for functions in the same segment and for string constants (which are kept at the end of the function code module). The default is to place string constants in the global data area, and to generate A5-relative (jump table) references for function addresses. Useful for writing DAs, WDEFs, and so on.

-b2 Provide the actions of option **-b**, but also allow the code generator to reduce code size by overlaying the storage constants where possible.

-b3 Cause the code generator to keep string constants within the code and overlay them when possible (but always generate A5-relative references for function addresses).

-c Do not generate object code. Write the intermediate C code to standard output. This option is useful for two purposes: to pipe the output of *Cfront* to a different C compiler, and to produce the C code for human inspection in order to clarify the semantics of a C++ construct.

-d *name* Define *name* to the preprocessor with value 1. This is the same as writing

```
#define name  1
```

at the beginning of the source file. (The **-d** option does not override `#define` statements in the source file.)

-d *name=string*

Define *name* to the preprocessor with value *string*. This is the same as writing

```
#define name  string
```

at the beginning of the source file.

-e Do not compile the program. Instead, write the output of the preprocessor to standard output. This option is useful for debugging preprocessor macros.

-e2 Implies **-e**, but also suppresses comments.

-elems881 Use in-line MC68881 instructions for all transcendental functions available on the MC68881 processor. See the *MPW 3.0 C Reference* for a complete list of these functions. This option implies the **-mc68881** option.

-f When *CFront* gets its source from standard input—for example, when the source is sent to a stand-alone preprocessor whose output is piped to *Cfront*—the option **-f** <*filename*> will cause correct line number information to be sent to debuggers.

-f2 Causes *CFront* to send a tokenized version of the intermediate code to the C compiler. This option is required if SADE offsets are needed.

-i *pathname*[,*pathname*]...

Search for `include` files in the specified directories. Multiple **-i** options may be specified. A maximum of 15 directories can be searched. The following is the search order:

1. The `include` filename is used as specified. If a *full pathname* is given, no other searching is applied. If the file wasn't found, and the pathname used to specify the file is a *partial pathname* (no colons in the name or a leading colon), then the following directories are searched:
2. The directory containing the current input file.
3. The directories specified in the **-i** options, in the order listed.
4. The directories specified in the Shell variable {Cincludes}.

-l Generate the *#line* pragma in the nonstandard form

 # <line_number> "<file>".

-m Generate 32-bit references for data. Required when there is more than 32K of global data.

-mgb off Do not include symbols for the MacsBug debugger.

-mbg full | on

Include full (untruncated) symbols for MacsBug.

-mbg ch8 Include V2.0-compatible MacsBug symbols (eight characters only, in a special format). This option is useful for generating symbols for the MacApp debugger.

-mbg *number*

Include MacsBug symbols truncated to length *number*.

-mc68020 Generate MC68020 instructions whenever doing so would provide faster and/or smaller object code.

-mc68881 Generate MC68881 instructions for all basic floating-point operations.

-mtbl0 Suppress output of method tables for each Object Pascal Class.

-mtbl1 Force output of method tables for each Object Pascal Class.

If neither **-mtbl0** nor **-mtbl1** is selected, the default is to emit method tables only for classes for which one or more virtual member functions are defined in the source file.

-n Turn pointer assignment incompatibility errors into warnings.

-o *objname* Pathname for the generated object file. If *objname* ends with a colon, it indicates a directory for the output file, whose name is then formed by the normal rules (that is, *inputfilename*.o). If *objname* does not end with a colon, the object file is written to the file *objname*.

-p Write progress information (include filenames, function names, and sizes) and summary information (number of errors and warnings, code size, global data size, and compilation time) to diagnostic output.

-s *name* Name the object code segment. (The default segment name is "Main".)

-sym on | full

Write complete object file records containing information for SADE, the MPW symbolic debugger. This option can be limited by also specifying one or more of **nolines**, **novars**, **notypes**, which cause omission of line, type, and variable information, respectively, from the object file (such as **-sym on**, **nolines**, **novars**). For more information, see the *MPW 3.0 C Reference Manual.*

-t Write C compilation time to diagnostic output.

-u *name* Undefine the predefined preprocessor symbol *name* . This is the same as writing

```
#undef name
```

at the beginning of the source file.

-vtbl0 Suppress output of virtual tables for each ordinary class with virtual functions.

-vtbl1 Force output of virtual tables for each ordinary class with virtual functions. If neither **-vtb10** nor **-vtb1** is selected, the default is to emit virtual tables only for classes for which one or more virtual member functions are defined in the source file.

-w Suppress compiler warning messages. (By default, warnings are written to diagnostic output.)

-w1 Cause *Cfront* to generate additional warnings.

-x *filename*

This option names the file containing a cross-compilation table. It is used when doing cross-development to a different processor.

-y *pathname*

Put the C compiler's temporary intermediate (".o.i") files in the directory specified by *pathname*.

-z0 Force "inline" functions to be non-inline.

-z6 Do not optimize enumerations. ENUM variables become the same as INT.

Example

```
cplus -p Sample.c
```

Compiles Sample.c, producing the object file Sample.c.o. Writes progress information to diagnostic output. (Sample.c is found in Examples:CPlusExamples.)

See Also *MPW 3.0 C Reference, MPW C++ Reference.*

CreateMake—create a simple makefile

Syntax

CreateMake [-Application [-c creator] | -Tool | -DA | -CR -m *entry point* -rt *resource type* [-c creator -t *file type*]] [-sym on] *program file...*

Description

CreateMake creates a simple makefile for building the specified program. The parameter *program* is the name of the program. Makefile *program*.make is created. The list of files includes both source and library files. Source files may be written in any combination of assembly language (suffix ".a"), C (".c"), C++ (".cp"), Pascal (".p"), and/or Rez (".r").

You can also specify Library files (suffix ".o"). Link the program with these files. CreateMake automatically links with the library files listed below. It is not necessary to specify these files as parameters to CreateMake.

You can create Makefiles for building applications (the default), desk accessories, and tools.

CreateMake generates commands that link the program with the following set of MPW libraries:

- *Inside Macintosh* Interfaces
  {Libraries}Interface.o
- Runtime support—one of the following:
  {Libraries}Stubs.o # a tool is to be built
  {Libraries}Runtime.o # no C object files
  {CLibraries}CRuntime.o # any C object files
- C Libraries—if any source is in C
  {CLibraries}StdCLib.o
  {CLibraries}CSANELib.o
  {CLibraries}Math.o
  {CLibraries}CInterface.o
- C Libraries—if any source is in C++
  {CLibraries}CPlusStreams.o # a tool is to be built
  {CLibraries}CPlusStubs.o # a DA is to be built
  {CLibraries}CSANELib.o
  {CLibraries}Math.o
  {CLibraries}CInterface.o

- Pascal Libraries—if any source is in Pascal
  {PLibraries}PasLib.o
  {PLibraries}SANELib.o
- For tools:
  {Libraries}ToolLibs.o
- For desk accessories:
  {Libraries}DRVRRuntime.o

CreateMake does not include dependencies on `include` files and `USES` files in the makefile. Libraries other than those listed above are not included in the Link command generated by CreateMake, unless specified as parameters. CreateMake is used to implement the Create Build Commands item in the Build menu.

Type Script.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

0 Successful completion.
1 Parameter or option error.

Options

-Application
Create build commands for building an application. This is the default.

-c *creator* If a code resource or an application, optionally provide the *creator.*

-CR Create build commands for building a stand-alone code resource.

-DA Create build commands for building a desk accessory.

-m *main entry point*
If a code resource, provide the *main entry point.*

-rt *resource type*
If a code resource, provide the *type* and *ID* in the form *type=ID.*

-T *file type* If a code resource, optionally provide the *file type.*

-tool Create build commands for building a tool.

-sym on Create build commands that construct an object containing symbolic debugger information for SADE.

Example

```
CreateMake -tool count count.c count.r
```

Creates the makefile Count.make containing commands for building the tool Count from the source files Count.c and Count.r. The makefile is similar to the following:

```
#    File:        count.make
#    Target:      count
#    Sources:     count.c count.r
#    Created:     Thursday, June 2, 1988 5:33:38 PM

count.c.o ƒ count.make count.c
    C count.c
count ƒƒ count.make count.r
    Rez count.r -append -o count

SOURCES = count.c count.r
OBJECTS = count.c.o

count ƒƒ count.make {OBJECTS}
    Link -w -t MPST -c 'MPS ' ∂
          "{Libraries}"Stubs.o ∂
          "{CLibraries}"CRuntime.o ∂
          "{Libraries}"Interface.o ∂
          "{CLibraries}"StdCLib.o ∂
          "{CLibraries}"CSANELib.o ∂
          "{CLibraries}"Math.o ∂
          "{CLibraries}"CInterface.o ∂
          "{Libraries}"ToolLibs.o ∂
          "{OBJECTS}" ∂
          -o count
```

See also BuildMenu and BuildProgram commands.

"Building a Program: An Introduction" in Chapter 2.

Cut—copy selection to Clipboard and delete it

Syntax Cut [-c *count*] *selection* [*window*]

Description Finds *selection* in the specified window, copies its contents to the Clipboard, and then deletes the selection. If no window is specified, the command operates on the target window (the second window from the front). It's an error to specify a window that doesn't exist.

For a definition of *selection,* see "Selections" in Chapter 6; a summary of the selection syntax is contained in Appendix B.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status Cut may return the following status codes:

0 At least one instance of the selection was found.
1 Syntax error.
2 Any other error.

Option **-c** *count* Find and cut *count* instances of *selection.*

Examples

```
Cut §
```

Cuts the current selection in the target window. (This is the same as the Cut menu item, except that it operates on the target window rather than the active window.)

```
Cut  /BEGIN/:/END/
```

Selects everything from the next BEGIN through the following END, copies the contents of the selection to the Clipboard, and then deletes the selection.

See also Clear, Copy, and Paste commands.

"Selections" in Chapter 6 and in Appendix B.

Date—write the date and time

Syntax Date [[-a | -s][-d | -t][-c *num*]] | [-n]

Description Writes the current date and time to standard output in a variety of standard and user-specified formats. Date arithmetic is supported with the **-n** and **-c** options that work with the number of seconds since January 1, 1904. With no options the Date output has this form: Thursday, August 30, 1988 10:45:51 A.M.

Type Built-in.

Input None.

Output The date is written to standard output.

Diagnostics Errors are written to diagnostic output.

Status Date may return the following status codes:

0 No error.
1 Syntax error.

Options

-a Abbreviated date. Three-character abbreviations are used for the month and day of the week. For example, `Thu, Aug 29, 1988.`

-c *num* Write the date corresponding to *num*, which is interpreted as the number of seconds since midnight, January 1, 1904. You can use the other output format options with **-c** to specify the output format.

-d Write the date only.

-n Return a numeric value for the current date and time, in terms of the number of seconds since midnight, January 1, 1904. This option is useful for date and time arithmetic.

-s Short date form. Numeric values are used for the date. The day of the week is not given. For example, `8/30/88 10:45:51`

-t Write the time only.

Examples

```
Date
```

returns the date in the form

```
Friday, February 14, 1988 10:34:25 PM
```

```
Date -a
```

returns

```
Fri, Feb 14, 1988 10:34:25 PM
```

```
Date -s -d
```

returns

```
2/14/86
```

```
Set starttime `Date -n`
BuildMyProgram
Set endTime `Date -n`
Echo Total time for BuildMyProgram ∂
`Evaluate {endTime} - {startTime}`
```

This example demonstrates how date arithmetic may be used to show how long a tool or script takes to execute.

Delete—delete files and directories

Syntax Delete [-y | -n | -c] [-i] [-p] *name...*

Description Deletes file or directory *name*. If *name* is a directory, *name* and its contents (including all subdirectories) are deleted.

Before deleting directories, a dialog box will request confirmation for the deletion. Use the **-y**, **-n**, or **-c** options in scripts to avoid this interaction. Be sure to see the warning at the end of this section.

Type Built-in.

Input None.

Output None.

Diagnostics Errors and warnings are written to diagnostic output. Progress and summary information is also written to diagnostic output if the **-p** option is specified.

Status The following status codes may be returned:

0 All specified objects were deleted (except for any directories skipped with the **-n** option).
1 Syntax error.
2 An error occurred during the delete.
4 Cancel was selected or implied by the **-c** option.

Options

-i Ignore errors (that is, do not print messages, and return a status code of 0).

-n Answer "No" to any confirmation dialog that may occur, skipping the delete for any directories encountered.

-p List progress information as the delete takes place.

-y Answer "Yes" to any confirmation dialog that may occur, causing any directory encountered to be deleted.

-c Answer "Cancel" to any confirmation dialog that may occur, causing the delete to stop when a directory is encountered.

Example

```
Delete HD:MPW:≈.c
```

Deletes *all* items in the MPW folder that end in ".c". (Recall that the Shell first replaces the parameter "≈.c" with a list of filenames matching the pattern—the Delete command then deletes each of these files.)

Warning

Beware of potentially disastrous typographical mistakes such as the following:

```
Delete ≈ .c
```

Note the space after "≈"—this space causes "≈" and ".c" to be treated as two separate parameters. In this case, Delete deletes *all files* in the current directory and also attempts to delete a file named ".c".

Also note that the following command deletes *everything:*

```
Delete ≈:
```

That is, the filename pattern ≈: expands to the names of *all volumes online* (including the startup volume!).

When deleting files *en masse,* it's a good practice to use the Echo command to verify the action of the filename generation operators; for example,

```
Echo ≈.c
```

See also

Clear command (for deleting selections).

"Filename Generation" in Chapter 5.

DeleteMenu—delete user-defined menus and items

Syntax DeleteMenu [*menuName* [*itemName*]]

Description Deletes the user-defined item *itemName* in the menu *menuName*. If *itemName* is omitted, all user-defined items for *menuName* are deleted.

▲ **Caution** If *itemName* and *menuName* are both omitted, all user-defined items are deleted. Menu items that haven't been added with AddMenu can't be deleted with DeleteMenu. ▲

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status DeleteMenu may return the following status codes:

0 No errors.
1 Syntax error.
2 Other errors.

Options None.

Example

```
DeleteMenu File
```

Deletes all user-defined items from the File menu.

See also AddMenu command.

DeleteNames—delete symbolic names

Syntax DeleteNames [-u *user*] [-project *project*] [-public] [-r] [*names*... | -a]

Description Delete symbolic *names* used to represent a set of revisions under Projector. You can create symbolic *names* by using the NameRevisions command.

You can use the **-log** option of the ProjectInfo command to see which names have been deleted and what their values were.

See Chapter 7 for complete definitions of the terms and symbols used in Projector commands.

Type Built-in.

Input None.

Output None.

Diagnostics Errors and warnings are written to diagnostic output.

Status The following status codes may be returned:

0 No errors.
1 Syntax error.
2 Error in processing.

Options

-u *user* Name of the current user. This overrides the {User} Shell variable.

-project *project*
Name of the project which contains the files. This becomes the current project for this command.

-public Delete public Names.

-a Delete all names in the project.

-r Recursively execute the DeleteNames command on the current project and all its subprojects.

Examples Suppose you have created a Name "Work" that is expanded to the files file.c and interactive.c using the command

```
NameRevisions Work file.c interactive.c
```

Then :

```
DeleteNames Work
```

removes "Work" from the list of symbolic names.

See Also NameRevisions, ProjectInfo.

DeleteRevisions—delete revisions and branches

Syntax

DeleteRevisions [-u *user*] [-project *project*] [-file] [-y] *revision...*

Description

Delete old revisions by specifying the oldest revision that you want to keep. All prior revisions are deleted. Delete all revisions on a branch by naming the branch or branches in the named files under Projector. It is an error to try to delete a revision that is currently checked out for modification.

Revision is either a filename, a filename followed by a comma and a revision number, or a filename followed by a comma and a branch name (such as foo.c,22a).

You can use the **-file** option to remove the file and all of its revisions from the project.

▲ **Warning** DeleteRevisions permanently removes the revisions and branches specified. They cannot be recovered. ▲

You can use the **-log** option of the ProjectInfo command to see which revisions have been deleted and who deleted them.

See Chapter 7 for complete definitions of the terms and symbols used in Projector commands.

Type

Built-in.

Input

None.

Output

None.

Diagnostics

Errors and warnings are written to diagnostic output.

Status

The following status codes may be returned:

0 No errors.
1 Syntax error.
2 Error in processing.
3 System error.

Options

-u *user* Name of the current user. This overrides the {User} Shell variable.

-project *project*
Name of the project that contains the files. This option becomes the current project for this command.

-file Deletes the file and all its revisions.

-y Deletes the file/revision (avoids dialogs).

Examples

```
DeleteRevisions -project Zoom∫Utilities∫MyProject file.c
```

This example deletes all revisions except the latest in file.c in the named project.

```
DeleteRevisions file.c,22a3
```

This example deletes all revisions on branch 22a before revision 3 of file.c.

```
DeleteRevisions file.c,22a
```

This command deletes all the revisions on branch 22a in file.c of the current project.

```
DeleteRevisions -file file.c
```

This command deletes the file file.c and all of its revisions from the current project.

See Also NameRevisions, ProjectInfo.

DeRez—Resource decompiler

Syntax

DeRez [*option...*] *resourceFile* [*resourceDescriptionFile...*]

Description

Creates a text representation (resource description) of the resource fork of *resourceFile*, according to the resource type declarations in the resource description file(s). The resource description is written to standard output.

A **resource description file** is a file of type declarations in the format used by the resource compiler, Rez. The type declarations for standard Macintosh resources are contained in the files Types.r and SysTypes.r, contained in the {RIncludes} folder. If no resource description file is specified, the output consists of `data` statements giving the resource data in hexadecimal form, without any additional format information.

If the output of DeRez is used as input to Rez, with the same resource description files, it produces the same resource fork that was originally input to DeRez. DeRez is not guaranteed to be able to run a declaration backwards; if it can't, it produces a `data` statement instead of the appropriate `resource` statement.

DeRez ignores all `include` (but not `#include`), `read`, `data`, `change`, `delete`, and `resource` statements found in the *resourceDescriptionFile*. (It still parses these statements for correct syntax.)

For the format of resource type declarations, see Chapter 11 and Appendix D.

Type

Tool.

Input

Standard input is never read. DeRez requires a resource file as input. You may give optional formatting information by specifying one or more resource description files.

For all resource description files on the command line, the following search rules are applied:

1. DeRez tries to open the file with the name specified "as is."
2. If rule 1 fails and the filename contains no colons or begins with a colon, DeRez appends the filename to each of the pathnames specified by the {RIncludes} variable and tries to open the file.

Output

A resource description is written to standard output. The resource description consists of `resource` and `data` statements that can be understood by Rez. (See Chapter 11.)

Diagnostics

If no errors or warnings are detected, DeRez runs silently. Errors and warnings are written to diagnostic output.

Status

DeRez may return the following status codes:

0 No errors.
1 Error in parameters.
2 Syntax error in file.
3 I/O or program error.

Options

-c[ompatible]

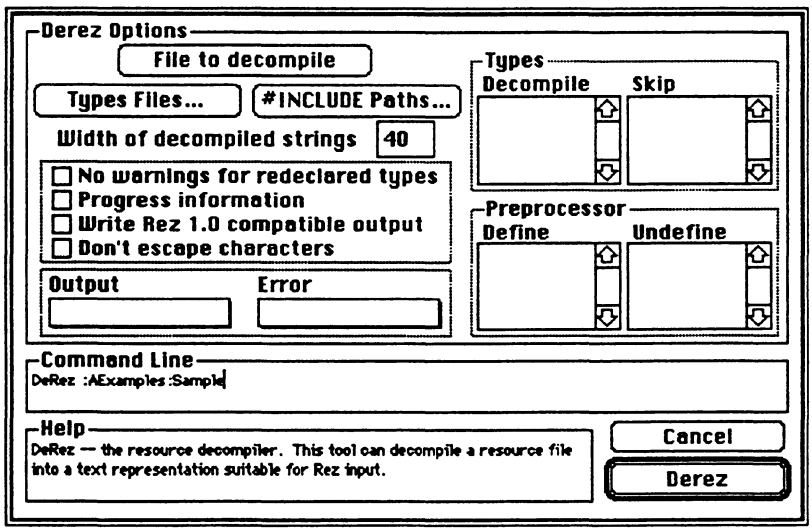
Generate output that is backward compatible with Rez 1.0.

-d[efine] *macro*[=*data*]

Define the macro variable *macro* to have the value *data*. If *data* is omitted, *macro* is set to the null string—note that this still means that *macro* is defined. Using the **-d** option is the same as writing

`#define` *macro*[*data*]

at the beginning of the input. The **-d** option may be repeated any number of times.



-e[scape] When this option is specified, characters that are normally escaped (such as \0xff) are no longer escaped. Instead they are printed as extended Macintosh characters. (*Note:* Not all fonts have all the characters defined.) Normally characters with values between $20 and $D8 are printed as Macintosh characters. With this option, however, all characters (except null, newline, tab, backspace, form feed, vertical tab, and rubout) are printed as characters, not as escape sequences.

-i Lets you specify one or more pathnames to search for `#include` files. This option may be specified more than once. The paths will be searched in the order they appear on the command line.

```
derez -i {mpw}myStuff:  -i hd:tools...
```

-m[axstringsize] *n*

Set the maximum string size to *n; n* must be in the range 2–120. This setting controls string width in the output.

-only *typeExpr* [(*ID1* [:*ID2*]) | *resourceName*]

Read only resources of resource type *typeExpr*. If an ID, range of IDs, or resource name is given, read only those resources for the given type. This option may be repeated.

- *Note: typeExpr* is an expression, so literal quotation marks (') might be needed. If an ID, range of IDs, or name is given, the entire option parameter must be quoted; for example,

```
DeRez -only "'MENU'(1:128)" ...
```

See also the "Examples" section below.

- *Note:* The **-only** option cannot be specified together with the **-skip** option.

-only *type* A simpler version of the above option: no quotation marks are needed to specify a literal type as long as it starts with a letter. Items such as escape characters are not allowed. For example,

```
DeRez -only MENU ...
```

-p Display progress and version information.

-rd Suppress warning messages if a resource type is redeclared.

-s[kip] *typeExpr* [(*ID1* [: *ID2*]) | *resourceName*]
Skip resources of type *typeExpr* in the resource file. For example, it's very useful to be able to skip 'CODE' resources. *typeExpr* is an expression; see the note under **-only.** The **-s** option may be repeated any number of times.

-s[kip] *type* A simpler version of the **-s** option; no quotation marks are needed to specify a literal as long as it starts with a letter.

-u[ndef] *macro*
Undefine the macro variable *macro.* This is the same as writing

```
#undef macro
```

at the beginning of the input file. It is meaningful to undefine only the preset macro variables. This option may be repeated.

Examples

```
DeRez "{ShellDirectory}MPW Shell" -only MENU  Types.r
```

Displays all of the 'MENU' resources used by the MPW Shell. The type definition for 'MENU' resources is found in the file Types.r.

```
DeRez HD:OS:System SysTypes.r ∂
   -only "'DRVR' (∂"\0x00Scrapbook∂")"
```

Decompiles the Scrapbook desk accessory in the copy of the System file that's located in directory HD:OS:. (The type definition for 'DRVR' resources is found in the file SysTypes.r.)

See also

Rez and RezDet commands.

Chapter 11.

Type declaration files in RIncludes folder:

- Types.r
- SysTypes.r
- MPWTypes.r
- Pict.r

Directory—set or write the default directory

Syntax

Directory [-q | *directory*]

Description

If specified, *directory* becomes the new default directory. Otherwise the pathname of the current default directory is written to standard output.

If *directory* is a leafname, the command searches for *directory* in the directories listed in the Shell variable {DirectoryPath}. If the variable is undefined, the command looks in the current directory.

- *Note:* To display a directory's contents, use the Files command.

Type

Built-in.

Input

None.

Output

If no directory is specified, the default directory pathname is written to standard output.

Diagnostics

Errors are written to diagnostic output.

Status

Directory may return the following status codes:

0 No error.
1 Directory not found, command aborted, or parameter error.

Option

-q Don't quote the pathname that is written to standard output. Normally, a directory name is quoted if it contains spaces or other special characters.

Examples

```
Directory
```

Writes the pathname of the current directory to standard output.

```
Directory HD:MPW:Examples:
```

Sets the default directory to the folder Examples in the folder MPW on the volume HD. The final colon is optional.

```
Directory Reports:
```

Sets the default directory to the volume Reports. Note that volume names must end in a colon.

```
Directory :Include:Pascal:
```

Sets the default directory to the folder Pascal in the folder Include in the current default directory.

```
Set DirectoryPath ":, {MPW}, {MPW}Projects:"
Directory Tools
```

Sets the directory to the Tools directory. The current directory is searched first, followed by the {MPW} directory, and finally by the {MPW} Projects directory. If there is no Tools directory in your current directory, the directory is set to {MPW}Tools.

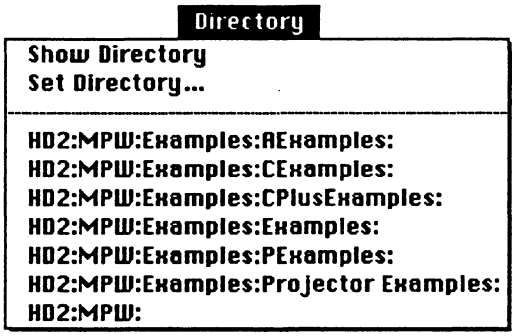
See also "File and Window Names" in Chapter 4.

Files, NewFolder, and SetDirectory commands.

DirectoryMenu—create the Directory menu

Syntax DirectoryMenu [*directory...*]

Description Creates the Directory menu shown here. The optional *directory...* parameter specifies the initial list of directories that appears in the menu. The menu items are described in Chapter 3.



The lower section of the Directory menu contains a list of directories. Initially this list consists of the parameters to DirectoryMenu. As other directories become the current directory (using the Set Directory menu item or the SetDirectory command), they are added to the list.

Type Script.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status Status code 0 (no problem) is always returned.

Options None.

Example

```
DirectoryMenu `(Files -d -i "{MPW}"Examples:≈) ≥ Dev:Null`∂
`Directory`
```

Creates the Directory menu. Directories in directory "{MPW}" that match the pattern Examples:≈ will be included in the Directory menu, along with the current directory.

This DirectoryMenu command should be included in your UserStartup file to install the Directory menu. You might replace the Examples directories and the default directory with your favorite list of directories.

DoIt—highlight and execute a series of commands

Syntax

DoIt (*CommandFile* [-echo] [-dump]) | -selection

Description

DoIt will execute a series of Shell commands, highlighting each command as it is executed. The commands can be either in a file or in the current selection of the active window. If a CommandFile is passed to DoIt, the file is opened (as the active window) and each command is executed. The window is closed when all commands have been processed.

This command will not work for a series of commands that contains structured commands such as If statements or Loops.

Type

Script.

Input

None.

Output

Errors produced by the DoIt script are sent to standard output. If the **-echo** option is specified, the commands are echoed to the WorkSheet as they are executed.

Diagnostics

Errors and warnings generated by the commands being executed by the DoIt script are written to diagnostic output.

Status

DoIt may return these status codes:

0 No errors.
1 Syntax error.
n Any status code returned by a command being executed by DoIt.

Options

-echo Each command is echoed to the WorkSheet before execution.

-dump If an error occurs in one of the commands being executed, all the remaining commands (including the command that failed) are written to the WorkSheet and marked with a marker called "ToDo."

-selection Execute the commands in the current selection of the active window.

Examples

```
Backup -from "HD:Src:" -to "Backup:Src" -a -r -c > out
DoIt out
```

The above command will highlight and execute all the Duplicate commands generated by the Backup command. In this way you can see progress as the files are being duplicated.

```
AddMenu DoIt "DoIt Selection" "DoIt -selection"
```

The above AddMenu command will create a menu that can be used to highlight and execute the current selection. This could be used on a series of commands generated by Make or Backup that were written to the Active window. Simply select the commands and select the "DoIt Selection" menu item.

```
Make > make.out
DoIt -dump make.out
```

This DoIt command will open the make.out file and highlight and execute each of the commands generated by the previous make command. In this way you can see progress as the files are being compiled and linked. If an error occurs (for instance, in one of the compiles), that compile command along with the rest of the commands in the make.out will be written to the WorkSheet. At this point you could fix the error (in the source file), select the "ToDo" marker (which would select the remaining commands), and select the "DoIt Selection" menu item to execute the remaining commands.

DumpCode—write formatted resources

Syntax

DumpCode [*option...*] *resourceFile*

Description

Disassembles object code that is stored in resources such as 'CODE', 'DRVR', and 'PDEF'. DumpCode reads from the resource fork of the specified file and writes the formatted assembly code to standard output. The default formatting convention is to disassemble the code and to display the corresponding bytes in hexadecimal and ASCII.

The default behavior of DumpCode is to dump all the 'CODE' resources from a program file. The **-rt** option can be used to dump resources of other types, such as drivers and desk accessories.

Some conventions about executable code resources are built into DumpCode and affect the formatted output in special ways:

- 'CODE' resources with ID 0 are formatted as a jump table (unloaded format).
- Other 'CODE' resources have information about jump table entries in the first four bytes.
- 'DRVR' resources have a special format at the beginning of the resource.

In addition, you can direct DumpCode to give a symbolic dump of data initialization descriptors and initial values.

Type

Tool.

Input

None.

Output

DumpCode writes formatted resources to standard output.

Diagnostics

Errors and warnings are written to diagnostic output. Progress information can also be written to diagnostic output (with the **-p** option).

Status

DumpCode may return the following status codes:

0 No problem.
1 Syntax error.
2 Fatal error.

Options

Note: Numeric values for options can be specified as decimal constants, or as hex constants preceded by a "$".

-d Suppress the disassembly and dumping of code. (The default is to disassemble the code.)

This option is useful in producing a small output file and looking at just the resource names, sizes, and resource header information. It is also useful when just some specialized information is desired, such as the jump table or data descriptors.

-di Suppress display of data initialization code.

-h Suppress the writing of header information, such as resource relative locations, hexadecimal and ASCII equivalents, and so on. The default is to produce this type of header information.

This option is useful in producing output that can be edited and submitted to the assembler for reassembly.

-jt Suppress formatting of the jump table. Only summary information for the jump table is given. (The default is to format the jump table unless one of the options **-s**, **-rt**, **-n**, or **-jt** is specified.)

-n Write only the resource names associated with resources. This option is useful for finding segments or desk accessories by name.

-p Write progress information (filenames, resource names, IDs, and sizes) to diagnostic output.

-r *byte1*[,*byteN*]

Limit the disassembly of code to the range *byte1...byteN.* The default is to disassemble all bytes in a segment. If *byteN* is omitted, the rest of the segment is disassembled. This does not affect disassembly alignment; the disassembly still starts at the base of the resource, but instructions are printed only for the specified range.

-rt *type*[=*ID*]

Dump only the single resource with type *type* and ID number *ID.* If *ID* is omitted, all resources of the specified type are dumped.

-s *resourceName*

Dump only the single resource named *resourceName.*

Example

```
DumpCode Sample > SampleDump
```

Formats the 'CODE' resources in the file Sample, writing the output to the file SampleDump. The output has this format:

```
File: sample, Resource 3, Type: CODE, Name: _DataInit
Offset of first jump table entry: $00000018
Segment is $000000D2 bytes long,  and uses 1 jump table entry
000000: 48E7 FFF0            'H...'      MOVEM.L     D0-D7/A0-A3,-(
000004: 4247                 'BG'        CLR.W       D7
000006: 4EAD 0032            'N..2'      JSR         $0032(A5)
00000A: 2218                 '".'        MOVE.L      (A0)+,D1
etc.
```

See also

DumpObj command.

"The Jump Table" in the chapter "Segment Loader" of *Inside Macintosh*, for a description of the jump table.

Appendix H, "Object File Format."

DumpFile—display contents of an arbitrary file

Syntax DumpFile [*option...*] *filename*

Description DumpFile lets you display the contents of the resource fork or data fork of a file in a variety of formats.

Type Tool.

Input DumpFile does not read standard input.

Output DumpFile writes formatted object file records and disassembled code to standard output.

Diagnostics Errors and warnings are written to diagnostic output. Progress information is also written to diagnostic output with the **-p** option.

Status DumpFile may return the following status codes:

0 No problem.
1 Syntax error.
2 Fatal error.

Options

-rf Display the resource fork of the file. (Default is data fork.)

-bf Display both forks of file.

-a Suppress display of ASCII character values.

-h Suppress display of hexadecimal characters.

-o Suppress display of file offsets.

-w *mm* Display width of *mm* bytes on each line of output. (Default is 16.)

- *Note:* The *mm* value in option **-w** must be a multiple of the *nn* value in **-g**.

-g *nn* Group *nn* bytes together without intervening spaces. (The default is 1.)

-p Write progress information (such as the name of the file being dumped and the version of DumpFile) to diagnostic output.

-r *byte1*[,*byteN*]
Display only the byte range *byte1* to *byteN*.

Examples

```
DumpFile -p ATestFile
```

Formats the data fork of the file *ATestFile* and writes its contents to standard output. This output has the following format:

```
DumpFile -p ATestFile
MPW File Display Utility Version 3.0B1 Release April 15,
1988   Start: 1:24:09 PM 4/19/88

 Copyright Apple Computer, Inc. 1985-1988
 All Rights Reserved.

File : ATestFile
Data Fork Length     : 20
Resource Fork Length : 382
Dumping Data Fork from offset 0 to 20

   0: 54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 66 This.is.a.test.file.

  10: 69 6C 65 2E
DumpFile completed normally

Execution required 0 seconds.
```

```
DumpFile -w 12 -g 4 ATestFile
```

Formats the data fork of the file *ATestFile* and writes its contents to standard output, grouping four bytes at a time and displaying 12 bytes per line. This output has the following format:

```
File : ATestFile
Data Fork Length     : 20
Resource Fork Length : 382
Dumping Data Fork from offset 0 to 20

   0: 54686973 20697320 61207465 This.is.a.te
   C: 73742066 696C652E          st.file.
```

```
DumpFile -rf -r 0,30 -g 4 ATestFile
```

Formats the resource fork of the file *ATestFile* and writes the contents of bytes 0 through 30 to standard output in 4-byte groups. This output has the following format:

```
File : ATestFile
Data Fork Length     : 20
Resource Fork Length : 382
Dumping Resource Fork from offset 0 to 30

   0: 00000100 0000014C 0000004C 00000032 .......L...L...2
  10: 696C652E 6F727920 2227227B 646972   ile.ory."'"{dir
```

DumpObj—write formatted object file

Syntax DumpObj [*option...*] *objectFile*

Description Disassembles object code that is stored in the data fork of an object file. By convention, object files end in the suffix ".o". In addition, the object file must have type `'OBJ'`.

Type Tool.

Input DumpObj does not read standard input.

Output DumpObj writes formatted object file records and disassembled code to standard output.

Diagnostics Errors and warnings are written to diagnostic output. Progress information is also written to diagnostic output with the **-p** option.

Status DumpObj may return the following status codes:

0 No problem.
1 Syntax error.
2 Fatal error.

Options

-d Suppress disassembly of code and display of data. The default is to disassemble code and to display data in hexadecimal and ASCII.

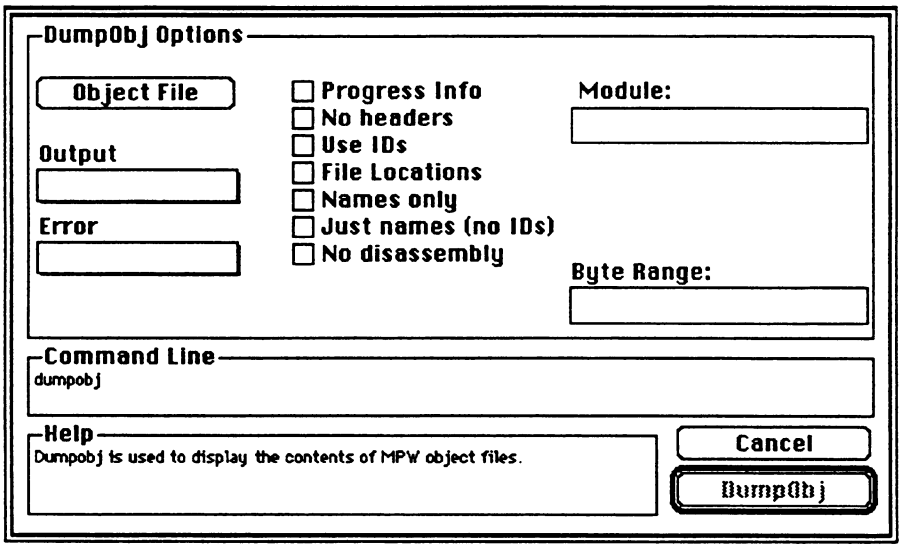
-h Suppress printing of header information on code lines. Header information includes the offset of the code and the code bytes in hex and ASCII. The default is to print header information.

Use this option to produce code that can be edited and submitted to the assembler for reassembly.

-jn Print just names for IDs, omitting the ID numbers. This option is useful for comparing object files that have identical names but different IDs.

-i Suppress substitution of names for IDs. The default is to preread the entire file, process the Dictionary records, and then show names in place of ID numbers.

This option is useful in examining an object file up to the point where an object file format error has been reported by Link or Lib; that is, it suppresses the preread, which is also likely to fail.



-l Print file locations of object records. The default is not to print these locations.

This option is useful in debugging compilers and assemblers, particularly when debugging code used to generate Pad records to assure alignment. (See Appendix H.)

-m *name* Dump a particular module. If *name* is an entry point, the module containing *name* is dumped. Other options that control format still have an effect.

Note: *name* is case sensitive, as are all object file identifiers.

-n Print names only. When this option is specified, only the **-p** option has an effect.

This option is useful in determining which names exist in an object file, particularly when there appears to be a discrepancy in spelling, capitalization, or length of identifiers.

-p Write progress information (such as the name of the file being dumped and the version of DumpObj) to diagnostic output.

-r *byte1*[,*byteN*]
Limit the disassembly of code to the range *byte1...byteN*. The default is to disassemble all bytes in a module. If *byteN* is omitted, the rest of the module is disassembled. This does not affect disassembly alignment; the disassembly still starts at the base of each contents record, but instructions are printed only for the specified range.

-sym [on | off
Enable or disable writing symbolic records to support SADE. The default is ON.

,nolines Omit line information.

,nolabels Omit label information.

,novars Omit variable information.

,notypes] Omit type information.

Example

```
DumpObj Sample.p.o >SampleDump
```

Formats the file Sample.p.o and writes its contents to the file SampleDump. This output has the following format:

```
Dump of file sample.p.o
First:      Kind 0 Version 1
Dictionary: FirstId 2
      2: Main
Pad
Module:         Flags $00 ModuleId  1 SegmentId Main
Content:        Flags $00
Contents offset 0000 size 006A
000000: 4E56 FFFE           'NV..'      LINK  A6,#$FFFE
000004: 2F07                '/.'        MOVE.L      D7,-(A7)
000006: 42A7                'B.'        CLR.L       -(A7)
000008: 3F3C 0080           '?<..'      MOVE.W      #$0080,-(A7)
etc.
```

For more information, see Appendix H.

See also DumpCode command.

Appendix H, "Object File Format."

Duplicate—duplicate files and directories

Syntax Duplicate [-y | -n | -c] [-d | -r] [-p] *name... targetName*

Description Duplicates *name* to *targetName*. (*Name* and *targetName* are file or directory names.) If *targetName* is a file or doesn't exist, the file or directory *name* is duplicated and named *targetName*. If *targetName* is a directory, the objects named are duplicated into that directory. (If more than one *name* is present, *targetName* must be a directory.) Created objects are given the same creation and modification dates as their source.

If a directory is duplicated, its contents (including all subdirectories) are also duplicated. No directory duplicated can be a parent of *targetName*.

Name can also be a volume; if *targetName* is a directory, *name* is copied into *targetName*.

A dialog box requests a confirmation if the duplication would overwrite an existing file or folder. You can use the **-y**, **-n**, or **-c** option in scripts to avoid this interaction.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output. Progress and summary information is written to diagnostic output if the **-p** option is specified.

Status The following status codes may be returned:

0 All objects were duplicated.
1 Syntax error.
2 An error occurred.
4 Cancel was selected or implied from the **-c** option.

Options

-y	Answer "Yes" to any confirmation dialog that occurs, causing conflicting files or folders to be overwritten.
-n	Answer "No" to any confirmation dialog that occurs, skipping files or folders that already exist.
-c	Answer "Cancel" to any confirmation dialog that occurs, causing the duplication to stop when a name conflict is encountered.
-d	Duplicate the data fork only. If *targetName* is an existing file, its data fork is overwritten and its resource fork remains untouched.
-r	Duplicate the resource fork only. If *targetName* is an existing file, its resource fork is overwritten and its data fork remains untouched.
-p	List progress information.

Examples

```
Duplicate Aug86 "Monthly Reports"
```

Assuming "Monthly Reports" is an existing directory, duplicates the file Aug86 into that directory.

```
Duplicate File1 Folder1 "Backup Disk:"
```

Duplicates File1 and Folder1 (including its contents) onto Backup Disk.

```
Duplicate -y File1 File2
```

Duplicates File1 to File2, overwriting File2 if it exists.

```
Duplicate Disk1:≈  HD:Files:
```

Duplicates all of the files on Disk1 into the directory HD:Files.

```
Duplicate Disk1:  HD:Files:
```

Duplicates all of Disk1 (as a directory) into HD:Files.

Limitation

Duplicate doesn't recognize folders on non-HFS disks.

See also

Move and Rename commands.

"File and Window Names" in Chapter 4.

"Filename Generation" in Chapter 5.

Echo—echo parameters

Syntax Echo [-n] [*parameters...*]

Description Writes its parameters, separated by spaces and terminated by a return, to standard output. If no parameters are specified, only a return is written.

Echo is especially useful for checking the results of variable substitution, command substitution, and filename generation.

Type Built-in.

Input None.

Output Parameters are written to standard output.

Diagnostics None.

Status Status code 0 is always returned.

Option

-n Don't write a return following Echo's parameters (that is, the insertion point remains at the end of the output line). The **-n** isn't echoed.

Examples

```
Echo "Use Echo to write progress info from scripts."
Use Echo to write progress info from scripts.
```

The Echo command above writes the second line to standard output.

```
Echo {Status}
```

Writes the current value of the {Status} variable—that is, the status of the last command executed.

```
Echo ≈.a
```

Echoes the names of all files in the current directory that end with ".a". (This might be useful as a precaution before executing another command with the argument "≈.a".)

```
Echo -n > EmptyFile
```

If EmptyFile exists, this command deletes its contents; if the file doesn't exist, it is created.

See also Parameters and Quote commands.

Eject—eject volumes

Syntax Eject [-m] *volume...*

Description Flushes the volume, unmounts it, and then ejects it, if it is a 3.5-inch disk. A volume name must end with a colon (:). If *volume* is a number without a colon, it's interpreted as a drive number.

- *Note:* If you unmount the current volume (the volume containing the current directory), the boot volume becomes the current volume. You can keep the volume mounted with the **-m** option. (See the chapter "File Manager" of *Inside Macintosh.*)

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

0 The disk was successfully ejected.
1 Syntax error.
2 An error occurred.

Option **-m** Leave the volume mounted.

Examples

```
Eject Memos:
```

Ejects (and unmounts) the disk titled Memos.

```
Eject 1
```

Ejects and unmounts the disk in drive 1 (the internal drive).

See also Mount, Unmount, and Volumes commands.

Entab—convert runs of spaces to tabs

Syntax Entab [*option...*] [*file...*]

Description Copies the specified text files to standard output, replacing runs of spaces with tabs. The default behavior of Entab is to do the following:

1. Detab the input file, using the file's tab setting (a resource saved with the file by the Shell editor), or 4 if there is none. You can override this "detab" value with the **-d** option.
2. Entab the file, setting tab stops every 4 spaces. You can specify another tab setting with the **-t** option. The entabbed output file looks the same as the original file(s), but contains fewer characters.

Options are also provided for controlling the processing of blanks between quoted strings.

Type Tool.

Input If no filenames are specified, standard input is processed.

Output All files are written to standard output.

Diagnostics Parameter errors and progress information (with the **-p** option) are written to diagnostic output.

Status The following status codes may be returned to the Shell:

0 Normal termination.
1 Parameter or option error.

Options

-a Minimum run of blanks that can be replaced with a tab. The default is 1.

-d *tabSetting* Override the input file's default tab setting with *tabSetting*. This option is useful for detabbing non-MPW files.

- *Note:* Entab always detabs the input file, using the file's tab setting, or 4 if there is none. For MPW files, specifying a **-d** option would override the file's own tab setting, leading to incorrect results if a different value were used.

-l *quote...* Specify a list of left quoting characters. *Quote...* is a string of one or more nonblank characters. If **-l** is specified, **-r** must also be specified. Single quotation marks (') and double quotation marks (") are assumed as the default quoting characters.

-n Treat all quotes as "normal" characters—entab the file, replacing runs of spaces embedded in quoted strings with tabs.

▲ **Caution** This option should not be used when entabbing program source files. If this option is used, the **-q**, **-l**, and **-r** options are ignored. ▲

-p Write version and progress information to diagnostic output.

-q *quote...* Specify a list of characters to be used as both left and right quoting characters. *Quote...* is a string of one or more nonblank characters. This is the default option; single quotation marks (') and double quotation marks (") are assumed as the quoting characters.

-r *quote...* Specify a list of right quote characters. *Quote...* is a string of one or more nonblank characters. If **-r** is specified, **-l** must also be specified.

◆ *Note:* Entab does not check that a particular left quoting character matches a particular right quoting character.

-t *tabSetting*

Set the output file's tab setting to *tabSetting*. If the **-t** option is omitted, 4 is assumed for the tab setting. If you specify a tab setting of 0, no tabs are placed in the output. Thus **-t 0** may be used to completely detab input files.

▲ **Caution** If you specify the **-q, -l,** or **-r** option, you should quote the entire string parameter to these options (otherwise, the Shell may misinterpret special characters in the parameter string). ▲

Example

```
Entab -t 2 Example.p > CleanExample.p
```

Detabs the file Example.p (using the file's default tab setting), re-entabs it with a tab setting of 2, and writes the resulting output to CleanExample.p.

Warning

Beware of command formats such as

```
Entab Foo > Foo
```

Limitations

Entab does not take into account embedded formatting characters other than tab characters. Thus backspace characters may cause incorrect results.

The maximum width for an input line is 255 characters.

See also

Format command.

Equal—compare files and directories

Syntax Equal [-d | -r] [-i] [-p] [-q] *name... targetName*

Description Compares *name* to *targetName*. By default, Equal makes no comment if files are the same; if they differ, it announces the byte at which the difference occurred. When comparing directories, the default condition is to report all differences, including files not found—the **-i** option ignores files in *targetName* that are not present in *name*.

If *targetName* is a file, every *name* must also be a file. The specified files are compared with *targetName*.

If *targetName* is a directory and *name* is a file, Equal checks in *targetName* for the file *name* and compares the two files. That is, the command

```
Equal File1 Dir1
```

compares File1 with :Dir1:File1.

If more than one name is specified, Equal compares each name with the corresponding file or directory in *targetName*. All subdirectories are also compared. For example,

```
Equal File1 Dir1 Dir2
```

If *targetName* is a directory, *name* is a directory, and only one name is specified, the Equal command directly compares the two directories. That is, the command

```
Equal Dir1 Dir2
```

compares Dir1 (and all subdirectories) with Dir2.

Type Built-in.

Input None.

Output Differences are written to standard output.

Diagnostics Errors are written to diagnostic output.

Status

The following status codes may be returned:

0 Identical files.
1 Syntax error.
2 Inaccessible or missing parameter.
3 Files not equal.

Options

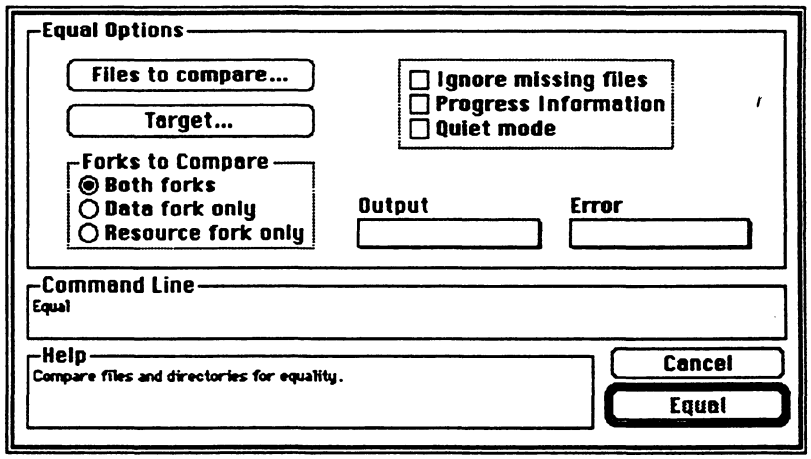
-i Ignore files missing from directory *name;* that is, if files in *targetName* are not present in *name,* Equal won't report the missing files as differences.

-d Compare the data forks only.

-r Compare the resource forks only.

-p List progress information as files are compared.

-q Remain quiet about differences; return status codes only.



Examples

```
Equal File1 File1Backup
```

Reports if the files are different and at what point they differ, in a message such as

```
File1 File1Backup differ in data fork, at byte 5
```

```
Equal -i  HD:Dir1  Disk1:Dir1
```

Compares all files and directories in HD:Dir1 with files and directories with the same names found in Disk1:Dir1, and reports any differences. This command does not report files in Disk1:Dir1 that aren't found in HD:Dir1.

```
Equal -i -d Backup:  HD:Source
```

Compares the data forks of all files on the volume Backup: with all those of the same name in the directory HD:Source.

```
Equal -p Old:≈.c  HD:Source
```

Compares all files on Old: ending in ".c" with their counterparts in HD:Source. Prints progress information as the comparison proceeds.

See also

Compare command.

Erase—initialize volumes

Syntax Erase [-y] [-s] *volume...*

Description Initializes the specified volumes— the previous contents are destroyed. A volume name must end with a colon (:). If *volume* is a number without a colon, it's interpreted as a disk drive number.

A dialog box requests confirmation before proceeding with the command, unless the **-y** option is specified. The **-y** option can be used in scripts to avoid this interaction.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

0 Successful initialization.
1 Syntax error.
2 No such volume, or boot volume.
3 Errors during the initialization procedure.

Options

-y Answer "Yes" to the confirmation dialog, causing initialization to begin immediately.

-s Format the disk for single-sided use (that is, as a 400K, non-HFS disk).

Examples

```
Erase Reports:
```

Initializes the volume entitled Reports.

```
Erase 1
```

Initializes the volume in drive 1 (the internal drive). The disk will be formatted as a 400K disk if drive 1 is a 400K drive, or as an 800K disk if drive 1 is an 800K drive.

Evaluate—evaluate an expression

Syntax

Evaluate [-h | -o | -b] [*word...*]
Evaluate *name* [*binary operator*] = [*word...*]

Description

The list of words is taken as an expression. After evaluation, the result is written to standard output. Missing or null parameters are taken as zero. You should quote string operands that contain blanks or any of the characters listed in the table that follows.

The operators and precedence are mostly those of the C language; descriptions follow.

The second form of the Evaluate command evaluates the list of words and assigns the result to the variable *name*. The result of the expression is not written to standard output in this case. C style operations of the form "+=", "-=", and so on, are supported. If *name* is undefined at the time of execution, it is interpreted as zero.

Different radices can be used in the input expression, and the result can be output in a different radix by using the **-h**, **-o**, or **-b** option. The default radix is decimal.

Expressions: An expression can include any of the following operators. (In some cases, two or three different symbols can be used for the same operation.) The operators are listed in order of precedence; within each group, operators have the same precedence.

	Operator			Operation
1.	*(expr)*			Parentheses are used to group expressions
2.	–			Unary negation
	~			Bitwise negation
	!	NOT	¬	Logical NOT
3.	*			Multiplication
	÷	DIV		Division
	%	MOD		Modulus division
4.	+			Addition
	– ·			Subtraction
5.	<<			Shift left
	>>			Shift right
6.	<			Less than
	<=	≤		Less than or equal to
	>			Greater than
	>=	≥		Greater than or equal to
7.	==			Equal
	!=	<>	≠	Not equal
	=~			Equal—regular expression
	!~			Not equal—regular expression
8.	&			Bitwise AND
9.	^			Bitwise XOR
10.	|			Bitwise OR
11.	&&	AND		Logical AND
12.	||	OR		Logical OR

All operators group from left to right. Parentheses can be used to override the operator precedence. Null or missing operands are interpreted as zero. The result of an expression is always a string representing a number in the specified radix (the default is decimal).

The logical operators !, NOT, ¬, &&, AND, || , and OR interpret null and zero operands as false, and nonzero operands as true. Relational operators return the value 1 when the relation is true, and the value 0 when the relation is false.

The string operators ==, !=, =~, and !~ compare their operands as strings. All others operate on numbers. Numbers may be decimal, hexadecimal, octal, or binary integers representable by a 32-bit signed value. Hexadecimal numbers begin with either **$** or **0x.** Octal numbers begin with a **0** (zero). Binary numbers begin with **0b**. Every expression is computed as a 32-bit signed value. Overflows are ignored.

Input	**Radices**
Decimal number	[0–9]
Hexadecimal number	0x[0–9A–F]
Octal number	0[0–7]
Binary number	0b[01]

The pattern-matching operators =~ and !~ are like == and != except that the right side is a regular expression that is matched against the left operand. Regular expressions must be enclosed within the regular expression delimiters /.../. Regular expressions are summarized in Appendix B.

- *Note:* There is one difference between using regular expressions after =~ and !~ and using them in editing commands. When evaluating an expression that contains the tagging operator, ®, the Shell creates variables of the form {®*n*}, containing the matched substrings for each ® operator. (See the examples that follow.)

Filename generation, conditional execution, pipe specifications, and input/output specifications are disabled within expressions, to allow the use of many special characters that would otherwise have to be quoted.

Expressions are also used in the If, Else, Break, Continue, and Exit commands.

Type Built-in.

Input None.

Output The result of the expression is written to standard output. Logical operators return the values 0 (false) and 1 (true).

- *Note:* To redirect Evaluate's output (or diagnostic output), enclose the Evaluate command in parentheses; otherwise, the > and ≥ symbols are interpreted as expression operators, and an error occurs. (See the fifth example that follows.)

Diagnostics

Errors are written to diagnostic output.

Status

These status codes may be returned:

0 Valid expression.
1 Invalid expression.

Options

-h Output the result in hexadecimal. The number will be prefixed with a 0x.

-o Output the result in octal. The number will be prefixed with a 0.

-b Output the result in binary. The number will be prefixed with a 0b.

Examples

```
Evaluate (1+2) * (3+4)
```

Does the computation and writes the result to standard output.

```
Evaluate -h 8 + 8
```

Does the computation and writes the result to standard output in hexadecimal (0x10).

```
Evaluate 0xA + 6
```

Writes the result 16 to standard output. (The default output radix is decimal. Use **-h** for hexadecimal.)

```
Evaluate lines += 1
```

The Evaluate command increments the value of the Shell variable {lines} by 1. If {lines} was undefined before executing the command, {lines} would be 1 after execution.

```
( Evaluate "{aPathname}" =~ /(([¬:]+:)*)®1≈/ ) > Dev:Null
Echo {®1}
```

These commands examine a pathname contained in the variable {aPathname} and return the directory prefix portion of the name. In this case, Evaluate is used for its side effect of enabling regular expression processing of a filename pattern. The right side of the expression `( /(([¬:]+:)*)®1≈/ )` is a regular expression that matches everything in a pathname up to the last colon and remembers it as the Shell variable {®1}. Evaluate's actual output is not of interest, so it's redirected to the bit bucket, Dev:Null. (See "Pseudo-Filenames" in Chapter 5.) Note that the use of I/O redirection means that the Evaluate command must be enclosed in parentheses so that the output redirection symbol, >, is not taken as an expression operator.

This is a complex but useful example of implementing a "substring" function. For a similar example, see the Rename command.

See also "Structured Commands" in Chapter 5.

"Pattern Matching (Using Regular Expressions)" in Chapter 6, and Appendix B.

Execute—execute a script in the current scope

Syntax Execute *script*

Description Executes the script as if its contents appeared in place of the Execute command. This means that variable definitions, exports, and aliases in the script will continue to exist after it has finished executing. (Normally these definitions, exports, and aliases would be local to the script.) Any parameters following *script* are ignored. Any parameters to the enclosing script are available within *script*.

- *Note:* If *script* is not a command file (that is, if it's a built-in command, tool, or application), the command is run as if the word Execute did not appear. Parameters are passed to the command as usual.

Type Built-in.

Input None.

Output None.

Diagnostics None.

Status Execute returns the status returned by *script*.

Options None.

Example

```
Execute "{ShellDirectory}"Startup
```

Executes the Startup (and UserStartup) scripts. This command is useful for testing any changes you've made to the Startup-UserStartup script. Variable definitions, exports, and aliases set in Startup and UserStartup will be available after Startup is done executing.

See also "Defining and Redefining Variables" in Chapter 5.

"The Startup and UserStartup Files" in Chapter 5.

Exists—confirm the existence of a file or directory

Syntax Exists [-d | -f | -w] [-q] *name...*

Description Determines the existence of the file or directory *name*. The options help you to distinguish between directories and files and different access permissions. The nonexistence of *name* is not considered an error (status remains zero).

Type Built-in.

Input None.

Output Files that exist and match the specifications have their names written to standard output.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

0 No error.
1 Syntax error.
2 Other error.

Options

-d Check if *name* is a directory.

-f Check if *name* is a file (as opposed to a directory).

-w Check if the user has write access to the file *name*. You cannot write to a file if it is open or locked.

-q Do not quote pathnames that are written to standard output.

Example

```
If Not "`Exists -d HD:dir`"
          NewFolder HD:dir
End
Duplicate ≈.c HD:dir
```

This example creates a new directory and copies all files ending with ".c" in the current directory to this new directory.

See also Newer command.

Exit—exit from a script

Syntax Exit [*status*] [If *expression*]

Description If the *expression* is nonzero (that is, true), Exit terminates execution of the script in which it appears. When used interactively, Exit terminates execution of previously entered commands. *Status* is a number; if present, it is returned as the status value of the script. Otherwise, the status of the previous command is returned. If the "If *expression*" is omitted, the Exit is unconditional. (For a definition of *expression,* refer to the description of the Evaluate command.)

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status If *status* is present, it is returned as the status value of the script. If the expression is invalid, –5 is returned. Otherwise, the status of the last command executed is returned.

Options None.

Example

```
Exit {ExitStatus}
```

As the last line of a script, this Exit command would return as a status value whatever value had previously been assigned to {ExitStatus}.

See also Evaluate command (for information on expressions).

"Structured Commands" in Chapter 5.

{Exit} and {Status} variables, in "Variables," Chapter 5.

Export—make variables available to programs

Syntax Export [-r | -s | *name...*]

Description Make the specified variables available to scripts and tools. The list of variables exported within a script is local to that script. An enclosed script or tool inherits a list of exported variables from the enclosing script . (See Figure 5-1 in Chapter 5 for clarification.)

- *Note:* You can make a variable available to all scripts and tools by setting and exporting it in the Startup or UserStartup files. (Startup acts as the enclosing script for all Shell operations.)

If no names are specified, a list of exported variables is written to standard output. (Note that the default output of Export is in the form of Export commands.)

Type Built-in.

Input None.

Output If no names are given, Export writes a list of exported variables to standard output.

Diagnostics None.

Status Export may return the following status codes:

0 No errors.
1 Syntax error.

Options

-r Reverse the sense of the output, causing Export to generate Unexport commands for all exported variables.

-s Suppress the printing of "Export" before the exported variables.

Example

```
Set AIncludes "{MPW}Interfaces:AIncludes:"
Export AIncludes
```

Defines the variable {AIncludes} as the pathname "{MPW}Interfaces:AIncludes:" and makes it available to scripts and programs.

See also

Unexport, Set, and Execute commands.

"The Startup and UserStartup Files" in Chapter 5.

"Exporting Variables" in Chapter 5.

FileDiv—divide a file into several smaller files

Syntax FileDiv [-f] [-n *splitpoint*] [-p] *file* [*prefix*]

Description FileDiv is the inverse of the Catenate command. It is used to break a large file into several smaller pieces. The input file is divided into smaller files, each containing an equal number of lines determined by the *splitpoint* (default=2000). The last file contains whatever is left over.

There is also an option (**-f**) for splitting a file only when a form feed character (ASCII $0C) occurs as the first character of a line that is beyond the splitpoint. This option lets you split a file at points that are known to be the tops of pages.

Each group of *splitpoint* lines is written to a file with the name *prefixNN*, where *NN* is a number starting at 01. If the *prefix* is omitted, the input file name is used as the prefix.

Type Tool.

Input An input file must be specified in the command line. Standard input is not used.

Output FileDiv creates files with names of the form *prefixNN*, where *NN* is a number. (If *prefix* is omitted, the input filename is used as a prefix.) Standard output is not used.

Diagnostics Parameter errors and progress information are written to diagnostic output.

Status FileDiv may return the following status codes:

0 Normal termination.
1 Parameter or option error.

Options

-f Split the input file only when at least *splitpoint* lines have been written to the current output file *and* there is a form feed character (ASCII $0C) as the first character of a line. The line containing the form feed becomes the first line in the next output file.

-n *splitpoint*

Split the input file into groups of *splitpoint* lines (or, if the **-f** option was specified, *splitpoint* or more lines). If the **-n** option is omitted, 2000 is assumed.

-p Write version information and progress information to diagnostic output.

Example

```
FileDiv -f -n 2500 Bigfile
```

Splits Bigfile into files of at least 2500 lines; splits the file at points where there are form feed characters. The output files have the names Bigfile*NN*, where *NN* is 01, 02, and so on.

Limitation

The maximum length of an input line is 255 characters.

Files—list files and directories

Syntax Files [*option...*] [*name...*]

Description For each disk or directory named, Files lists its contents; for each file named, Files writes its name and any other information requested. Information is written to standard output. When a directory is listed, all subdirectories are listed first in alphabetical order, followed by all files in alphabetical order. If no name is given, the current directory is listed.

Type Built-in.

Input None.

Output File information is written to standard output.

Diagnostics Errors are written to diagnostic output.

Status Files may return the following status codes:

0	All names were processed successfully.
1	Syntax error.
2	An error occurred.

Options

-c *creator*	List only those files with the given file creator.
-d	List subdirectories only.
-f	Give full pathnames for all files listed.
-i	Treat directories on the command line as files (ignore differences). That is, don't list the contents of directories; instead, just list the directory and any other information requested.
-l	List files in long format. This format is: name, type, creator, size, flags, last modification date, and creation date.
-m *count*	Multicolumn output. This option is not valid if specified with **-l** or **-x.**
-n	No header in the long or extended format. Without the **-l** or **-x** option, this option has no meaning.

-o List only the contents of the directories; do not print the directory titles themselves. Useful when combined with the **-r** option (or if multiple directories are given in the command line) to list only the contents of the directories.

-q Don't quote names in the output. Normally, the Files command quotes names that contain spaces or special characters.

-r Recursively list the subfolders encountered; that is, list every file in every directory.

-s Suppress the printing of directory names. Useful when combined with the **-r** option to get listing of all files (excluding directories).

-t *type* List only those files with the given file type.

-x *format* Extended format. This option generates a listing similar to that produced by the **-l** option, except that the fields to be printed are determined by *format. Format* is a string composed of the following letters (in any order) where the order determines the fields position in the output:

a	Flag attributes
b	Logical size in bytes of the data fork
c	Creator of file ('Fldr' for folders)
d	Creation date
g	Group (applies only to folders on AppleShare)
k	Physical size in kilobytes of both forks
m	Modification date
o	Owner (applies only to folders on AppleShare)
p	Privileges (applies only to folders on AppleShare)
t	Type of file
r	Logical size in bytes of the resource fork

Examples

```
files -r -s -f
HD:source:defs.h
HD:source:main.c
HD:source:backup:main.c
HD:source:backup:defs.h
HD:source:junk:tmpfile
```

Recursively lists the contents of the current directory, giving full pathnames and suppressing the printing of directory names.

```
files -d
:backup:
:junk:
```

Lists only the directories in the current directory.

```
Files -i -x kd "{AIncludes}"
Name                                Size  Creation-Date
----------------------- ------ ----------------
HD:MPW:Interfaces:AIncludes:        365K  8/25/87  5:32 AM
```

Lists the size and creation date of the "{AIncludes}" directory. Notice how the **-i** option is used to avoid printing the contents of the directory.

```
files -m 2
:backup:  deFs.h
:junk:    main.c
```

This is the two-column format. Notice the order of the files.

Find—find and select a text pattern

Syntax Find [-c *count*] *selection* [*window*]

Description Creates a selection in *window*. If no window is specified, the target window (the second window from the front) is assumed. It's an error to specify a window that doesn't exist.

Selection is a selection as defined in Chapter 6 and in Appendix B.

- *Note:* Searches do not necessarily start at the beginning of a window. A forward search begins at the end of the current selection and continues to the end of the document. A backward search begins at the start of the current selection and continues to the beginning of the document.

All searches are not case sensitive by default. You can specify case-sensitive searches by first setting the Shell variable {CaseSensitive} to a nonzero value. (Or, you can automatically set {CaseSensitive} by checking Case Sensitive in the dialog boxes displayed by the Find and Replace menu items.)

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

0 At least one instance of the selection was found.
1 Syntax error.
2 Any other error.

Option **-c** *count* For a count of *n*, find the *n*th occurrence of the selection.

Examples

```
Find •
```

Positions the insertion point at the beginning of the target window.

```
Find -c 5 /procedure/ Sample.p
```

Selects the fifth occurrence of "procedure" in the window Sample.p.

```
Find 332
```

Selects line 332 in the target window.

See also

"Selections" and "Pattern Matching" in Chapter 6, and Appendix B.

"Find Menu" in Chapter 3.

Flush—clear the command cache

Syntax Flush

Description Flush clears the MPW Shell's tool cache.

The MPW Shell keeps the most recently used tools in memory so that execution can be faster. However, there are times when you don't want the tools to be in the cache. For example, you cannot run a tool, and then switch to the Finder and delete the file. The Finder will report that the tool is busy. You might also want to flush the cache is when you are running benchmarks or timing tool performance.

Type Built-in.

Input None.

Output None.

Diagnostics None.

Status Flush may return the following status code:

0 No errors.

Options None.

Example

```
Flush
```

Flush the current cache. This will free all tools in the cache.

For...—repeat commands once per parameter

Syntax

For *name* In *word...*
  command...
End

Description

Executes the list of commands once for each word from the "In *word...*" list. The current word is assigned to variable *name*, and you can therefore reference it in the list of commands by using the notation {*name*}. You must end each line with either a return character (as shown above) or with a semicolon (;).

The Break command can be used to terminate the loop. The Continue command can be used to terminate the current iteration of the loop.

The pipe specification (|), conditional command terminators (&& and | |), and input/output specifications (<, >, >>, ≥, ≥≥, Σ, and ΣΣ) may appear following the End; they apply to all of the commands in the list.

Type

Built-in.

Input

None.

Output

None.

Diagnostics

Errors are written to diagnostic output.

Status

The following status codes may be returned:

0 The list of words or list of commands was empty.
-3 There was an error in the parameters to For.

Otherwise, the status of the last command executed is returned.

Options

None.

Examples

```
For i In 1 2 3
    Echo i = {i}
End
```

Returns the following:

```
i = 1
i = 2
i = 3
```

```
For File In ≈.c
    C "{File}" ; Echo "{File}" compiled.
End
```

This example compiles every file in the current directory whose name ends with the suffix ".c". The Shell first expands the filename pattern `≈.c`, creating a list of the filenames after the "In" word. The enclosed commands are then executed once for each name in the list. Each time the loop is executed, the variable {File} represents the current word in the list. {File} is quoted because a filename could contain spaces or other special characters.

```
For file in Startup UserStartup Suspend Resume Quit
    Entab "{file}" > temp
    Rename -y temp "{file}"
    Print -h "{file}"
    Echo "{file}"
End
```

This example entabs (replaces multiple spaces with tabs) the five files listed, prints them with headings, and echoes the name of each file after printing is complete. You might want to use this set of commands before making copies of the files to give to a friend. Entabbing the files saves considerable disk space, and printing them gives you some quick documentation to go with the files.

See also

Loop, Break, and Continue commands.

"Structured Commands" in Chapter 5.

Format—set or view the window format

Syntax Format [[-f *fontname*] [-s *fontsize*] [-t *tabsize*] [-a *attributes*]] | [-x *formatting*] [*window...*]

Description This is a scriptable form of the Format menu command in the Edit menu. Use it to set the format of a specified list of windows. If no window is specified, the command operates on the target window. If no options are specified (other than **-x**), the current format settings are written to standard output.

- *Note:* The Format command (and the Format menu command) modify the format of an existing window. The format related variables such as {Tab} and {Font} are used to initialize the format of a new window.

Input None.

Output If the optional parameters are omitted, or the **-x** option is specified, the current format settings are written to standard output.

Diagnostics Errors are written to diagnostic output.

Status Format may return the following status codes:

0 No errors.
1 Syntax error (error in parameters).
2 All other errors.

Options

-f *fontname* Change the font in the specified windows to *fontname.*

-s *fontsize* Change the font size in the specified windows to *fontsize.*

-t *tabsize* Change the tab size in the specified windows to *tabsize.*

-a *attributes* Set or clear the invisible and auto-indent states. Attributes is a string composed of the characters in the following list. Attributes that aren't listed remain unchanged.

A auto-indent
I show invisibles

Uppercase letters turn on the attribute; lowercase letters turn off the attribute.

-x *formatting*

Use this option to specify the output format when the current settings are displayed. An error message occurs if any other option is specified. The parameter *formatting* is a string composed of the following letters (in any order), where the order determines the field's position in the output. The specified values will be separated by spaces when they are output.

f	Font name
s	Font size
t	Tab size
a	Auto-indent and show invisibles state

Examples

```
Format -f Monaco -t 8 -a A "{target}"
```

Sets the font, tab size, and auto-indent in the target window. The font size and invisible settings are not changed.

```
Format -s 12 MyWindow
```

Changes the font size in MyWindow to 12 point.

```
Format "{Target}"
```

A format statement with no options displays the current format of the window, such as the following:

```
Format -f Monaco -s 9 -t '8 -a Ai
```

You can then select and execute this output format.

```
Format -x tsf
4 9 Monaco
```

Displays only the values of the specified options. Use this option for easily retrieving one or two values and assigning them to Shell variables for later use.

See also

The "Edit Menu," in Chapter 3.

"Variables" in Chapter 5.

GetErrorText—display text for system error numbers

Syntax

GetErrorText [-f filename] [-s filename] [-n] [-p] errnbr [,insert,...] ...
GetErrorText -i idnbr ...

Description

Displays the error messages corresponding to a set of specified error numbers or ID numbers. By default, GetErrorText assumes that the error numbers correspond to Macintosh Operating System error numbers. The file SysErrs.Err is a special file used by MPW tools to determine the error messages corresponding to system error numbers. Other system error message files may be specified by using the **-s** option.

In addition to system errors, some tools have their own error message files. For example, the assembler's error message file is in the data resource fork of Asm itself. For such tools, you can display the error messages corresponding to tool error numbers by specifying the **-f** option. In this case, you can specify sample inserts, along with the error numbers, for error messages that take inserts, as shown above.

GetErrorText can also display the meanings of the ID numbers reported by the System Error Handler in alert dialog boxes. The **-i** option is used for this purpose.

Type

Tool.

Input

All input is specified through the parameters.

Output

The error messages are written to standard output.

Diagnostics

Errors are written to diagnostic file.

Status

GetErrorText may return the following status codes:

0 Normal termination.
1 Parameter or option error.

Options

-f *filename* A tool's error message filename. Either **-f** or **-s**, but not both, may be specified.

-i *fdnbr* Report the meaning of the specified System Error Handler ID number.

-s *filename* The error message filename for a system error. Either **-f** or **-s**, but not both, may be specified. The default is **-s** SysErrs.Err.

-n Do not generate error numbers as part of the error messages. This option is ignored if system errors are displayed.

-p Writes GetErrorText's version information to the diagnostic file.

Examples

```
GetErrorText -43 -44 -45
```

Displays the error messages corresponding to system errors -43, -44, and -45.

```
GetErrorText -i 28  2
```

Displays the error messages corresponding to system ID numbers 28 and 2.

GetFileName—display a standard file dialog box

Syntax

GetFileName [-q] [-s] [[-t TYPE]... | -p | -d] [-m *message*] [-b *buttontitle*] [*pathname*]] | [-c]

Description

GetFileName displays a standard file dialog box. Either SFPutFile or SFGetFile is called, and the returned filename or pathname is written to standard output. The standard file starting directory is set to *pathname* if specified. If *pathname* includes a local filename and if SFPutFile is called, the local filename is used as the default filename. See the examples.

Type

Tool.

Input

None.

Output

The filename or pathname you select is written to standard output. The pathname is always a full pathname starting at the selected volume's root.

Diagnostics

Parameter errors are written to diagnostic output.

Status

The following status codes may be returned:

0 User specified a file and no errors occurred.
1 Parameter or option error.
2 System error.
4 User canceled the standard file dialog box.

Options

-c Write the current Standard File pathname to standard output.

-p Display an SFPutFile dialog box.

-d Display an SFGetFile dialog for selecting a directory.

-m *msg* Specify a prompt message.

-b *buttontitle*
Specify the title for the default button in the various dialog boxes. If this option is not specified, *Open* is used in the standard SFGetFile dialog box, *Save* is used in the standard SFPutFile dialog box, and *Directory* is used in the directory SFGetFile dialog box.

-q Suppress quoting the filename written to standard output.

-s Return a status of zero even if Cancel is clicked.

-t *type* Specify a type to use in filtering the SFGetFile. Up to four types may be specified. This option is case sensitive.

Examples

```
open `GetFileName -t TEXT {pinterfaces}`
```

Opens the text file in directory {pinterfaces} chosen in SFGetFile by the user.

```
GetFileName -p HD:MPW:StartUp
```

An SFPutFile dialog box is displayed with the directory set to HD:MPW: and StartUp is displayed in the textedit field of the dialog box.

Limitation The resulting filename cannot be longer than 255 characters.

See also "The Standard File Package," *Inside Macintosh,* Volume I.

GetListItem—display items for selection in a dialog box

Syntax GetListItem [*option...*] [*items...*]

Description Takes the items on the command line (or, if no items are present on the command line, items from standard input) and lists them in a dialog box. Items in the list can be selected with the mouse and modifier keys. Selected items are written to standard output when the OK button is clicked.

Type Tool.

Input Reads standard input for the items if none are specified on the command line.

Output Selected items are written to standard output if the OK button is clicked.

Diagnostics Errors are written to diagnostic output.

Status GetListItem may return the following status codes:

0 No errors (or Cancel button was clicked if **-c** option is used).
1 Syntax error (bad option).
2 Cancel button was clicked.

Options

-c Return a status code of 0 when Cancel is clicked.

-d *item* *Item* is entered as an element in the list and comes up selected. This option may be specified more than once.

-m *message* Display *message* above the list of items.

-q Don't quote items in the output.

-r *rows* Make the list with this many rows.

-s Allow only a single item to be selected from the displayed list. In single-selection mode, GetListItem behaves very much like the list in the Standard File dialogs—the cursor keys move the selection, and keystroke matching is performed.

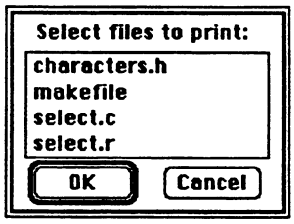
-w *width* Make the list this many pixels wide.

- *Note:* GetListItem uses the **-r** and **-w** values only as a guideline. For example, if the value given for *rows* is larger than the number of rows on the screen, GetListItem will use a smaller number of rows than requested. GetListItem does not give error messages when the **-r** or **-w** arguments are out of range. Rather, it makes a reasonable guess at a value.

Examples

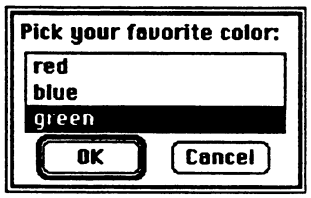
```
print `files -t TEXT | GetListItem -m "Select files to print:
```

Lists all text files in the current directory and prints those selected by the user, as shown below.




```
GetListItem red blue -d green -m "Pick your favorite
color:"
```

Display a list of three colors with green preselected, as shown below.



Limitation GetListItem cannot handle a list greater than 32K characters.

Help—display summary information

Syntax Help [-f *helpFile*] [*command...*]

Description Help writes information about the specified commands to standard output. If no command is specified, information about Help is written to standard output. *Command* can include any of the following:

commandName	Information about *commandName*
`commands`	A list of all MPW commands
`expressions`	A summary of expressions
`patterns`	A summary of pattern specifications (regular expressions)
`selections`	A summary of selection operators
`characters`	A summary of MPW Shell special characters
`shortcuts`	A summary of MPW shortcuts
`variables`	A summary of MPW Shell variables
`projector`	A summary of Projector commands

By default, the Help command looks for information in the file MPW.Help. It looks for this file first in {ShellDirectory}; if the file isn't found, Help looks in {SystemFolder}.

The following syntax notation is used to describe Macintosh Programmer's Workshop commands:

[*optional*]	Square brackets mean that the enclosed elements are optional.
repeated...	Ellipses indicate that the preceding item can be repeated one or more times.
a | *b*	A vertical bar indicates an either/or choice.
(*grouping*)	Parentheses indicate grouping (useful with "|" and "...").
< *input*	If *input* is not specified, the command reads from standard input.
> *output*	The command writes to standard output.
≥ *progress*	Progress information is written to diagnostic output (with the **-p** option).

A Help file is a set of entries, each separated by a blank line beginning with one hyphen. Each entry may contain one or more lines. The first word of the first line in each entry is the keyword that is looked up by the Help command. When the word is located, the line in which it occurs, and all following lines until a separator is encountered, are written to standard output. If no parameters are given to the Help command, the first entry is written to standard output. Here is an example from the MPW.Help file:

```
New [name...]
-
Newer [-c] [-e] [-q] file... target > newer
          -c    # compare creation dates
          -e    # report.names that have the same (equal)
                # date as target
          -q    # don't quote filenames with special characters
-
NewFolder name...
```

Type Built-in.

Input None.

Output Command information is written to standard output.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

0 Information was found for the given command.
1 Syntax error.
2 A command could not be found.
3 The help file could not be opened.

Option **-f** *helpFile* Specify help file to be searched. (A help file is an ordinary MPW text file.) The default file is MPW.Help.

Examples

```
Help Rez
```

Writes information such as

```
Rez [option...] [file...]   < file ≥ progress
-a[ppend]                   # merge resource into output resource file
-align word | longword      # align resource to word or longword boundaries
-c[reator] creator          # set output file creator
-d[efine] name[=value]      # equivalent to: #define macro [value]
-i[nclude] pathname         # path to search when looking for #include files
-o file                     # write output to file (default Rez.Out)
-ov                         # ok to overwrite protected resources when appending
-p                          # write progress information to diagnostics
-rd                         # suppress warnings for redeclared types
-ro                         # set the mapReadOnly flag in output
-s[earch] pathname          # path to search when looking for INCLUDE resources
-t[ype] type                # set output file type
-u[ndef] name               # equivalent to #undef name
```

```
Help -f myHelpFile myCommand
```

Uses a custom help file to display information about `myCommand`.

If...—conditional command execution

Syntax

If *expression*
    command...
[Else If *expression*
    command...] ...
[Else
  command...]
End

Description

Executes the list of commands following the first *expression* whose value is nonzero. (Null strings are considered zero.) At most one list of commands is executed. You may specify any number of "Else If" clauses. The final Else clause is optional. The return characters (as shown above) or semicolons must appear at the end of each line.

The pipe specification (|), conditional command terminators (&& and ||), and input/output specifications (<, >, >>, ≥, ≥≥, Σ, ΣΣ) may appear following the End and apply to all of the commands in the list.

For a definition of *expression*, see the description of the Evaluate command.

Type

Built-in.

Input

None.

Output

None.

Diagnostics

Errors are written to diagnostic output.

Status

The following status codes may be returned.

0 None of the lists of commands were executed.
−1 Invalid expression.

Otherwise, Status returns the value returned by the last command executed.

Options

None.

Examples

```
If {Status} == 0
   Beep  1a,25,200
Else
   Beep -3a,25,200
End
```

Produces an audible indication of the success or failure of the preceding command.

```
For window in `Windows`
   If "{window}" != "{Worksheet}" AND "{window}" != "{Active}"
      Close "{window}"
   End
End
```

Closes all of the open windows except the active window and the Worksheet window. (Refer also to the Windows command.)

The following commands, as a script, would implement a trivial case of a general "compile" command:

```
If "{1}" =~ /≈.c/
   C {COptions} "{1}"
Else If "{1}" =~ /≈.p/
   Pascal {POptions} "{1}"
End
```

If the above commands were saved in a file (say, as "Compile"), both C and Pascal programs could be compiled with the command

`Compile` *filename*

See also

Evaluate command (for a description of expressions).

"Structured Commands" in Chapter 5.

Lib—combine object files into a library file

Syntax

Lib [*option...*] *objectFile...*

Description

Combines the specified object files into a single file. Input files must have type `'OBJ'`.

Lib is used for the following:

- Combining object code from different languages into a single file.
- Combining several object files into a larger object file (a library).
- Combining several libraries into a single library, for use in building a particular application or desk accessory. This can greatly improve the performance of the Linker.
- Deleting unneeded modules (with the **-dm** option), changing segmentation (the **-sg** and **-sn** options), or changing the scope of a symbol from external to local (the **-dn** option). (These options are useful when you construct a specialized library for linking a particular program.)

Object files that have been processed with Lib result in significantly faster links when compared with the "raw" object files produced by the assembler or compilers.

The output of Lib is logically equivalent to the concatenation of the input files, except for the optional renaming, resegmentation, and deletion operations, and the possibility of overriding an external name. The resolution of external names in Lib is identical to Link—in fact, the two programs share the same code for reading object files. Although multiple symbols are reduced to a single symbol, no combining of modules into larger modules is performed, and no cross-module references are resolved. This behavior guarantees that the Linker's output will be the same size whether or not the output of Lib was used.

See "Library Construction" in Chapter 10 for a detailed discussion of the behavior and use of Lib.

Type

Tool.

Input

Lib does not read standard input.

Output

Lib does not write to standard output. The combined library output is placed in the data fork of the output library file. The default output file is Lib.Out.o—you can specify another name with the **-o** option. The output file is given type `'OBJ'` and creator `'MPS '`. Symbolic information is retained by default; use the **-sym** option to eliminate it.

Diagnostics

Errors and warnings are written to diagnostic output. Progress information is also written to the diagnostic file if you specify the **-p** option.

Status

Lib may return the following status codes:

0 No problem.
1 Syntax error.
2 Fatal error.

Options

-d

Suppress warnings for duplicate symbol definitions (data and code).

-df *deleteFile*

Delete the list of external modules found in *deleteFile*. *DeleteFile* is a text file generated by the Linker option **-uf.** See the Link command and "Library Construction" in Chapter 10 for information.

-dm *name*[,*name*...]

"Delete Module"—delete the specified external modules from the output file. The variable *name* may be either an external module or an entry-point name. When an entry point is deleted, only that entry point is deleted, not the module or any other entry point in that module. If a module is deleted, all of its associated entry points are also deleted. The contents of the module and all entry points are removed from the output file.

Note: References to names deleted in this way persist as references "by name." That is, if the references are from active code, they must be resolved by external modules or entry points in another file.

The primary use of this operation is to make the library file smaller, so that subsequent links are faster. You can use the Linker option **-uf,** which lists unreferenced ("dead") modules or entry points, to generate a list of names that can be deleted in this way.

-dn *name*[,*name*...]

"Delete Name"—delete the list of external names from the output file by reducing their scope to local. The option **-dn** is a "gentle" deletion in that it affects only the list of *external* module or entry-point names. The contents of the module, other entry points, references, and so on are still present in the output file. References to names "deleted" in this way will continue to refer to the same code, but with local scope. This is a useful operation when a global name conflict occurs between two pieces of code, one of which is library code from which you don't need to call the name directly.

-o *name*.o Place the output in file *name*.o. (The default name is Lib.Out.o).

-p Write progress and summary information to diagnostic output.

-sg *newSeg*=*oldSeg1* [,*oldSeg2*]...

Change segmentation. All code in the old segments named *oldSeg1*,*oldSeg2*,... is placed in the segment named *newSeg*.

-sn *oldSeg*=*newSeg*

Change a segment name. All code in the segment named *oldSeg* is placed in the segment named *newSeg*.

- *Note:* The **-sn** and **-sg** options behave exactly as in Link, except that **-sg** is limited to identifiers on both sides of the equal sign. The arbitrary string for a desk accessory name can be introduced only with Link, not with Lib. The major difference between **-sn** and **-sg** is that the order of the option parameters *oldSeg* and *newSeg* is reversed. (This is done for consistency with Link.)

-sym [on | full | off

Enable or disable writing symbolic data to support SADE. The default is to retain symbolic information.

,nolines	Omit line information.
,nolabels	Omit label information.
,novars	Omit variable information.
,notypes]	Omit type information.

-w Suppress warning messages.

-ver *number*

Set the version of the OMF file to *version* (not checked). This is useful when you are distributing a library for older versions of MPW (version 1 for MPW 2.0.2, version 0 for MPW 1.0).

Example

```
Lib {CLibraries}≈ -o {CLibraries}CLibrary.o
```

Combines all of the library object files from the {CLibraries} directory into a single library named CLibrary.o. For applications that require most or all of the C library files, using the new CLibrary file will reduce link time.

See also Link, DumpObj, and DumpCode commands.

"Optimizing Your Links" and "Library Construction" in Chapter 10.

Appendix H.

Line—find a line number

Syntax Line *n*

Description Line finds line *n* in the target window. The parameter *n* is usually an integer, but may be any selection expression. The target window becomes the active (frontmost) window.

Line is a script containing these two commands:

```
Find "{1}" "{target}" # Find line n in the target window
Open "{target}"       # Bring the target window to the top
```

Type Script.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status Status codes can be returned by either the Find or the Open commands that make up the Line script:

0 No errors.
1 Syntax error.
2 No target window; other error.
3 System error.

Options None.

Examples

```
Line 123
```

Finds line 123 in the target window and makes the target window the new active window.

```
### Undefined symbol: length
    File "Count.c"; Line 75
```

The File and Line commands above are part of an error message produced by the MPW C compiler. The MPW Assembler and MPW Pascal compilers produce errors when using similar formats. You can execute such error messages to find the line that contains the error.

The command File is defined as an alias for Target in the Startup file. Thus File opens the specified file as the target window. Line then selects the offending line in the window and brings the window to the front. Notice that the remainder of the error message is a comment.

See also Find command.

Link—link an application, tool, or resource

Syntax

Link [*option...*] *objectFile...*

Description

Links the specified object files into an application, tool, desk accessory, or driver. The input object files must have type 'OBJ'. Linked segments from the input object files are placed in code resources in the resource fork of the output file. The default output filename is Link.Out, but you can specify other names with the **-o** option.

For detailed information about the linker, and instructions for linking applications, MPW tools, and desk accessories, see Chapters 8 and 10. The first dialog box of Link's Commando dialog is reprinted here for convenience.

The linker's default action is to link an application, placing the output segments into 'CODE' resources. When you link an application, all old 'CODE' resources are deleted before the new 'CODE' resources are written. By default, resources created by the linker are given resource names that are the same as the corresponding segment names. You can change a resource (segment) name with the **-sn** or **-sg** options, and you can create unnamed resources with the **-rn** option.

The linker executes in four phases:

- **Input phase:** The linker reads all input files, finds all symbolic references and their corresponding definitions, and constructs a reference graph. Duplicate references are found and warnings are issued.
- **Analysis phase:** The linker allocates and relocates code and data, detects missing references, and builds the jump table. If the **-l** or **-x** option is given, Link produces a linker map or cross-reference listing. The linker also eliminates unused code and data.
- **Output phase:** The linker copies linked code segments into code resources in the resource fork of the output file. By default, these resources are given the same names as the corresponding segment names. (The cursor spins backward during this phase.)
- **Symbolic output phase:** Optionally, Link may be used to create the .SYM file for use with SADE.

Type

Tool.

Input

Link does not read standard input.

Output

By default, linked segments are placed in 'CODE' resources in the resource fork of the output file. The default output filename is Link.Out, but you can specify other names with the **-o** option. If the output file already exists, the linker adds or replaces code segments in the resource fork. If the output file doesn't exist, it is created with file type APPL and creator `'????'`. The **-t** and **-c** options can be used to set the output file type and output file creator to other values.

- *Note:* If a linker error or user interrupt causes the output file to be invalid, the linker sets the modification date on the file to "zero" (that is, January 1, 1904, 12:00 A.M.). This guarantees that Make will recognize that the file needs to be relinked.

If you specify **-sym**, Link creates a symbolics file for the debugger.

If you specify the **-l** option, Link writes a **location map** to standard output. The map is produced in location ordering—that is, it is sorted by *segNum, segOffset*. The format is divided into several fields:

name segName segNum, segOffset [*@JTOffset*] [#] [E] [C] [*fileNum, defOffset*]

There is also another **location map** option, **-map**, which is more readable and includes more information. See Chapter 10 for further information.

Diagnostics

Errors and warnings are written to diagnostic output. Progress information is also written to diagnostic output if the **-p** option is specified.

Status

These status codes may be returned:

0 No problem.
1 Syntax error.
2 Fatal error.

Options

Note: Numeric values for options can be specified as decimal constants or as hexadecimal constants preceded by the dollar sign character ($).

-ac *boundary*
Align code to *n* byte boundaries. The boundary must be a power of 2. The default alignment is 2.

-ad *boundary*
Align data to *n* byte boundaries. The boundary must be a power of 2. The default alignment is 2.

-c *creator* Set the output file creator to *creator*. The default creator is `'????'`.

-d Suppress warnings for duplicate symbol definitions (for data and code).

-da Convert segment names to desk accessory names at output time. Desk accessory names begin with a leading null character ($00). This option is used when linking assembly-language code into a final desk accessory (resource type 'DRVR').

-f Treat duplicate data definitions as FORTRAN "common" regions (multiple data modules with the same name); the size of the largest module is used. There may be at most one initialization of the data.

-l Write a location-ordered map to standard output. The performance-measurement tools and other scripts may rely on this option. Usually, this option is used with output redirection in effect. For example,

```
Link ObjFile -l > MyMapFile
```

-la List anonymous symbols in the location map. The default is to omit anonymous symbols from the map.

-lf Write a location map to standard output and include the symbol definition location in the input file—that is, the file number and byte offset of the module or entry-point record. (These records are discussed in detail in Appendix H.) The default is to omit the symbol definition location.

-map Write a location map to standard output, but print a more readable map, so that the A5 world has the correct offsets. Also provides sizes of all modules. This is the preferred location map.

-m *mainEntry*

Set (or override) the main entry point specified in the object files. *MainEntry* is a module or entry-point name.

- *Note:* For an application or MPW tool, the main entry point is assigned the first jump-table entry, as required by the Segment Loader. If a main entry point is specified for a desk accessory, driver, or other type of link for purposes of using Link's active-code analysis feature, the main entry point should be the first byte of code in the first Link input file. (A desk accessory has no jump table.)

-ma *name=alias*

"Module alias"—give the module or entry-point *name* the alternate name *alias*. This option lets you resolve undefined external symbols at link time, when the problem is caused by differences in spelling or capitalization. Note that you can't use an alias specification to override an existing module or entry point because the original name is retained.

-mf Let the linker use MultiFinder's temporary memory allocation routines, if they are available. If MultiFinder is not available, this option has no effect and is completely silent. If Link is in danger of running out of space in the MPW Shell's heap, and if the extra memory is available, Link will spill over into the MultiFinder temporary allocation region.

▲ **Caution** Link's use of this region excludes other applications, even the Finder. ▲

If Link aborts abnormally (that is, a crash or NMI, followed by a MacsBug "G sysrecover" or ES command), much of the MultiFinder temporary memory region might be left permanently allocated, thus crippling launches and Finder copy operations. The only way to recover from this situation is to restart the Macintosh.

-msg *keyword* [,*keyword...*]

Enable or suppress certain warning and error messages:

[no]dup Enable [suppress] warning messages about duplicate symbols. The default is **nodup**.

[no]multiple Enable [suppress] multiple error messages on undefined references to a label. This lets you catch all references to an undefined symbol with one link. The default is **nomultiple**.

[no]warn Enable [suppress] warnings. The default is **warn**.

-o *outputFile*

Place the linker output in *outputFile*. If no **-o** option is specified, the default output filename is Link.Out.

-opt *keyword* [,*keyword...*]

Optimize Object Pascal. This option is followed by one or more keywords, separated by commas or Shell-quoted spaces. Use this option instead of the Optimize tool distributed with MacApp, because Optimize does not work with MPW 3.0 files.

off Disable Object Pascal optimization (do nothing).

on Optimize Object Pascal method tables.

nobypass Always go through the jump table (do not optimize monomorphic method-calls to PC-relative JMP instructions).

-p Write progress and summary information to diagnostic output.

-ra [*seg*]=*nn* Set the resource attributes of a segment or segments. If *seg* is specified, the single segment named *seg* is given the attribute value *nn*. If *seg* is omitted, all segments except 0 and 1 are given the attribute value *nn*. (If you intend to set the attributes of all segments, you must specify this option before any other options that name segments, such as **-sn** and **-sg**.) The segment containing the main entry point (the `'CODE'` resource with ID=1) must be set individually to override the default resource attributes (described in Chapter 8).

Segment attributes may be specified as a decimal or hexadecimal number, or with a list of comma-separated resource attributes (the initial 'res' may be omitted):

resSysHeap	64	$40
resPurgeable	32	$20
resLocked	16	$10
resProtected	8	$08
resPreload	4	$04
resChanged	2	$02

The linker essentially ignores the `resChanged` bit, and does not check or enforce settings for the other resource attribute bits. Bits 0 and 7 ($01 and $80) are currently reserved and should not be set.

The default resource attributes for an application are:

'CODE' rsrc	Attributes		Description
0 (jump table)	32	$20	resPurgeable
1 ("Main")	52	$34	resPurgeable,resLocked, resPreload
others	32	$20	resPurgeable

When linking MPW tools (type `'MPST'` and creator `'MPS '`) all segments default to `resPurgeable`. Do not set the `resLocked` bit for a tool.

-rn Suppress the setting of resource names. (The default is to name each resource with the name of the segment.) Desk accessories must always be named.

-rt *type=ID* Set the output resource type to *type* and the ID to *ID*. This option indicates the link of a desk accessory or driver—that is, only one resource is modified. (The default is type 'CODE' and resource IDs numbered from 0.)

- *Assembly-language note:* When you link a desk accessory or driver, Link uses PC-relative offsets, and attempts to edit JSR, JMP, LEA, or PEA instructions from A5-relative to PC-relative addressing mode. Other instructions, specifying the A5-relative addressing, generate an error message.

-sg *newSeg*=[*oldSeg1* [,*oldSeg2*]...]...
Change segmentation. All code in the segments *oldSeg1, oldSeg2,...* is placed in the segment *newSeg*. If no *oldSeg* (and no =) is specified, Link will map all segments to *newSeg*.

-sn *oldSeg=newString*
Change a segment name. All code in the segment named *oldSeg* is placed in the segment named *newString*.

There are three major differences between **-sn** and **-sg:**

- The option **-sn** allows an arbitrary string for the new name, whereas **-sg** is intended only for identifiers separated by commas.
- The term newSeg is not just a name, but a segment.
- The order of the *oldSeg* and *newSeg* parameters is reversed.

For example,

```
Link... ∂
-sg Main=SAConsol,StdIO,%A5Init  ∂
-sn Main="MyDA"  ∂
...
```

The first option combines the three specified segments into one segment named Main; the second option renames Main to "MyDA".

-srt Sort A5-relative data into 32-bit and 16-bit groups. All 16-bit referenced data is placed as close as possible to A5.

-ss *size* Change the maximum segment size to *size*. The default value is 32,760 (32K minus a few overhead bytes). The value *size* can be any value greater than 32,760.

- *64K ROM note:* Caution! Applications with segments greater than 32K in size might not load correctly on Macintoshes with 64K ROMs.

-sym [on | full | off
Enable or disable writing symbolic data to support SADE.

,nolines	Omit line information.
,nolabels	Omit label information.
,novars	Omit variable information.
,notypes]	Omit type information.

-t *type* Set the output file type to *type*. The default type is APPL.

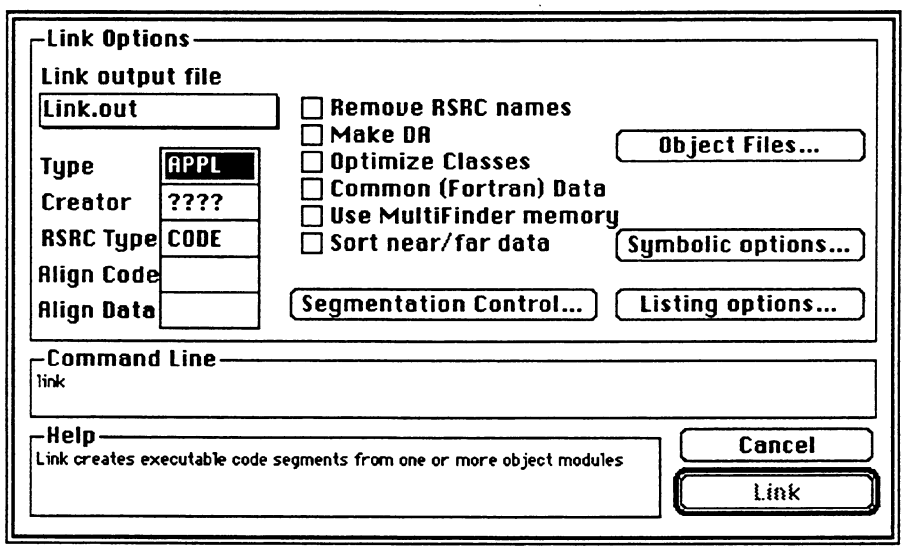
-uf *deleteFile*
List unreferenced modules in the text file *deleteFile*. (This option is useful in identifying dead source code.) This file can be used as input to Lib in building a specialized library that optimizes subsequent links. See the Lib command's **-df** option and "Library Construction" in Chapter 10 for more details.

-w Suppress warning messages.

- *Note:* Warnings generally indicate potential errors at run time.

-x *crossRefFile*

Generate a cross-reference listing of active modules and entry points. The listing is ordered by module within each segment. For each module, the following information is listed: each active entry point in the module, other modules and entry points that are referenced by the module, and other modules that reference this module. For each entry point in a module, the modules that reference the entry point are listed.



Examples

```
Link Sample.p.o ∂
        "{PLibraries}"PInterface.o ∂
        "{PLibraries}"PasLib.o ∂
        "{Libraries}"Runtime.o ∂
        -o Sample ∂
        -la >Sample.map
```

Links the main program file Sample.p.o with the libraries PInterface.o, PasLib.o, and Runtime.o, placing the output in Sample and placing the Linker map in the file Sample.map. Sample is an application that can be launched from the Finder or executed from MPW.

```
Link  -rt MROM=8  -c 'MPS '  -t ZROM  -ss 140000 ∂
    -l > ROMLocListing  -o MyROMImage  {LinkList}
```

Links the files defined in the Shell variable {LinkList} into a ROM image file, placing the output in the file MyROMImage. The segment size is set to 140,000 bytes, and the ROM is created as a resource 'MROM' with ID=8. The file is typed as being created by MPW (creator 'MPS '), with file type ZROM. Link's location-ordered listing is placed in the file ROMLocListing.

For additional examples, see "Link" in Chapter 10 and the makefiles in the Examples folders for the languages you are using.

See also

Lib command and Appendix H, "Object File Format."

Chapter 8, "The Build Process," and Chapter 10, "Link."

The Segment Loader and the Resource Manager chapters in *Inside Macintosh.*

Inside Macintosh, Volume IV, for information on the 128K ROM, the System Folder, and the Finder.

Loop...End—repeat command list until Break

Syntax

Loop
command...
End

Description

Executes the enclosed commands repeatedly. The Break command is used to terminate the loop. The Continue command can be used to terminate the current iteration of the loop.

The pipe specification (|), conditional command terminators (&& and | |), and input/output specifications (<, >, >>, ≥, ≥≥, Σ, and ΣΣ) may appear following the End, and apply to all of the commands in the list.

Type

Built-in.

Input

None.

Output

None.

Diagnostics

Errors are written to diagnostic output.

Status

Loop returns the status of the last command executed.

Options

None.

Example The following script runs a command several times, once for each parameter:

```
###  Repeat - Repeat a command for several parameters  ###
#  Syntax:
#             Repeat  command  parameter…
#
# Execute command once for each parameter in the parameter
# list. Options can be specified by including them with
# the command name in quotes.
#
Set cmd "{1}"
Loop
    Shift
    Break If "{1}" == ""
    {cmd} "{1}"
End
```

Notice that Shift is used to step through the parameters, and that Break ends the loop when all parameters have been used.

See also Break, For, and Continue commands.

"Structured Commands" in Chapter 5.

Make—build up-to-date version of a program

Syntax

Make [*option...*] [*targetFile...*]

Description

Generates a set of Shell commands that you can execute to build up-to-date versions of the specified target files. (If no target is specified, the target on the left side of the first dependency rule in the makefile is built.) Make allows you to rebuild only those components of a program that require rebuilding. Make determines which components need rebuilding by reading a **makefile**. This is a text file that describes dependencies between the components of a program, along with the Shell commands needed to rebuild each component. You can specify makefiles with the **-f** option. After processing the makefiles, Make writes to standard output the appropriate set(s) of commands needed to rebuild the target(s).

See "Format of a Makefile" in Chapter 9 for a description of the format of a makefile. The first dialog box of Make's Commando dialog is reproduced here for convenience.

Make executes in two phases:

- In the first phase, Make reads the makefile(s) and creates a file (target) dependency graph. (The "beachball" cursor spins counterclockwise during this phase.)
- In the second phase, Make generates the build commands for the target to be built (the cursor spins clockwise). If a target file doesn't exist or if it depends on files that are out-of-date or newer than the target, Make writes out the appropriate command lines for updating the target file. This process is recursive and "bottom up" so that commands are issued first for those lower-level dependencies that need to be rebuilt.

You can execute the generated build commands after Make is done executing.

Type

Tool.

Input

Standard input is not read. If you don't specify a makefile with the **-f** option, Make tries to open a file called MakeFile. If no target file is specified on the command line, Make uses the first target encountered in the makefile.

Output

If any files need to be updated, Make writes the appropriate Shell commands to standard output.

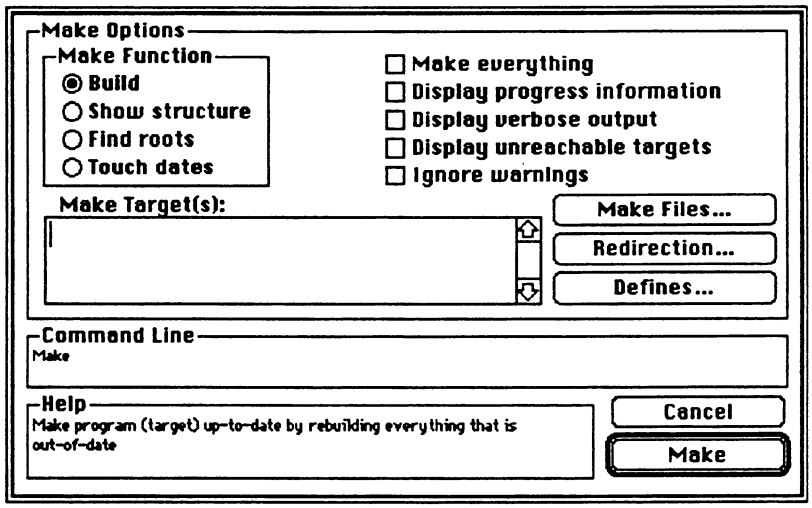
Diagnostics Errors, warnings, and diagnostic information (if requested) are written to diagnostic output. If you specify the **-p** option, progress and summary information is also written to diagnostic output.

Status The following status codes may be returned:

0 Successful completion.
1 Parameter or option error.
2 Execution error.

Options **-d** *name*[=*value*]

Define a variable *name* with the given *value*. Variables defined from the command line take precedence over definitions of the same variable in the makefile. Thus definitions in the makefiles act as defaults that may be overridden from the command line.



-e Rebuild everything that is a part of the specified or default target, regardless of whether targets are out-of-date. This option causes Make to unconditionally generate all of the commands to rebuild the specified targets.

Note: This option causes all components of the specified target to be rebuilt. However, it does not necessarily rebuild all targets if there are more than one top-level targets (roots) in the makefile.

-f *makefile* Read dependency information from *makefile*. You can specify more than one **-f** option—all dependency information is treated as if it were in a single file. (If no **-f** option is specified, the default file is a file named MakeFile in the current directory.)

-p Write progress information to diagnostic output. (Normally, Make runs silently unless errors are detected.)

-r [*target*] If no target is specified, the **-r** option finds all the roots (that is, the top-level targets) of the dependency graph. (See the **-s** option.) If a target is specified, **-r** finds the root (or roots) for which it is a prerequisite.

Note: This option overrides normal Make output.

-s Show structure of target dependencies. This option writes a dependency graph for the specified targets to standard output, using indentation to indicate levels in the dependency tree. Circular dependencies are noted, if present.

Note: This option overrides the normal Make output. It's useful in debugging or verifying complicated makefiles.

-t "Touch" dates of targets and their prerequisites; that is, bring files up-to-date by adjusting their modification dates, without generating build commands. This option is used to bring a set of files up-to-date when they appear not to be, such as when you've only made changes to comments. The **-t** option does the minimal adjustment needed to satisfy the dependency relationships; files are touched only if required and are given the date of their newest dependency, to minimize any repercussions of the date adjustments. This minimal adjustment of dates is especially useful if the touched file is also a prerequisite for other programs.

Note: This option overrides normal Make output.

-u Write a list of unreachable targets to diagnostic output (for debugging). Unreachable targets are those mentioned in the makefile that are not prerequisites (or prerequisites of prerequisites) of the specified target to be rebuilt.

-v Write verbose output to the diagnostic output file. This option is useful if you want to figure out what Make is doing and why. The diagnostic output indicates if targets do not exist, whether they need to be rebuilt, and why they need to be rebuilt. It also indicates targets in the makefile that were not reached in the build.

-w Suppress warning messages. Warning messages are issued for things such as files with dates in the future and circular dependency relationships.

Example

```
Make -p -f MakeFile Sample
```

Makes the target file Sample, and prints progress information. Sample's dependency relations are described in the makefile :AExamples:MakeFile.

```
Sample      ƒƒ Sample.r
     Rez Sample.r -o Sample -a
     SetFile -a B Sample -c ASMP -t APPL #set bundle bit

Sample      ƒƒ Sample.r Sample.a.o
     Link Sample.a.o -o Sample

Sample.a.o  ƒ  Sample.a
     Asm Sample.a
```

The ƒ (Option-F) character means "is a function of"—that is, the file on the left side depends on the files on the right side. If the files on the right are newer, the subsequent Shell commands are written to standard output. (See Chapter 9 for details.)

See also

"Format of a Makefile" in Chapter 9 for the format of a makefile, examples, and other information about using Make.

Makefiles for building sample programs are contained in the Examples folders:

- Examples:AExamples:Makefile
- Examples:PExamples:Makefile
- Examples:CExamples:Makefile

MakeErrorFile—create error message file

Syntax MakeErrorFile [option...] [file...]

Description MakeErrorFile creates specially formatted error message files used to retrieve the error messages associated with error numbers. The ErrMgr unit in the ToolLibs.o library is used by programs to access the error files created by MakeErrorFile. SysErrs.Err is one such error file; it is used by various MPW tools to get the textual messages associated with Macintosh system error codes. See the documentation on the ErrMgr unit for more information on how error files are accessed.

Type Tool.

Input Standard input is processed if no filenames are specified. Otherwise each file in the MakeErrorFile invocation is processed separately, with an error file created for each input. MakeErrorFile input files follow a very simple format, consisting of lines associating error messages with error numbers. Each line begins with an error number (in the range of 2-byte signed integers), followed by a space, followed by the corresponding error message text on the same line.

Output If the **-l** listing option is specified, an ordered list of error numbers and messages is written to standard output. The error file output is usually written to a file with the same name as the input file but with an ".Err" extension (unless the **-o** option was used to specify the output name). By default, if no input file was specified, the input comes from standard input and the default error output filename is "Out.Err".

Diagnostics Errors and warnings are written to diagnostic output.

Status The following status codes may be returned:

0 No errors.
1 Syntax error.
2 Error in processing.

Options

-l	Write an ordered list of error numbers and messages to standard output.
-o *objname*	Pathname for the generated error file if *objname* is a full pathname. If *objname* is a directory, it specifies where to put the error output file.
-p	Write progress information to diagnostic output.

Example

```
MakeErrorFile SysErrs -l >SysErrsList
```

Writes an ordered list of system error numbers and messages to the file SysErrsList.

Mark—assign a marker to a selection

Syntax

Mark [*-y* | *-n*] *selection name* [*window*]

Description

Mark assigns the marker *name* to the range of text specified by the *selection* in *window*. If no window is specified, the command operates on the target window (the second window from the front). The new marker name is included in the Mark menu when *window* is the current active window. A marker is associated with a logical, as opposed to absolute, range of text. The ranges of markers may overlap, but each marker must have a unique name. Marker names are case sensitive.

A dialog box requests confirmation if the marker name conflicts with an existing marker name. The **-y** or **-n** option can be used in scripts to avoid this interaction.

Deletion and insertion operations affect markers according to these rules:

- Any editing outside the range of a marker will not affect the logical range of the marker, where "outside" means that the range of editing changes does not intersect the range of the marker.
- Any editing inside the range of a marker will change the logical range of the marker by the amount of the editing change. For example, adding ten characters to the inside of a marker's range will increase the range of the marker by ten characters. Another way to say this is that a marker has responsibility for all the characters added to (or deleted from) its range.
- Any deletion that totally encloses a marker will delete the marker.

Type

Built-in.

Input

None.

Output

None.

Diagnostics

Errors are written to diagnostic output.

Status

The following status codes may be returned:

0 No errors.
1 Syntax error.
2 Error in processing.
3 System error.

Options

-y	Answer "Yes" to any confirmation dialog that occurs, causing the old marker to be replaced with the new marker.
-n	Answer "No" to any confirmation dialog that occurs, so that the old marker is left intact.

Example

```
Mark § 'Procedure 1'
```

Assigns a marker with the name "Procedure 1" to the current selection in the target window.

Limitation

It is currently not possible to "Undo" the effects of any editing operations on markers.

See also

Unmark and Markers commands.

"Mark Menu" in Chapter 3.

"Markers" in Chapter 6.

Markers—list markers

Syntax Markers [-q] [*window*]

Description Markers prints the names of all markers associated with *window*. The names are written one per line, and they are ordered from the beginning to the end of the window.

Type Tool.

Input None.

Output The list of marker names is written to standard output.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

0 No errors.
1 Syntax error.
2 Error in processing.
3 System error.

Options

-q Do not quote marker names that contain special characters. (The default is to quote names with spaces or other special characters.)

Example

```
Markers "{Target}"
```

Lists all markers associated with the target window.

See also "Mark Menu" in Chapter 3.

"Markers" in Chapter 6.

MatchIt—match paired language delimiters

Syntax MatchIt [-h] [-l] [-n] [-v] [[-c] | [-p[ascal]] | [-a[sm]]] [*window*]

Description Matches C, Pascal, and assembly-language delimiters with their mates in the specified window. The default *window* is the target window (second from front). The characters highlighted as the current selection[1] in the window are used as the left delimiter. MatchIt attempts to find the corresponding right mate for the selected delimiter.

MatchIt is syntax sensitive, so that it is semi-intelligent about how it finds the matching delimiter. For example, if a Pascal BEGIN is the specified selection, MatchIt finds the proper END that matches it. All commenting conventions, strings, nesting, and so on are taken into account when searching for the end delimiter.

The functionality of MatchIt is similar to the Shell editor's ability to find mates for the characters ([{ " and ' when you double-click any of these characters. However, MatchIt differs from the Shell editor by supporting even more delimiters and using the knowledge of the target language syntax to find the proper match. The following table summarizes all the delimiters supported and for which languages:

Left delimiter	Right delimiter	Language(s)
{	}	Pascal, C, Asm
[	]	Pascal, C, Asm
(	)	Pascal, C, Asm
'	'	Pascal, C, Asm
"	"	C, Asm
/*	*/	C
//	cr	C
do	while	C
#if	#endif	C
#ifdef	#endif	C
#ifndef	#endif	C
#elif	#endif	C
#else	#endif	C
(*	*)	Pascal
BEGIN	END	Pascal

[1] Leading and trailing blanks are ignored in the selection.

(MatchIt delimiters continued)

Left delimiter	Right delimiter	Language(s)
REPEAT	UNTIL	Pascal
CASE	END	Pascal
RECORD	END	Pascal
$IFC[2]	$ENDC	Pascal
$ELSEC[2]	$ENDC	Pascal
RECORD	ENDR	Asm

The language can be explicitly specified by option (**-p**, **-c**, or **-a**), in which case only the left delimiters in the above table appropriate to that language are accepted. The normal situation is for MatchIt to determine the language from the window name suffix; ".p " for Pascal, ".a " for Asm, and ".c", ".h", or ".r" (Rez) for C. If there is no suffix and no explicit language option, MatchIt attempts to determine the language on the basis of the selected left delimiter. For the ambiguous cases, C is assumed.

Type

Tool.

Input

None.

Output

None. If the matching delimiter is found, only the current selection in the specified window is changed. Normally only the matched right delimiter becomes the new current selection. However, there is some additional control over the resulting selection with the **-h** and **-l** options. If a match cannot be found, the original selection is not changed.

Diagnostics

Errors are written to diagnostic output.

Status

MatchIt may return the following status codes:

0 Normal termination.
1 Parameter or option error.
3 Matching delimiter not found (only if **-n** option specified).

[2] These delimiters are unique in that they occur in Pascal comments. MatchIt treats them specially by allowing you to optionally select the leading "{" or "(*". If you don't, MatchIt will include them as part of the selection itself. If you select the entire comment, MatchIt will highlight the entire matching {$ENDC} comment.

Options

-a[sm]	The target language is assumed to be assembler. Only left delimiters defined in the above table for the assembler are allowed as the selection.
-c	The target language is assumed to be C. Only left delimiters defined in the above table for C are allowed as the selection.
-h	If the match is found, *all* characters starting from the original selection up to the matching delimiter are highlighted and made the current selection.
-l	If a match is found, the *entire line* containing the matching delimiter is made the new current selection. If this option is used together with the **-h** option, all lines—starting with the line containing the left delimiter up to the line containing its mate—are highlighted and made the new current selection.
-n	Generate an error message to diagnostic output when a match cannot be found. Normally, no message is generated and the returned status code is 0. This is usually sufficient because the current selection is not changed. However, this option can be useful in Shell scripts and AddMenu commands.
-p[ascal]	The target language is assumed to be Pascal. Only left delimiters defined in the above table for Pascal are allowed as the selection.
-v	Write MatchIt's version number to diagnostic output.

Examples

```
matchit  mysource.p
```

For the current selected delimiter in the open window mysource.p, find the delimiter's mate. The language is assumed to be Pascal (because of the .p suffix.). No message is reported and the selection is not changed if the mate cannot be found. Of course, errors are still reported to diagnostic output if the input selection is not a valid Pascal delimiter (according to the table in "Options"). If MatchIt is to be used explicitly, a more general form for its use is shown in this example:

```
matchit  -n  "{target}"
```

For the current selected delimiter in the open target window, find the delimiter's mate. The "{target}" specification could have been omitted, as it is MatchIt's default. If explicitly specified, as shown here, it is best to quote it. The language is determined by the window's name suffix (if present), or by the the selection, if the suffix is not acceptable to MatchIt. An error is reported if the mate cannot be found (**-n**).

While the second example is more general than the first, and either might be useful for Shell scripts (particularly when the **-n** option is used), the *real* use for MatchIt is as a generalization of the the Shell editor's own double-clicking delimiter matching mechanism. The following example illustrates this purpose:

```
addmenu Edit 'Match It/μ' 'matchit  -n -h  ∂
{active}"  ≥ "{MPW}"Errors || ∂
alert  < "{MPW}"Errors'
```

This example places a MatchIt call into the Edit menu as the command Match It with a command key Option-m (the μ). A selection is made in the current (that is, the {active}) window and the menu command invoked (by pressing Command-Option-m). If the match is found, all characters from the initially selected delimiter to its mate are highlighted (**-h**). If a match is not found, or if any other errors occur, an alert dialog box appears containing the error message. An auxiliary file, "{MPW}"Errors, is used for this purpose.

Of course, you might not be interested in displaying the dialog box because you can see that the selection doesn't change if there are any errors. Furthermore, you might not want superfluous files laying around ("{MPW}"Errors—although you could create a more elaborate AddMenu command to always delete this file). Thus, you could make the following simplification:

```
addmenu Edit 'Match It/μ' 'matchit  -h  "{active}"  ≥ dev:null'
```

This example places a MatchIt call into the Edit menu but with all errors ignored when the MatchIt command is executed.

Limitations

MatchIt does *not* process conditionals (that is, Pascal `$ifc`, C `#if`, and so on) during its scan except to find matching pairs. This might confuse MatchIt's scanning process. Similarily, C macros and "\" continuations may also confuse MatchIt.

MatchIt only finds a right delimiter to the specified left delimiter. Right-to-left matching is not supported.

MergeBranch—merge a branch revision onto the trunk

Syntax MergeBranch *file*

Description Merge the branch revision of the HFS file *file* onto the trunk. The file must belong to a currently mounted project and must be a branch revision (that is, the revision number contains one or more letters).

MergeBranch uses the ProjectInfo command to determine what project *file* belongs to and whether *file* is in fact a branch revision. If all of the file's revisions are older than the branch, the branch will be checked in as the latest trunk revision. Otherwise MergeBranch checks out the latest revision on the trunk and calls CompareFiles to allow the user to manually cut and paste changes from the branch into the trunk revision. When done, the user can check the modified trunk revision back into the project.

MergeBranch uses the CompareFiles script.

Type Script.

Input None.

Output None.

Diagnostics Errors and warnings are written to diagnostic output.

Status The following status codes may be returned:

0 No Errors.
1 Syntax Error.
2 Error in Processing.
3 System Error.

Options None.

Examples

```
MergeBranch file.c
```

This example merges the branch revision in the file "file.c" onto the trunk.

```
AddMenu Project 'Merge Branch' 'Merge Branch "{Active}" ΣΣ ∂
"{WorkSheet}"'
```

This example adds MergeBranch to the Project menu and allows you to merge branch revisions onto the trunk.

See Also CompareFiles.

ModifyReadOnly—allow editing of a read-only file

Syntax ModifyReadOnly *filename*

Description Write-enable a file that has been checked out as read-only. After executing this command on a file, the modified read-only icon is displayed in the window.

This command is most useful on those rare occasions when you need to modify a read-only file. For example, suppose you have taken a number of modifiable files home. You may have also brought along certain read-only copies of files that you did not expect to modify, but once you get into your work at home you discover that you do, after all, need to make changes in these files.

Note that this command takes only a single file for a parameter. This "feature" was intentional so that this command would not be overused.

See Chapter 7 for complete definitions of the terms and symbols used in Projector commands.

Type Built-in.

Input None.

Output None.

Diagnostics Errors and warnings are written to diagnostic output.

Status The following status codes may be returned:

0 No errors.
1 Syntax error.
2 Error in processing.

Options None.

Examples

Suppose file.c is checked out as read-only. You can write-enable it by using the ModifyReadOnly command:

```
ModifyReadOnly file.c
ProjectInfo :file.c -s
```

The ProjectInfo command writes the following to standard output:

```
file.c,5*
```

Notice that an asterisk appears after the revision number when you get information about modified read-only files.

See Also

CheckIn, CheckOut, CheckOutDir.

Mount—mount volumes

Syntax Mount *drive...*

Description Mounts the disks in the specified drives, making them accessible to the file system. *Drive* is the drive number.

Mounting is normally automatic when a disk is inserted. The Mount command is needed for mounting multiple hard disks, which cannot be "inserted," or for volumes that have been unmounted via the Unmount command.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

0 The disk was mounted.
1 Syntax error.
2 An error occurred.

Options None.

Example

```
Mount 1
```

Mounts the disk in drive 1 (the internal drive).

See also Unmount and Volumes commands.

MountProject—mount an existing project

Syntax MountProject [-s] [-pp] [-q] [-r] [*project*]

Description MountProject mounts (establishes a connection to) the specified project. *Project* is the HFS path of the project directory for the project. Once a project is mounted, that project and all its subprojects can be accessed.

MountProject commands typically appear in the UserStartup file, a script, or an AddMenu to automatically mount the projects you typically access.

If *project* is omitted, a list of all root projects is written to standard output in the form of MountProject commands.

See Chapter 7 for complete definitions of the terms and symbols used in Projector commands.

Type Built-in.

Input None.

Output If no parameters are given, MountProject generates a list of root projects.

Diagnostics Errors and warnings are written to diagnostic output.

Status These status codes may be returned:

0 No errors.
1 Syntax error.
2 Error in processing.
3 System error.

Options

-pp List mounted projects using project paths.

-q Don't quote names with special characters.

-r List projects recursively.

-s Print names only, not commands.

Examples

```
MountProject  FS:Zoom
MountProject  HD:localProjects:Test
```

These commands mount the projects Zoom and Test.

```
MountProject
MountProject  FS:MPW
MountProject  HD:localProjects:sort
```

To obtain a list of the current root projects, execute the MountProject command without parameters.

See Also UnmountProject, Project, CheckOutDir.

Move—move files and directories

Syntax Move [-y | -n | -c] [-p] *name... targetName*

Description Moves *name* to *targetName*. (*Name* and *targetName* are file or directory names.) If *targetName* is a directory, one or more objects (files and/or directories) are moved into that directory. If *targetName* is a file or doesn't exist, file or directory *name* replaces *targetName*. In either case, the old objects are deleted. Moved objects retain their current creation and modification dates.

If a directory is moved, its contents, including all subdirectories, are also moved. No directory moved can be a parent of *targetName*.

A dialog box requests a confirmation if the move would overwrite an existing file or folder. The **-y, -n,** or **-c** option can be used to avoid this interaction.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output. Progress and summary information is also written to diagnostic output if the **-p** option is specified.

Status Move may return the following status codes:

0	All objects were moved.
1	Syntax error.
2	An error occurred during the move.
4	Cancel was selected or implied with the **-c** option.

Options

-y	Answer "Yes" to any confirmation dialog that may occur, causing conflicting files or folders to be overwritten.
-n	Answer "No" to any confirmation dialog that may occur, skipping the move for files or folders that already exist.
-c	Answer "Cancel" to any confirmation dialog that may appear, causing the move to stop when a name conflict is encountered.
-p	List progress information as the move takes place.

Examples

```
Move Startup Suspend Resume Quit "{SystemFolder}"
```

Moves the four files from the current directory to the System Folder.

```
Move File ::
```

Moves File from the current directory to the enclosing (parent) directory.

```
Move -y File1 File2
```

Moves File1 to File2, overwriting File2 if it exists. (This is the same as renaming the file.)

See also

Duplicate and Rename commands.

"File and Window Names" in Chapter 4.

"Filename Generation" in Chapter 5.

MoveWindow—move window to h v location

Syntax MoveWindow [-i] [**h v**] [*window*]

Description Moves the upper-left corner of the specified *window* to the location (**h v**), where **h** and **v** are horizontal and vertical integers, respectively. Use a space to separate the numbers **h** and **v** on the command line.

The coordinates (0,0) are located at the left side of the screen at the bottom of the menu bar. If the location specified would place the window's title bar entirely off the visible screen, an error is returned. (The **-i** option overrides the error.) If no window is specified, the target window (the second window from the front) is assumed. If no location is specified, the specified window's location is returned without any effect on the window.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status MoveWindow may return the following status codes:

0 No errors.
1 Syntax error (error in parameters).
2 The specified window does not exist.
3 The **h v** location specified is invalid.

Option **-i** Ignore any errors relating to the window's position. This option allows a window to be completely off-screen. However, the position must still be within the range of -32,768 to 32,767.

Examples

```
MoveWindow 72 72
```

Moves the target window's upper-left corner to a point approximately one inch in from the upper-left corner of the screen, and one inch below the bottom of the menu bar. (There are about 72 pixels per inch on the Macintosh display screen.)

```
MoveWindow
```

Returns MoveWindow 72 72 when executed after the above example.

```
MoveWindow 0 0 "{Worksheet}"
```

Moves the Worksheet window to the upper-left corner of the screen (below the menu bar).

See also

SizeWindow, StackWindows, RotateWindows, TileWindows, and ZoomWindow commands.

NameRevisions—name files and revisions

Syntax

NameRevisions [-u *User*] [-project *Project*] [-public | -b] [-r]
[[-only] | *name* [[-expand] [-s] [-replace] | [(*names*... [-dynamic]) | [-a]]]]

Description

Create a symbolic *name* to represent a set of revisions under Projector. Subsequently, when *name* is used in Projector commands, its value, *names*, is substituted in its place. Symbolic names are kept on a per-project basis and can be composed of filenames, revisions, branches, and other defined symbolic names. A symbolic name can include only one revision per file. The first character of a Name cannot be a digit (0–9). Also, commas, greater-than or less-than signs, (<, ≤, > ≥), or hyphens (-) are not allowed anywhere in a Name. Names are not case sensitive.

If *names* is missing, the definition for *name* is listed. If *name* is missing, then NameRevisions lists all symbolic names in the project. In either case, the output is in the form of NameRevisions commands.

By default, if *names* currently refers to a file listed in *name*, the revision for the file in *name* is modified to be the revision associated with the file in *names*. If there is a file in *names* which is not currently referred to by *name*, that file and revision is appended to *name*. To replace the definition of *name*, include the **-replace** option.

The default is to create a private symbolic name. Include the **-public** option to make the symbolic name available to all users. You can add definitions for private symbolic names to UserStartup. Public symbolic name are stored with the project so they need to be defined only once. Do not put public symbolic name definitions in UserStartup.

Projector checks for various errors both when a symbolic name is defined and when it is used. Errors include referring to a nonexistent file or referring to more than one revision in a file.

See Chapter 7 for complete definitions of the terms and symbols used in Projector commands.

Type

Built-in.

Input

None.

Output

When *name* or *names* are missing, the command writes symbolic names and their values to standard output.

Diagnostics Errors and warnings are written to diagnostic output.

Status The following status codes may be returned:

0 No errors.
1 Syntax error.
2 Error in processing.
3 System error.

Options

-u *user* Name of the current user. This overrides the {User} Shell variable.

-project *project*
Name of the project in which to create this name. This becomes the current project for this command.

-public Create a public symbolic name . This lets all users in the project have access to the name. Without this option a private symbolic name is defined.

-b Print both public and private names.

-r Recursively execute the NameRevisions command on the current project and all its subprojects.

-only List every *name* defined in the current project, but not their associated *names.*

-expand Evaluate and expand Names and files to the revision level when listing values if *name* or *names* is missing.

-s Print a single name per line.

-replace Replace the current definition of name with a new definition.

-dynamic Evaluate and expand symbolic names and files to the revision level when Name is used—not when it is defined.

-a All files in the project. The symbolic name expands to all the files in the project.

Examples Assuming the latest revisions of the files file.c and interactive.c are 9 and 13 respectively, the first example defines a symbolic name "Work" that always expands to the files file.c,9 and interactive.c,13.

```
NameRevisions Work file.c interactive.c
```

The following command:

```
CheckOut Work
```

Is equivalent to:

```
CheckOut file.c,9 interactive.c,13
```

By omitting the Names parameter, the next NameRevisions command generates the current definition of Work.

```
NameRevisions Work
NameRevisions Work file.c,9 interactive.c,13
```

The **-dynamic** is an important option. The following two commands illustrate its function:

```
NameRevisions fred file.c
NameRevisions -dynamic fred file.c
```

The first command defines a symbolic name "fred" that always expands to the latest revision of file.c when fred was defined. The second example expands to the latest revision at the time of use. If the latest revision of file.c at the time fred was defined was 7 and the current latest revision is 9, the second NameRevisions command is equivalent to

```
NameRevisions fred file.c,9
```

The next command creates the symbolic name "file.c" that expands to the second revision off the first branch off the 1.1 revision of file.c.

```
NameRevisions file.c file.c,1.1a2
```

The command

```
CheckOut file.c
```

checks out revision 1.1a2 of file.c. The next example creates a Name "file.c" that expands to the latest version of the first branch off the 1.1 revision of file.c.

```
NameRevisions -dynamic file.c file.c,1.1a
```

So the checkout command

```
CheckOut file.c
```

will check out the latest revision on the first branch off revision 1.1 of file.c.

The next example defines all the latest revisions in the project Kerfroodi to be part of "v1.0 B1". Because this a global name, all users accessing the Kerfroodi project will be able to use the name "v1.0 B1".

```
NameRevisions -public "vB1 1.0" -project Kerfroodi -a
```

The name "BetaRelease" is defined recursively for all projects within the Zoom project:

```
NameRevisions -project Zoom∫ -r "BetaRelease" -a
```

Its behavior is the same as executing the following commands individually:

```
NameRevisions -project Zoom "BetaRelease" -a
NameRevisions -project Zoom∫Vroom "BetaRelease" -a
NameRevisions -project Zoom∫Utilities "BetaRelease" -a
NameRevisions -project Zoom∫Utilities∫Port "BetaRelease" -a
...
```

See Also ProjectInfo, DeleteNames.

New—open a new window

Syntax New [*name...*]

Description Opens a new window as the active (frontmost) window. If *name* is not specified, the Shell generates a unique name for the new window, of the form "Untitled-*n*", where *n* is a decimal number. If *name* already exists, an error results.

You can use New to open several new windows by specifying a list of names separated by spaces. Note that New differs from Open **-n** by returning an error if the file already exists, whereas Open **-n** either opens an existing file or creates a new file.

If the Shell variable {NewWindowRect} is defined, the windows are opened to that size and location.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status New may return the following status codes:

0 No errors.
1 Syntax error (error in parameters).
2 Unable to complete operation; a file with the specified name already exists.
3 System error.

Options None.

Examples

```
New
```

Opens a new window with a Shell-generated name.

```
New Test.a Test.p Test.c
```

Creates three windows called Test.a, Test.p, and Test.c.

See also Open command.

Newer—compare modification dates between files

Syntax Newer [-e] [-c] [-q] *name... target*

Description Compares the modification dates of *name* and *target*. Files that have a more recent modification date than *target* have their names written to standard output. If the target is a nonexistent file or directory, all names that exist are considered newer than the target.

Type Built-in.

Input None.

Output Newer files are written to standard output. The names are written out one per line as they appear on the command line.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

0 No error.
1 Syntax error.
2 File not found.

Options

-c Compare creation dates instead of modification dates.

-e Look for files with equal modification dates (or creation dates when used with the **-c** option).

-q Do not quote pathnames that are written to standard output.

Examples

```
Newer main.c main.c.bak
```

Writes out main.c if its modification date is more recent than its backup.

```
Newer HD:Source:≈.c HD:TimeStamp
```

Writes to the screen all the source files in the Source directory that have been modified since the modification date of TimeStamp.

```
If `Newer main.c main.c.bak`
    Duplicate main.c main.c.bak
End
```

Makes a backup copy of main.c only if it has been modified since the last backup was made.

```
If   "`Newer File.c  File.h  File.c.o`"
    C File.c  -o  file.c.o
End
```

Rebuilds the source file file.c if either file.c or file.h has been modified since file.c.o was last built.

See also Exists command.

NewFolder—create a directory

Syntax NewFolder *name...*

Description Creates new directories with the names specified. Any parent directories included in the *name* specification must already exist.

- *Note:* This command can be used only on hierarchical file system (HFS) disks.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

0 Folders were created for each name listed.
1 Syntax error.
2 An error occurred.
3 Attempt to use NewFolder on a non-HFS volume.

Options None.

Examples

```
NewFolder Memos
```

Creates Memos as a subdirectory of the current directory.

```
NewFolder Parent :Parent:Kid
```

Creates Parent as a subdirectory of the current directory, and Kid as a subdirectory of Parent.

NewProject—create a project

Syntax NewProject -w | -close | ([-u *user*] [-cs *comment* | -cf *file*] *project*)

Description NewProject creates a project under control of Projector. A project directory is created to store the files, subprojects, and other information related to the project. The name of the directory is the name of the project.

If *project* is a project pathname (such as MPW∫Tools∫Enterprise), Projector creates Enterprise as a subproject of the existing MPW∫Tools project. In this case MPW∫Tools must be a mounted project (see the MountProject command).

If *project* is a leafname (such as Enterprise), project directory Enterprise is created in the current directory.

Finally, if *project* is a partial or full HFS pathname (such as :Work:Enterprise or FS:Projects:Enterprise), the project Enterprise is created in the HFS location specified.

Add a MountProject command to the UserStartup file, a script, or AddMenu to easily mount the new project.

The checkout directory is initially set to the current directory (:). To change the checkout directory, refer to the CheckOutDir command.

To add files to the new project, use the CheckIn command (with the **-new** option) or the Check In window.

See Chapter 7 for complete definitions of the terms and symbols used in Projector commands.

Type Built-in.

Input None.

Output None.

Diagnostics Errors and warnings are written to diagnostic output.

Status

The following status codes may be returned:

0 No errors.
1 Syntax error.
2 Error in processing.
3 System error.

Options

-w Open the New Project window.

-close Close the New Project window.

-u *user* Name of the current user. This overrides the {User} Shell variable.

-cs *comment*
A short description of the project.

-cf *filename* The comment is contained in the file *filename.*

Examples

The following command creates a project Enterprise in the current directory. No comment is saved with the project, but you can add one later by selecting the project in the Check Out window's /Info view.

```
NewProject Enterprise
```

The next example creates a project Zoom in the FS:work:Zoom. The **-cf** option indicates that the comment for the new project is contained in the file Info.

```
NewProject FS:work:Zoom -cf Info
```

Finally, given that the project Enterprise∫Utilities exists and has been mounted using the MountProject command, the next command creates a Zoom project in the Enterprise∫Utilities project. In this case you don't need to add a MountProject command to UserStartup, but you may want to add a CheckOutDir command to set the checkout directory.

```
NewProject Enterprise∫Utilities∫Zoom -cs ∂
"Upgrade Zoom utility"
```

See Also

CheckOutDir, MountProject, Project.

Open—open a window

Syntax Open [-n | -r] [-t] [*name...*]

Description Opens a file as the active (frontmost) window. If *name* is not specifed, StdFile's GetFile routine is called, allowing you to use a dialog box to choose a file. If *name* is already open as a window, that window becomes the active (frontmost) window.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status Open may return the following status codes:

0 No errors.
1 Error in parameters.
2 Unable to complete operation; specified file not found.
3 System error.

Options

-n Open a new window with the title *name*. If file *name* already exists, that file is opened.

-r Open a read-only window associated with the file *name*. If the file *name* doesn't exist, an error occurs.

-t Open the window as the target window rather than as the active window (that is, make it the second window from the front). This option is identical to the Target command.

Examples

```
Open
```

Displays StdFile from which to choose a file to open.

```
Open -r -t Test.a
```

Opens the file Test.a as the target window, read-only.

```
Open ≈.a
```

Opens all the files that end with ".a".

See also Target, New, and Close commands.

OrphanFiles—remove projector info from files

Syntax OrphanFiles *filename...*

Description Remove the 'CKID' resource from file(s). This removes the identification information from the file that Projector uses to uniquely identify it.

▲ **Warning** Once the projector information is removed from a file, you cannot check the file back into the Project as a checked-out file. ▲

See Chapter 7 for complete definitions of the terms and symbols used in Projector commands.

Type Script.

Input None.

Output None.

Diagnostics Errors and warnings are written to diagnostic output.

Status The following status codes may be returned:

0 No errors.
1 Syntax error.
2 Error in processing.

Options None.

Example Suppose file.c and interactive.c belong to a project that has been deleted. We can remove the Projector information from them (so that they can be used for other purposes) with the command

```
OrphanFiles file.c interactive.c
```

See Also TransferCkid.

Parameters—write parameters

Syntax Parameters [*parameter* ...]

Description The Parameters command writes its parameters, including its name, to standard output. The parameters are written one per line, and each is preceded by its parameter number (in braces) and a blank. This command is useful for checking the results of variable substitution, command substitution, quoting, blank interpretation, and filename generation.

Type Built-in.

Input None.

Output Parameters are written to standard output.

Diagnostics None.

Status A status code of 0 (no problem) is always returned.

Options None.

Example

```
Parameters One Two "and Three"
```

Writes the following four lines to standard output:

```
{0} Parameters
{1} One
{2} Two
{3} and Three
```

Recall that "..." and '...' quotation marks are removed before parameters are passed to commands.

See also Echo and Quote commands.

"Parameters to Scripts" in Chapter 5.

Pascal—Pascal compiler

Syntax Pascal [*option* ...] [*file* ...]

Description Compiles the specified Pascal source files (programs or units). You can specify zero or more filenames. Each file is compiled separately—compiling file *Name* .p creates object file *Name* .p.o. By convention, Pascal source filenames end in a ".p" suffix.

See the *MPW 3.0 Pascal Reference* for details of the language definition.

Type Tool.

Input If no filenames are specified, standard input is compiled, with output directed to the file `p.o`. You can terminate input by pressing Command-Enter.

Output Nothing is written to standard output. For each input file *name*, object code is sent to the file *name* .o.

Diagnostics Errors are written to diagnostic output. Progress and summary information is also written to diagnostic output if the **-p** option is selected.

Status The following status codes may be returned:

0 Successful completion.
1 Error in parameters.
2 Compilation halted.

Options

-b Generate A5-relative references whenever the address of a procedure or function is taken. (By default, PC-relative references are generated for routines in the same segment.)

-c Syntax check only; no object file is generated.

-clean Erase all symbol table references.

-d *name* =TRUE | FALSE
Set the compile time variable *name* to TRUE or FALSE.

-e *errLogFile*
Write all errors to the error log file *errLogFile*. A copy of the error report is still sent to diagnostic output.

-h Suppress error messages regarding the use of unsafe handles.

-m Allow greater than 32K globals by using 32-bit references.

-i *pathname* [, *pathname*]...

Search for `include` or `USES` files in the specified directories. Multiple **-i** options may be specified. At most, 15 directories are searched. The search order is

1. In the case of a `USES` filename, if no prior `$U` filename was specified, the filename is assumed to be the same as the unit name (with a ".p" appended).
2. The filename is used as specified. If a *full pathname* is given, no other searching is applied.

   If the file isn't found, and the pathname used to specify the file is a *partial pathname* (no colons in the name or a leading colon), the following directories are searched:
3. The directory containing the current input file.
4. The directories specified in **-i** options, in the order listed.
5. The directories specified in the Shell variable {PInterfaces}.

The source filenames specified on the command line must include any relevant prefixes.

-mbg ch8 Include V2.0-compatible MacsBug symbols (eight characters only, in a special format).

-mbg full Include full (untruncated) symbols for MacsBug.

-mbg off Don't include symbols for MacsBug.

-mbg *number*

Include MacsBug symbols truncated to length *number.*

-mc68020 Generate code to take advantage of the MC68020 processor.

-mc68881 Generate code to take advantage of the MC68881 coprocessor.

-n Generate separate global data modules for better allocation.

-noload Don't use or create any symbol table resources.

-o *objName*

Specify the pathname for the generated object file. If *objName* ends with a colon (:), it indicates a directory for the output file, whose name is then formed by the normal rules (that is, *inputFilename* .o). If the source filename contains a pathname, it is stripped off before *objName:* is used as a prefix. If *objName* does not end with a colon, the object file is written to the file *objName.* (In this case, only one source file should be specified.)

-ov Turn on overflow checking.

▲ **Warning** Doing so may significantly increase code size. ▲

-p Supply progress and summary information to diagnostic output, including Compiler header information (copyright notice and version number), module names and code sizes in bytes, and number of errors and compilation time.

-r Suppress range checking.

-rebuild Rebuild all symbol table reference.

-sym off Don't generate SADE object file information.

-sym on | full

Generate complete SADE object information. You can limit this option by also specifying one or more **nolines**, **novars**,and **notypes**. These cause omission of line, variable, and type information, respectively, from the object file.

-t Report compilation time to diagnostic output. The **-p** option also reports the compilation time.

-u Initialize local and global data to the value $7267 (for debugging use).

-w Turn off peephole optimizer.

-y *pathname*

Put the compiler's temporary intermediate (".o.i") files in the directory specified by *pathname.*

Examples

```
Pascal Sample.p
```

Compiles the Sample program provided in the PExamples folder.

```
Pascal File1.p File2.p -r
```

Compiles File1.p and File2.p, producing object files File1.p.o and File2.p.o but performing no range checking.

- *Note:* Listing files are not produced directly by the compiler. Refer to the PasMat and PasRef tools.

Availability

The MPW Pascal compiler is available as a separate Apple product.

See also

PasMat and PasRef commands.
MPW 3.0 Pascal Reference.

PasMat—Pascal program formatter

Syntax PasMat [*option*...] [*inputfile* [*outputfile*]]

Description Reformats Pascal source code into a standard format, suitable for printouts or compilation. PasMat accepts full programs, external procedures, blocks, and groups of statements.

◆ *Note:* A syntactically incorrect program causes PasMat to abort. If this happens, the generated output will contain the formatted source up to the point of the error.

PasMat options let you do the following:

- Convert a program to uniform case conventions.
- Indent a program to show its logical structure, and adjust lines to fit into a specified line length.
- Change the comment delimiters `(*  *)` to `{ }`.
- Remove the underscore character (_) from identifiers, rename identifiers, or change their case.
- Format `include` files named in MPW Pascal `include` directives.

PasMat specifications can be made through PasMat options or through special formatter directives, which resemble Pascal compiler directives, and are inserted into the source file as Pascal comments. PasMat's default formatting is straightforward and does not require you to use any options. The best way to find out how PasMat formats something is to try out a small example.

See Appendix K of the *MPW 3.0 Pascal Reference* for details of PasMat directives and their functions. The first dialog box of the Pascal Commando dialog is reproduced here for your convenience.

Type Tool.

Input If no input files are specified, standard input is formatted.

Output If no output file is specified, the formatted output is written to standard output. Refer to "Limitations" below for more information about PasMat's treatment of errors in the source.

Diagnostics

The following errors are detected and written to diagnostic output:

- In general, premature end-of-file conditions in the input are not reported as errors, in order to accommodate formatting of individual `include` files, which may be only program segments. There are cases, however, where the `include` file is a partial program, which PasMat interprets and reports as a syntax error.
- There is a limit on the number of indentation levels that PasMat can handle. If this limit is exceeded, processing will abort. This problem should be exceedingly rare.
- If a comment would require more than the maximum output length (150) to meet the rules given, processing will abort. This problem should be even rarer than indentation level problems.

If a syntax error in the input code causes formatting to abort, an error message gives the input line number on which the error was detected. The error checking is not perfect—successful formatting is no guarantee that the program will compile.

Status

PasMat may return the following status codes:

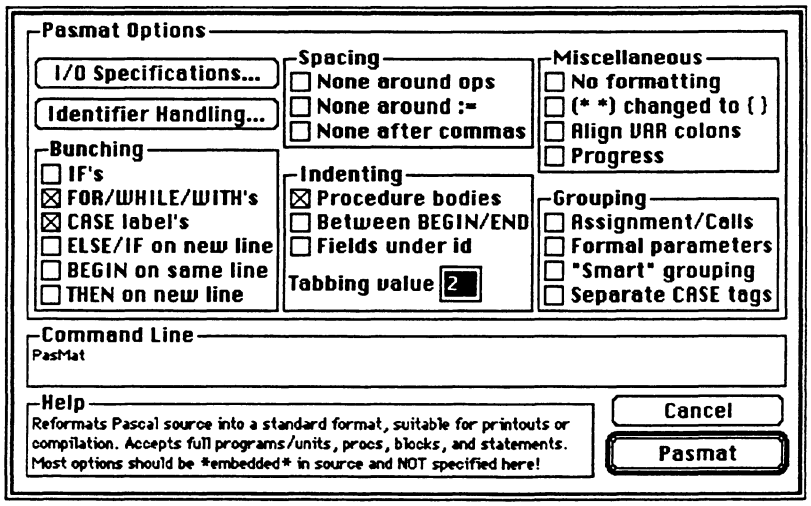
0 Normal termination.
1 Parameter or option error.

Options

Most of the following options modify the initial default settings of the directives described in Appendix K of the *MPW 3.0 Pascal Reference.*

-a Set **a-** to disable CASE label bunching.

-b Set **b+** to enable IF bunching.



-body Set **body+** to align procedure bodies with their enclosing BEGIN/END pair.

-c Set **c+** for placement of BEGIN on same line as previous word.

d Set **d+** to enable the replacement of (* *) with { } comment delimiters.

-e Set **e+** to capitalize identifiers.

-entab Replace runs of blanks with tabs. The tab value is determined by the **-t** option or current **t=*n*** directive (*not* by the file's tab setting).

-f Set **f-** to disable formatting.

-g Set **g+** to group assignment and call statements.

-h Set **h-** to disable FOR, WHILE, and WITH bunching.

-i *pathname* [, *pathname*]...
Search for `include` files in the specified directories. Multiple **-i** options may be specified. At most 15 directories will be searched. The search order for `include` files is specified under the description of the **-i** option for the Pascal command. (Note, however, that `USES` are not processed by PasMat.)

-in Set **in+** to process Pascal compiler `include` files. This option is implied if the **-i** option is used. (Be sure to read the "Limitations" at the end of this command section.)

-k Set **k+** to indent statements between BEGIN/END pairs.

-l Set **l+** for literal copy of reserved words and identifiers.

-list *listingFile*
Generate a listing of the formatted source. The listing is written to the specified file.

-n Set **n+** to group formal parameters.

-o *width* Set the output line width. The maximum value allowed is 150. The default is 80.

-p Display version information and progress information to diagnostic output.

-pattern = *pattern* = *replacement* =

Process `include` files (**-in**) and generate a set of output files with exactly the same `include` structure as the input, but with new names.The new output filenames and `include` directives are generated by editing the input (or `include`) filenames according to the *pattern* and *replacement* strings. *Pattern* is a pathname to be looked for in the input file and in each `include` file (the *entire* pathname is used, and case is ignored). If the pattern is found, it is replaced by the *replacement* string. The result is a new pathname, which becomes the name for an output file. For example,

```
PasMat -pattern =OldFile=NewFile=
```

replaces each name containing the string "`OldFile`" with the string "`NewFile`".

Note: Any character not contained in the *pattern* or *replacement* strings can be used in place of an equal sign. Special characters must be quoted. (See "Example" below.)

-q Set **q+** not to treat the ELSE IF sequence specially.

-r Set **r+** to make reserved words uppercase.

-rec Indent a RECORD's field list under the record identifier.

-s *renameFile*

Rename identifiers. *RenameFile* is a file containing a list of identifiers and their new names. Each line in this file contains two identifiers of up to 63 characters each: The first name is the identifier to be renamed; the second name will replace all occurrences of the first identifier in the output. There must be at least one space or tab between the two identifiers. Leading and trailing spaces and tabs are optional. The case of the first identifier doesn't matter, but the second identifier must be specified exactly as it is to appear in the output. The case of all identifiers not specified in the *renameFile* is subject to the other case options (**-e, -l, -u,** and **-w**) or their corresponding directives. Reserved words cannot be renamed.

-t *tab* Set the tab amount for each indentation level. If the **-entab** option is also specified, tab characters will actually be generated. The default tab value is 2.

-u Rename all identifiers based on their first occurrence in the source. Specifications in the rename (**-s**) file always have precedence over this option—that is, the identifier's translation is based on the rename file rather than on the first occurrence.

-v Set **v+** to put THEN on a separate line.

-w Set **w+** to make identifiers uppercase.

-x Set **x+** to suppress space around operators.

-y Set **y+** to suppress space around :=.

-z Set **z+** to suppress space after commas.

-: Set **:+** to align colons in VAR declarations (only if a **j** PasMat directive in the source specifies a *width*).

-@ Set **@+** to force multiple CASE tags onto separate lines.

"-#" Set # + for "smart" grouping of assignment and call statements. Grouped assignment and call statements on an input line will appear grouped on output.

- *Note:* Because **#** is the Shell's comment character, this option must be quoted on the command line.

-_ Set **_+** for "portability" mode (underscores are deleted from identifiers).

All options except for **-list, -pattern, -s,** and **-entab** have directive counterparts. It's recommended that you specify the options as directives in the input source so that you won't have to specify them each time you call PasMat.

{PasMatOpts} variable: You can also specify a set of default options in the exported Shell variable {PasMatOpts}—PasMat processes these options before it processes the command line options. {PasMatOpts} should contain a string (maximum length 255) specifying the options exactly as you would specify them on the command line. The exception is command-line quoting, which should be omitted. Also note that the options **-pattern, -list, -s,** and **-i,** which require a string parameter, can be specified only on the command line. For example, you can define {PasMatOpts} to the Shell (perhaps in the UserStartup file) as follows:

```
Set PasMatOpts "-n -u -r -d -entab -# -o 82 -t 2"
Export PasMatOpts
```

The entire definition string must be quoted to preserve the spaces.

As an alternative to specifying the options directly, you can indicate that the options are stored in a file by specifying the file's full pathname prefixed with the character ^ :

```
Set PasMatOpts "^ pathname "
Export PasMatOpts
```

PasMat will now look for the default options in the specified file. The lines in this file can contain any sequence of command-line options (*except* for **-pattern**, **-list**, **-s**, and **-i**), grouped together on the same or separate lines. You can comment the lines by placing the comment in braces ({...}). A typical options file might appear like this:

```
-n          {group formal params on same line}
-u          {auto translation of id's based on 1st
            occurrence}
-r          {uppercase reserved words}
-d          {leave comment braces alone}
-entab      {place real tabs in the output}
-#          {smart grouping}
-o 82       {output line width}
-t 2        {indent tab value}
```

(Except for the tab value, this example shows the recommended set of options.)

If PasMat does find a default set of options, those options will be echoed as part of the status information given with the **-p** option.

Example

```
Pasmat -n -u -r -d -pattern "==formatted/="  Sample.p  ∂
"formatted/Sample.p"
```

Formats the file Sample.p with the **-n, -u, -r,** and **-d** options and writes the output to the file "formatted/Sample.p". `Include` files are processed (**-pattern**), and each Pascal compiler `$I include` file causes additional output files to be generated. Each of these files is created with the name "formatted/ *filename* ", where *filename* is the filename specified in the corresponding `include`. (The **-pattern** parameter contains a null pattern (==) with "formatted/" as a replacement string—a null pattern always matches the start of a string.)

Care must be taken when a command line contains quotes, slashes, or other special characters that are processed by the Shell itself. In this example, we used the slash character, so the strings containing it had to be quoted.

Limitations

PasMat has these limitations:

- The maximum length of an input line is 255 characters.
- The maximum output line length is 150 characters.
- The input files and output files must be different.
- Only syntactically correct programs, units, blocks, procedures, and statements are formatted. This limitation must be taken into consideration when separate MPW 3.0 `include` files and conditional compiler directives are to be formatted.
- The Pascal `include` directive should be the last thing on the input line if `include` files are to be processed. `Include` files are processed to a maximum nesting depth of five. All `include` files not processed are summarized at the end of formatting. (This assumes, of course, that the **in** directive/option is in effect.)
- The identifiers CYCLE and LEAVE are treated as reserved Pascal keywords by PasMat. They are treated as two loop control statements by Pascal unless explicitly declared.
- While Pasmat supports Pascal's `$$Shell` facility in `include` files, the processing of MPW's {PInterfaces} files is *not* fully supported because these files conditionally `include` files (remember, conditionals are not processed). For this reason, do *not* use the **-in** or **-e** option to process files that include MPW {PInterfaces} files.

Availability

PasMat is available as part of a separate Apple product, MPW 3.0 Pascal .

See also

Pascal and PasRef commands.
Appendix K of the *MPW 3.0 Pascal Reference.*

PasRef—Pascal cross-referencer

Syntax

PasRef [*option...*] [*sourceFile...*]

Description

Reads Pascal source files and writes a listing of the source followed by a cross-reference listing of all identifiers. Each identifier is listed in alphabetical order, followed by the number of the line on which it appears. Line numbers can refer to the entire source file, or can be relative to individual `include` files and units. Each reference indicates whether the identifier is defined, assigned, or simply named (for example, used in an expression).

See the *MPW 3.0 Pascal Reference* for more information about the Pascal language. The first dialog box of PasRef's Commando dialog is reproduced here for your convenience.

Identifiers may be up to 63 characters long and are displayed in their entirety unless overridden with the **-x** directive. Identifiers can remain as they appear in the input, or they can be converted to all lowercase (**-l**) or all uppercase (**-u**).

For `include` files, line numbers are relative to the start of the `include` file; an additional key number indicates which `include` file is referred to. A list of each `include` file processed and its associated key number is displayed prior to the cross-reference listing.

`USES` declarations can also be processed by PasRef (their associated `$U` *filename* compiler directives are processed as in the Pascal compiler). These declarations are treated exactly like includes, and, as with the compiler, only the outermost `USES` declaration is processed (that is, a used unit's `USES` declaration is not processed).

As an alternative to processing `USES` declarations, PasRef accepts multiple source files. Thus you cross-reference a set of main programs together with the units they use. All the sources are treated like `include` files for display purposes. In addition, PasRef checks to see whether it has already processed a file (for example, if it appeared twice on the input list, or if one of the files already used or included it). The file is skipped it has already been processed.

Type

Tool.

Input

If no filenames are specified, standard input is processed. Unless the **-d** option is specified, multiple source files are cross-referenced as a whole, producing a single source listing and a single cross-reference listing. Specifying the **-d** option is the same as executing PasRef individually for each file.

Output All listings are written to standard output. Form feed characters are placed in the file before each new source listing and its associated cross-reference. Pascal `$P` (page eject) compiler directives are also processed by PasRef, which may generate additional form feeds in the standard output listing.

Diagnostics Parameter errors and progress information are written to diagnostic output.

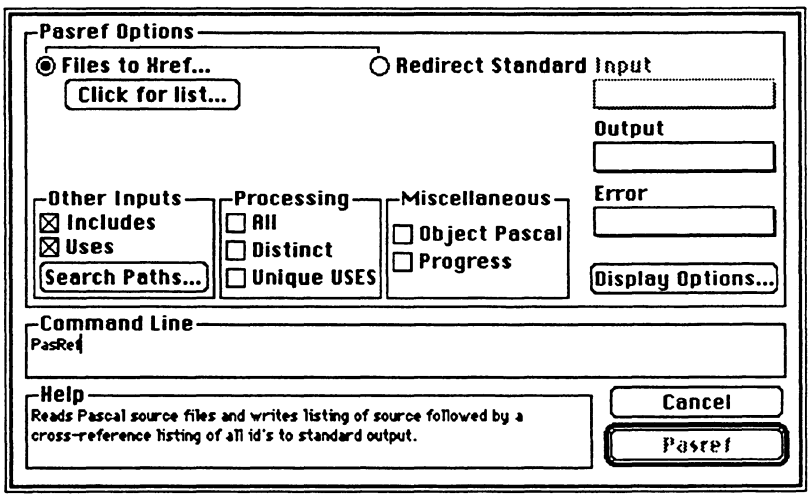
Status PasRef may return the following status codes:

0 Normal termination.
1 Parameter or option error.

Options

-a Process all files, even if they are duplicates of those already processed. The default is to process each (`include`) file or used unit only once.

-c Do *not* process a unit if the unit's filename is specified in the list of files to be processed on the command line, or if the unit has already been processed.



-d Treat each file specified on the command line as distinct. The default is to treat the entire list of files as a whole, producing a single source listing and a single cross-reference listing. This option is the same as executing PasRef individually for each specified file.

-i *pathname*[,*pathname*]...

Search for `include` or `USES` files in the specified directories. Multiple **-i** options may be specified. At most, 15 directories will be searched. The search order is specified under the description of the **-i** option for the Pascal command.

-l Display all identifiers in the cross-reference table in lowercase. This option should not be used if **-u** is specified, but if it is, the **-u** is ignored.

-ni | -noincludes

Do not process `include` files. (The default is to process the `include` files.)

-nl | -nolisting

Do not display the input source as it is being processed. (The default is to list the input.)

-nolex Do not display the lexical information on the source listing.

-nt | -nototal

Do not display the total line count in the source listing. This option is ignored if no listing is generated (**-nl**).

-n[u] | -nouses

Do not process `USES` declarations. (The default is to process `USES` declarations.) If **-nu** is specified, the **-c** option is ignored. (Be sure to read "Limitations" at the end of this PasRef section.)

-o The source file is an Object Pascal program. The identifier OBJECT is considered as a reserved word so that Object Pascal declarations can be processed. The default is to assume that the source is not an Object Pascal program.

-p Write version and progress information to diagnostic output.

-s Do not display `include` and `USES` information in the listing or cross reference, and cross-reference by total source line-number count rather than by `include`-file line number.

-t Cross-reference by total source line-number count rather than by `include`-file line number. This option can be used if you are not interested in knowing the positions in included files. However, the `include` information is still displayed (unless **-s, -ni,** or **-nu** is specified). This option is implied by the **-s** option.

-u Display all identifiers in the cross-reference table in uppercase. This option should not be used if **-l** is specified.

-w *width* Set the maximum output width of the cross-reference listing. This setting determines how many line numbers are displayed on one line of the cross-reference listing. It does not affect the source listing. *Width* can be a value from 40 to 255; the default is 110.

-x *width* Set the maximum display width for identifiers in the cross-reference listing. (The default is to set the width to the size of the largest identifier cross-referenced.) If an identifier is too long to fit in the specified width, it is indicated by preceding the last four characters with an ellipsis (...). *Width* can be a value from 8 to 63.

Normally, both `include` files and `USES` declarations are processed. The **-noincludes** option suppresses processing of `includes`. The **-nouses** option suppresses processing of `USES`.

Omitting the **-nouses** option causes PasRef to process a `USES` declaration exactly as does the Pascal compiler. However, you may want to cross-reference an entire system, including all of the units of that system. Processing the units through the `USES` declaration would cause only the Interface section of each unit to be processed. If you use the **-nouses** option, then `USES` will not be processed and each unit from the parameter list can be cross-referenced, treating the entire list as a single source.

PasRef can also cross-reference all the units of a program while still expanding other units not directly part of that program, such as the Toolbox units. If you wish to do this, use the **-c** option. With the **-c** option, if the (`$U` interface) filename is the same as one of the filenames specified on the parameter list, then the unit will not be processed from the `USES` declaration, because its full source will be (or has been) processed.

To summarize, you have the following choices:

- Don't process the `USES` declarations, and specify a list of all files you want to process by using the **-nouses** option.
- Process only the Interfaces through the `USES` declarations (like the compiler) by omitting the **-nouses** option.
- Process some of the units through the `USES` declarations and other units as full sources by specifying the **-c** option.

In all cases where a list of files is specified, no unit will ever be processed more than *once* (unless the **-a** option is given).

Example

```
PasRef -nu -w 80  Memory.p > Memory.p.Xref
```

Cross-references the sample desk accessory Memory.p and writes the output to the file Memory.p.Xref. No USES declarations are processed (**-nu**). The following source and cross-reference listings are generated:

```
1       1       1 --    {
2       1       2 --    File Memory.p
3       1       3 --
4       1       4 --    Copyright Apple Computer, Inc. 1985-1987
5       1       5 --    All rights reserved.
6       1       6 --    }
7       1       7 --
8       1       8 --      {$D+} { MacsBug symbols on }
9       1       9 --      {$R-} { No range checking  }
10      1       10 --
11      1       11 --   UNIT Memory;
12      1       12 --
13      1       13 --   INTERFACE
14      1       14 --
15      1       15 --   USES
16      1       16 --           MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf;
17      1       17 --
18      1       18 --
19      1       19 --   FUNCTION DRVROpen     (ctlPB: ParmBlkPtr; dCtl: DCtlPtr): OSErr;
20      1       20 --   FUNCTION DRVRControl  (ctlPB: ParmBlkPtr; dCtl: DCtlPtr): OSErr;
21      1       21 --   FUNCTION DRVRStatus   (ctlPB: ParmBlkPtr; dCtl:.DCtlPtr): OSErr;
22      1       22 --   FUNCTION DRVRPrime    (ctlPB: ParmBlkPtr; dCtl: DCtlPtr): OSErr;
23      1       23 --   FUNCTION DRVRClose    (ctlPB: ParmBlkPtr; dCtl: DCtlPtr): OSErr;
24      1       24 --
25      1       25 --
26      1       26 --   IMPLEMENTATION
etc.
63      1       63 --A FUNCTION DRVRClose    (ctlPB: ParmBlkPtr; dCtl: DCtlPtr): OSErr;
64      1       64 0-A BEGIN
65      1       65 --       IF dCtl^.dCtlwindow <> NIL THEN
66      1       66 1-          BEGIN
67      1       67 --             DisposeWindow (WindowPtr(dCtl^.dCtlWindow));
68      1       68 --             dCtl^.dCtlWindow := NIL;
69      1       69 -1          END;
70      1       70 --          DRVRClose := NOErr;
71      1       71 -0A    END;
etc.
178     1       178 --
179     1       179 --  END. {of memory UNIT}
180     1       180 --
```

Each line of the source listing is preceded by five columns of information:

1. The total line count.
2. The include key assigned by PasRef for an `include` or USES file. (See below.)
3. The line number within the `include` or main file.
4. Two indicators (left and right) that reflect the static block nesting level. The left indicator is incremented (mod 10) and displayed whenever a BEGIN, REPEAT, or CASE is encountered. On termination of these structures with an END or UNTIL, the right indicator is displayed, then decremented. It is thus easy to match BEGIN, REPEAT, and CASE statements with their matching terminations.
5. A letter that reflects the static level of procedures. The character is updated for each procedure nest level ("A" for level 1, "B" for level 2, and so on), and displayed on the line containing the heading, and on the BEGIN and END associated with the procedure body. Using this column you can easily find the procedure body for a procedure heading when there are nested procedures declared between the heading and its body.

The cross-reference listing follows:

```
1. Memory.p

-A-

accEvent              144            ( 1)
accRun                158            ( 1)
ApplicZone            121            ( 1)
Away                   33*           ( 1)  146 ( 1)

-B-

BeginUpdate           151            ( 1)
BNOT              39           ( 1)
Bold                  ( 1)  117      ( 1)
Boolean           31*          ( 1)
BOR               39           ( 1)
BSL                   39             ( 1)

-C-

csCode                143            ( 1)
CSParam               146            ( 1)
ctlPB             19*          ( 1)  20*( 1)   21*( 1)  22*( 1)..23*( 1)  43*( 1)
                     63*           ( 1)  74*( 1)  143 ( 1) 146 ( 1) 168*( 1) 173*( 1)
```

```
-D-

dCtl              19*              ( 1)  20*( 1)  21*( 1)  22*( 1)  23*( 1)  37*( 1)
                  39         ( 1)  43*( 1)  50 ( 1)  53 ( 1)  54 ( 1)  55 ( 1)
                  63*              ( 1)  65 ( 1)  67 ( 1)  68 ( 1)  74*( 1) 115 ( 1)
                 142         ( 1) 168*( 1) 173*( 1)
DCtlPtr           19         ( 1)  20 ( 1)  21 ( 1)  22 ( 1)  23 ( 1)  37 ( 1)
                  43         ( 1)  63 ( 1)  74 ( 1) 168 ( 1) 173 ( 1)
dCtlRefNum        39         ( 1)  54 ( 1)
dCtlWindow        50         ( 1)  55=( 1)  67 ( 1)  68=( 1) 142 ( 1)
etc.

-V-

VolName                 79*              ( 1) 100 ( 1) 135 ( 1)

-W-

  what           149         ( 1)
  WindowKind      54=                     ( 1)
  windowpeek      54         ( 1)
  WindowPtr       48         ( 1)  67 ( 1) 151 ( 1) 153 ( 1)
  wRect           47*              ( 1)

*** End PasRef: 105 id's   249 references
```

The numbers in parentheses following the line numbers are the `include` keys of the associated `include` files (shown in column 2 of the source listing). The `include` filenames are shown following the source listing. You can thus see what line number was in which `include` file. An asterisk (*) following a line number indicates a definition of the variable. An equal sign (=) indicates an assignment. A line number with nothing following it indicates a reference to the identifier.

Limitations

PasRef has these limitations:

- PasRef does not process conditional compilation directives! Thus, given the "right" combination of `$IFC`s and `$ELSEC`s, PasRef's lexical (nesting) information can be thrown off. If this happens, or if you don't want the lexical information, you can specify the **-nolex** option.
- PasRef stores all its information on the Pascal heap. Up to 5000 identifiers can be handled, but more identifiers will mean less cross-reference space. A message appears if PasRef runs out of heap space.

◆ *Note:* Although PasRef never misses a reference, it can infrequently be fooled into thinking that a variable is defined when it actually isn't. One case where this happens is in record structure variants. The record variant's case tag is always flagged as a definition (even when there is no tag type) and the variant's case label constants (if they are identifiers) are also sometimes incorrectly flagged, depending on the context. (This occurs only in the declaration parts of the program.)

- While PasRef supports Pascal's `$$Shell` facility in `include` files and `USES` declarations, the processing of MPW's {PInterfaces} files is *not* fully supported because these files conditionally include files (remember, conditionals are not processed). For this reason, always use the **-nu** option to suppress processing of `USES` declarations.
- The identifiers `CYCLE` and `LEAVE` are treated as reserved Pascal keywords by PasRef. These are treated as two loop control statements by Pascal unless explicitly declared.

Availability

PasRef is available as part of a separate Apple product, MPW 3.0 Pascal.

See also

Pascal command.

MPW 3.0 Pascal Reference.

Paste—replace selection with Clipboard contents

Syntax Paste [-c *count*] *selection* [*window*]

Description Finds *selection* in the specified window and replaces its contents with the contents of the Clipboard. If no window is specified, the command operates on the target window (the second window from the front). It's an error to specify a window that doesn't exist.

For a definition of *selection,* see "Selections" in Chapter 6; a summary of the selection syntax is contained in Appendix B.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

0 At least one instance of the selection was found.
1 Syntax error.
2 Any other error.

Option **-c** *count* For a count of *n,* replace the next *n* instances of the selection with the contents of the Clipboard.

Examples

```
Paste §
```

Replaces the current selection with the contents of the Clipboard. This command is like the Paste item in the Edit menu, except that the action occurs in the target window.

```
Paste /BEGIN/:/END/
```

Selects everything from the next `BEGIN` to the following `END` and replaces the selection with the contents of the Clipboard.

See also Copy, Cut, and Replace commands.
"Edit Menu" in Chapter 3.
"Selections" in Chapter 6.

PerformReport—generate a performance report

Syntax PerformReport [*option...*]

Description PerformReport reads a link map file and a performance data file and produces a report that relates the performance data to procedure names. The input files are both text files and are distinguished as separate options. For a full discussion of MPW's performance measurement tools, see Chapter 14.

Type Tool.

Input Standard input is not processed.

Output The report file is written to standard output.

Diagnostics If no errors are detected, PerformReport runs silently. Errors and warnings are written to the diagnostic output file. Progress and summary information is also written to the diagnostic output if the **-p** option is specified.

Status The following status codes may be returned:

0 No errors.
1 Warning issued.
2 Error encountered.
3 Heap error;usually insufficient memory.

Options

-a Produce a listing of all procedures (in segment order). (The default is to produce only a partial listing sorted by the number of possible hits.)

-l *fileName* Read the link map of the file named *fileName.*

-m *fileName*
Read the performance data file named *fileName.* The default name is Perform.out.

-n *NN* Show the top *NN* procedures. The default is 50.

-p Write progress and summary information to the diagnostic output file.

Example

```
Catenate "{MPW}ROM.Maps:MacIIROM.map >> myMapFileName
PerformReport -l myMapFileName > myReport
```

Adds the ROM map file to the end of the link map file, *myMapFileName*. Reads the files *myMapFileName* and Perform.out and writes the output to *myReport*.

See also

Chapter 14, "Performance-Measurement Tools."

MPW 3.0 Pascal Reference.

MPW 3.0 C Reference .

Position—list position of selection in window

Syntax Position [-c | -l] [*window...*]

Description Position displays the position of the selection in each of the windows specified. If no window is specified, the position of the selection in the Target window is given. By default, the position is displayed as both the line number of the start of the selection and the character positions of the start and end of the selection. The **-c** option can be used to display only the character positions of the selection. Similarly, the **-l** option can be used to display only the line number.

Type Built-in.

Input None.

Output The position information is written to standard output.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

0 No errors.
1 Syntax error.
2 Any other error.

Options

-l Display just the line number of the start of the selection.

c Display just the character positions of the start and end of the selection.

Examples

```
Position "{Target}" file2
```

Displays the position of the selection in both the Target and file2 in the following form:

```
578         23129,23140
211         8440,8440
```

See also Find command.

Print—print text files

Syntax Print [*option...*] *file...*

Description Prints text files on the currently selected printer. (Printers are selected with the Chooser desk accessory.) One or more files may be printed.

- *Note:* Print does not substitute fonts unless the "Font Substitution" box is checked in the "LaserWriter Page Setup" dialog. To print in a font other than that indicated in the resource fork of the file where the MPW editor stores font information, use the **-font** option.

△ **Important** Print requires the printer drivers available on version 1.0 (or later) of the *Printer Installation* disk. △

Type Tool.

Input Prints the file names on the command line.

Output All output goes to the currently selected printer. Print sends no output to standard output.

Diagnostics Errors and warnings are written to diagnostic output. If the **-p** option is specified, progress and summary information is also written to diagnostic output.

Status The following status codes may be returned:

0 Successful completion.
1 Parameter or option error.
2 Execution error.

Options *Note:* You can also apply the Print options to the Print Window/Print Selection menu item by including them in the exported Shell variable {PrintOptions}. {PrintOptions} is originally set to **-h** in the Startup file.

-b Print a round-rect border around the printable area of the page. Headers, if specified with the **-h** option, are separated from the body text by an extra line.

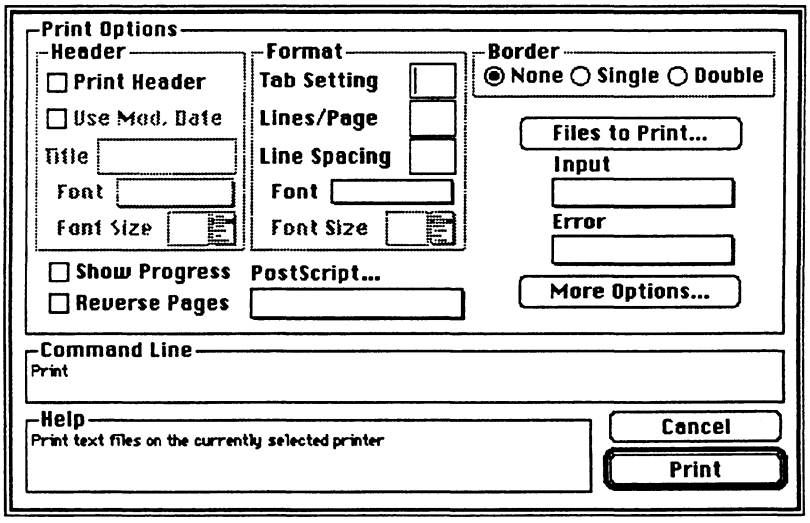
-b2 Print an alternate form of the above border in which the header appears above and outside the border.

-c[opies] *n* Print *n* copies of the file or selection.

-f[ont] *name*

Print using the font identified by *name* (for example, Courier). The default is the font indicated in information in the resource fork of the file, if present, and otherwise Monaco 9. (See also the **-size** option.)

- *Note:* Printing with a font that is not directly supported by the printer is significantly slower than printing with a built-in font.



-ff *string* Specifies a string that will be treated like a form-feed character if it is encountered at the beginning of a line. If the string is the only item on the line, that line will be omitted. If the string is followed by additional characters on the line, the additional characters will be printed on the first line of the new page.

-from *n* Print pages starting from page number *n*. The default is to start with the first page of the file.

-h Print page headers at the top of each page. The header indicates the time of printing, the name of the file, and the page number.

-hf[ont] *name*

Specify the font to be used in headers (**-h** option). The default is the font used in the file.

-hs[ize] *n* Specify the font size to be used in headers. The default is 10.

-l[ines] *n* Print (at most) *n* lines per page. Line spacing is adjusted so that the full page is used. If both **-l** and **-ls** are specified, the **-l** option takes precedence.

-ls *n* Set line spacing. A value of 1 indicates normal (single) spacing (the default), 2 indicates double-spacing, and so on. Fractional values are permitted.

-md Print the file's last modified date, rather than the date and time of printing, in the header (if headers are specified).

-n Turn on line numbering; numbers appear to the left of the printed text.

-nw *n* Specify the width of the line number (**-n**) field in characters. (The default value is 5.) Negative values for *n* cause the field to be zero-padded. The valid range of values is -10 through 10.

-p Write progress information to diagnostic output, indicating which files are printing and the number of lines and pages printed.

-ps *file* Send PostScript commands in the file to the LaserWriter prior to printing each page.

-page *n* Number the pages of the file, beginning with *n*. (By default, page numbers start with 1.)

-q *quality* Set print quality on the ImageWriter®. The value of *quality* can be any of the following strings:

```
high    standard    draft
```

- *Note:* This option is ignored when printing on the LaserWriter.

-r Output pages to the printer in reverse order. This option eliminates the need to reorder pages on the LaserWriter and LaserWriter Plus (although not for the LaserWriter II printers).

-s[ize] *n* Print using the font size identified by *n*. The default is to use the font size indicated in the resource fork of the file, if present; otherwise, the default size is 9.

-t[abs] *n* Expand tabs, using the indicated tab setting. If this option isn't specified, the tab setting is taken from the resource fork of the file, if present; otherwise, the tab setting is taken from the {Tab} variable.

-title *name* If printing page headers (with **-h**), use *name* as the title. (The default is to use the filename.)

-to *n* Print pages up to page *n*. (The default is to print to the last page of the file.)

The following options control the page margins. *n* is the margin width in inches.

-tm *n* Top margin (default = 0 inches).

-bm *n* Bottom margin (default = 0 inches).

-lm *n* Left margin (default = 0.2778 inch, for three-hole punched pages).

-rm *n* Right margin (default = 0 inches).

Examples

```
Print -h -size 8 -ls 0.85 Startup UserStartup
```

Prints the files Startup and UserStartup with page headers, using Monaco 8 and compressing the line spacing.

```
Print -b -hf helvetica -hs 12 -r print.p
```

Prints the "print.p" source file with borders, with headers in Helvetica 12, and with pages in reverse order.

See also Print menu item in "File Menu," Chapter 3.

ProcNames—display Pascal procedure and function names

Syntax ProcNames [*option* ...] [*file* ...]

Description ProcNames is a Pascal utility that accepts a Pascal program or unit as input and produces a listing of all its procedure and function names. The names are shown indented as a function of their nesting level. The nesting level and line-number information is also displayed.

ProcNames can be used in conjunction with the Pascal "pretty-printer" PasMat when that utility is used to format separate `include` files. For that case, PasMat requires that the initial indenting level be specified. This level is exactly the information provided by ProcNames.

The line-number information displayed by ProcNames exactly matches that produced by the Pascal cross-reference utility PasRef (with or without `USES` declarations being processed), so ProcNames can be used in conjunction with the listing produced by PasRef to show just the line numbers of every procedure or function header.

Another possible use for the ProcNames output is to use the line-number and file information to find procedures and functions quickly with Shell editing commands.

Type Tool.

Input The file parameters specify a list of Pascal source file names to be processed. Standard input is processed if no filenames are specified. Unless the **-d** option is specified, the entire list of files is treated as a single group of files to be processed as a whole, producing a single procedure/function summary. Specifying the **-d** option is equivalent to executing ProcNames individually for each specified file.

Output The procedure/function name listing is written to the standard output file. Form feed characters are placed in the file before each new list (unless the **-e** option is specified).

Diagnostics Errors are written to diagnostic output.

Status ProcNames may return the following status codes:

0 Normal termination.
1 Parameter or option error.

Options

-c Do not process a used unit if the unit's `$U` interface filename is specified in the list of files to be processed. This option has the same effect on the line numbering as does the **-c** option in the PasRef utility.

-d Reset total line-number count to 1 on each new file. If a list of files is specified, the total line-number count may either start at 1 or continue from where it left off in the previous file. The default is to agree with the listing produced by PasRef when it processes a list of files—that is, to continue the count. However, if you want ProcNames to treat each file independently, you can specify the **-d** option so that the total line-number count is reset to 1 before each file is processed.

-e Suppress page eject (form feed) between each procedure/function listing.

-f PasMat format compatibility mode. The default lists the procedure and function names as a function of their Pascal Compiler indenting level. However, for indenting purposes only, a special case is made of level 1 procedures in the Implementation section of a unit. PasMat formats these procedures indented under the word Implementation. Thus they are indented as if they were level 2 procedures. If you intend to use the level information for PasMat, you should specify the **-f** option.

-i *pathname*[,*pathname*]...
Search for `include` or `USES` files in the specified directories. Multiple **-i** options may be specified. At most 15 directories will be searched. The search order is specified under the description of the **-i** option for the Pascal command.

-n Suppress all line-number and level information in the output display. Only the procedure and function names will be shown appropriately indented.

-o The source file is an Object Pascal program. The identifier OBJECT is considered as a reserved word so that Object Pascal declarations may be processed. The default assumes that the source is not an Object Pascal program.

-p Display version information and progress information in the diagnostic file.

-u Process USES declarations. The only reason you would need to process USES declarations with ProcNames would be to make the line-number information agree with a PasRef listing that also contains processed USES declarations. The default does not process the USES declarations because they have no effect on the procedure name listing, only on the associated line numbers. Thus, if you specify the **-n** option to suppress the line-number information, it makes no sense to process USES declarations; thus, the **-u** option will be ignored when the **-n** option is specified. (See the notes in "Limitations.")

Examples

```
procnames Memory.p >names
```

Lists all the procedures and functions for the Pascal program Memory.p and writes the output to the file "names". The listing below is the output generated in the "names" file.

```
Procedure/Function names for Memory.p
    11    11    0     Memory[Main] Memory.p
    37    37    1     RsrcID
    43    43    1     DRVROpen
    63    63    1     DRVRCloseaa
    74    74    1     DRVRControla
    76    76    2           DrawWindow
    83    83    3                 PrintNum
    93    93    3                 GetVolStuff
    108   108   3                 PrtRsrcStr
    168   168   1     DRVRPrime
    173   173   1     DRVRStatus
*** End ProcNames: 11 Procedures and Functions
```

The first two columns on each line are line-number information. The third column is the level number. The first column shows the line number of a routine within the total source. The second column shows the line number within an `include` file (`include` files are always processed). As each `include` file changes, the name of the file from which input is being processed is shown along with the routine name on the first line after the change in source. Segment names (from Pascal compiler `$S` directives) are similarly processed. These are shown enclosed in square brackets (the blank segment name is shown as "[Main]").

Limitations

Only syntactically correct programs are accepted by ProcNames. Conditional compilation compiler directives are not processed.

Although ProcNames supports `$$Shell` facility in `includes` and `USES`, the processing of MPW's {PInterfaces} files is not fully supported because these files conditionally include files. Therefore, do not use the **-u** option.

Project—set or write the current project

Syntax Project [-q | *projectname*]

Description Set the current project to *projectname* or list the current project if *projectname* is omitted. *Projectname* must be a mounted project. Refer to the MountProject command for information on how to mount projects.

See Chapter 7 for complete definitions of the terms and symbols used in Projector commands.

Type Built-in.

Input None.

Output If no project name is given, the current project name is written to standard output.

Diagnostics Errors and warnings are written to diagnostic output.

Status The following status codes may be returned:

0 No errors.
1 Syntax error.
2 Error in processing.

Option **-q** Do not quote the project name.

Examples The command

```
Project
```

causes the current project name to be written to standard output. To change the current project to OurProject, use

```
Project OurProject
```

See Also NewProject, MountProject.

ProjectInfo—list project information

Syntax

ProjectInfo [-project *project*] [-comments] [-latest] [-f] [-r] [-s] [-only | -m] [-af *author*] [-a *author*] [-df *dates*] [-d *dates*] [-cf *pattern*] [-c *pattern*] [-t *pattern*] [-n *name*] [-log] [-update | -newer] [*object*...]

Description

By default (with no options specified), ProjectInfo lists information about each revision in every revision tree (file) in the current project. This behavior can be changed using the various options. For example, using the **-latest** option will display only information about the latest revision on the main trunk of each revision tree. Using the **-f** option will display information about the revision tree, rather than the particular revisions within that tree. Various other options exist that filter the output such that only the information (typically revisions) that passes through the filter is listed.

If *object* is a project pathname such as Enterprise∫Phaser∫file.c or Enterprise∫Phaser, Projector lists information about every revision of file.c in the Phaser project, or information about every revision tree in the project Enterprise∫Phaser, respectively.

If *object* is a leafname such as file.c, Projector looks in the current project for a revision tree with that name. If found, information about every revision in that revision tree (file.c) is listed. If the file is not a member of the current project, Projector looks for the file in the current directory. If the file exists and is part of a project, then the current state of that file is listed. Projector can determine whether a file belongs to a project because that information is maintained in the resource fork of all checked-out files.

Finally, if *object* is a valid partial or full HFS pathname of a file, and the file is part of a project, then the current state of that file is listed.

To list the contents of a specific revision of a file, append a comma followed by the revision number to the filename specified. For example, revision 22 of file.c is specified as file.c,22.

You can use the **-af**, **-a**, **-df**, **-d**, **-n**, **-cf**, **-c**, and **-t** options to filter (constrain) the information listed to specific authors, dates, names, specific comments, or tasks.

Use the **-log** option to display a log of all changes to the project. These commands are logged: NameRevisions, DeleteRevisions, and DeleteNames.

See Chapter 7 for complete definitions of the terms and symbols used in Projector commands.

Type Built-in.

Input None.

Output Information is written to standard output.

The following template shows how ProjectInfo displays project information:

```
Project     Name
            filename,revision
            Author: author of current revision
            Status: Date
            Task:
            Comment:
```

The first line lists the project name to which the file or revision belongs. The project name is listed only at the beginning of the file or revision list corresponding to that project. The filename is something like file.c. By default, every revision of every revision tree is listed. If you use the **-latest** option, only the latest revision on the main trunk is listed. A plus sign (+) by the revision name indicates that the revision is currently checked out. An asterisk (*) by the revision name indicates that it is a modified read-only file. The status is either "Checked in" or "Checked out." The date is the date and time corresponding to the check-in or check-out of that revision. The task is the task associated with that file or revision. The comment is an optional field included with the **-comments** option.

Diagnostics Errors and warnings are written to the diagnostic output file.

Status The following status codes may be returned:

0 No errors.
1 Syntax error.
2 Error in processing.
3 System error.

Options

-a *author* List only revisions created by the author.

-af *author* List only files created by the author.

-c *pattern* List only those revisions whose comments contain *pattern.*

-cf *pattern* List only those files whose comments contain *pattern.* A pattern can be a literal string or a regular expression enclosed in slashes (/).

-comments Include comments associated with each project, file, and revision listed. They are normally omitted.

-d *dates* List only those revisions whose creation date is within *dates.*

-df *dates* List only those files whose creation date is within *dates.* A date is specifed as mm/dd/yy [[hh:mm [:ss] AM | PM].

Dates can take the following forms:

Format	Meaning
date	On *date.*
<*date*	Before but not including *date.*
≤*date*	Before and including *date.*
>*date*	After but not including *date.*
≥*date*	After and including *date.*
date1-date2	Between and including *date1* and *date2.*

- *Note:* Be sure to quote *dates* so that the MPW Shell does not interpret any of the special characters.

-f List file information.

-log Print the log information for the current or named project. The log contains information about the creation and deletion of public names, and the deletion of revisions.

-m List only modifiable files or revisions.

-n *name* List only those revisions in *name.*

Names can take the following forms:

Format	Revisions
name	With Name *name.*
<*name*	Before but not including *name.*
≤*name*	Before and including *name.*
>*name*	After but not including *name.*
≥*name*	After and including *name.*

- *Note:* If any of the name relations are used (<, ≤, >, ≥), quote *name* so that the MPW Shell does not interpret the special characters.

-newer Get information on the latest revision of all files in the current project that have more recent revisions than the file currently in the checkout directory for the project. Information is given for files that do not exist in the checkout directory. This option is similar to the **-newer** option to the CheckOut command, except that information is listed instead of checking out the file.

-only List only information about projects and subprojects in the current or named project—that is, do not list information about files.

-project *project*
Name of the project that contains the files. This becomes the current project for this command.

-r Recursively list all subprojects encountered—that is, list every file in every subproject.

-latest List only the latest revision on the main trunk.

-s Short listing (names and revision names only).

-t *pattern* List only those revisions whose task contains *pattern.*

-update Similar to the **-newer** option except that information is not given for files that do not exist in the checkout directory. This option parallels the **-update** option to the CheckOut command, except that information is listed instead of checking the file out.

Examples

In the example below, the current project has three files. The **-latest** option is used so that only information about the latest revision on the main trunk is listed. The presence of the plus sign (+) indicates that Bob currently has revision 22 of file.c checked out for modification, and that Peter has revision 33 of hdr.c checked out for modification. The date field of these two files reflects the date and time they were checked out. Because no plus sign appears on the line for file.h, it can be checked out for modification. The latest revision of file.h is 17, and the author of the revision is Bob.

```
ProjectInfo -latest

Sample∫
file.c,22+
    Owner: Bob
    Checked out: Fri, Apr 8, 1988, 3:45 PM
    Task:  Fixing bug #223

file.h,17
    Author: Bob
    Checked in: Mon, Apr 4, 1988, 10:10 AM
    Task:

hdr.c,33+
    Owner: Peter
    Checked out: Tue, Apr 12, 1988, 5:58 PM
    Task:  Fixing bug #333
```

Using the **-only** option causes ProjectInfo to list only information about the project itself.

```
ProjectInfo -only

Sample∫
    Author: Bob
    Create date:  Mon, Apr 4, 1988 8:20 AM
    Mod date: Thu, Apr 14, 1988, 6:00 PM
```

Use the **-f** option to list filenames. Note that revision numbers are absent and that the file's author and last-mod-date are listed. In the example below, file.c and hdr.c are currently checked out.

```
ProjectInfo -f
Sample∫

    file.c
          Author: Bob
          Create date: Mon, Apr 4, 1988, 10:00 AM
          Mod date: Tue, Apr 5, 1988, 2:15 PM
          Free: No
    file.h
          Author: Bob
          Create date: Mon, Apr 4, 1988, 10:00 AM
          Mod date: Mon, Apr 4, 1988, 10:00 AM
          Free: Yes
    hdr.c
          Author: Peter
          Create date: Mon, Apr 4, 1988, 3:30 PM
          Mod date: Mon, Apr 4, 1988, 6:00 PM
          Free: No
```

Use the **-f** and **-s** options together to output the list of files in the project:

```
ProjectInfo -f -s
Sample∫

    file.c
    file.h
    hdr.c
```

The following command will display the entire revision history of file.c. Note that the comment option has been included here as well.

```
ProjectInfo -comments file.c
file.c,2+

    Owner: Bob
    Checked out: Fri, Apr 8, 1988, 3:45 PM
    Task: Fixing bug #223
    Comment: COMMENT...

file.c,2
    Author: Bob
    Checked in: Thu, Apr 7, 1988, 1:10 PM
    Task: Fixing bug #222
    Comment: COMMENT...

file.c,1
    Author: Bob
    Checked in: Mon, Apr 4, 1988, 9:25 PM
    Task: Updating procedure comments
    Comment: COMMENT...
```

Information about HFS files may be displayed by specifying a partial or full HFS pathname. This displays the information in the `'ckid'` resource of the file.

```
ProjectInfo :file.c

:file.c,22*
    Owner: Bob
    Project: Sample∫
    Checked out: Fri, Apr 8, 1988, 3:45 PM
    Task: Fixing bug #223
```

The asterisk (*) following the name indicates that the file is a modified read-only file.

In the example below, only revisions created by Bob and created on or after April 4, 1988, are displayed.

```
ProjectInfo -a Bob -d "≥4/4/88"

Sample∫
    file.c,22+
          Owner: Bob
          Checked out: Fri, Apr 8, 1988, 3:45 PM
          Task: Fixing bug #223
    file.c,22
          Author: Bob
          Checked in: Thu, Apr 7, 1988, 1:10 PM
          Task: Fixing bug #222

    file.c,21
          Author: Bob
          Checked in: Mon, Apr 4, 1988, 9:25 PM
          Task: Updating procedure comments

    file.h,17
          Author: Bob
          Checked in: Mon, Apr 4, 1988, 10:10 AM
          Task:
```

In the example below, only revisions that have a task dealing with Bug #222 are listed.

```
ProjectInfo -t /bug≈222/

Sample∫
    file.c,22
          Author: Bob
          Checked in: Thu, Apr 7, 1988, 1:10 PM
          Task: Fixing bug #222

    hdr.c,31
          Author: Peter
          Checked in: Fri, Apr 1, 1988, 3:50 PM

          Task: Bug222 - Adding check procedure
```

The final example demonstrates the **-log** option.

```
ProjectInfo -log

TheShell∫Projector
  7/5/88 4:07 PM
    Peter J. Potrebic
    DeleteNames Work
  7/2/88 1:37 PM
    Peter J. Potrebic
    NameRevisions Work bitmaps.a,2 ckid.c,3a2
```

The log shows that Peter created a public name on July 2 and then deleted it on July 5.

See Also MountProject and UnmountProject.

Quit—quit MPW

Syntax Quit [-y | -n | -c]

Description This command is equivalent to the menu command Quit. Quit executes the standard quit procedures, asking confirmation to save modified files, close all windows, and so on.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

1 Syntax error.
2 Command aborted.

- *Note:* Quit cannot return a status of 0, because if there are no errors the command never returns.

Options

-y Answer "Yes" to any confirmation dialog that occurs, causing all modified windows to be saved before closing them.

-n Answer "No" to any confirmation dialog that occurs, causing all modified windows to be closed without saving any changes.

-c Answer "Cancel" to any confirmation dialog that occurs. This effectively aborts the command if any windows need to be saved.

Examples

```
Quit -y
```

Quits MPW answering "Yes" to any dialogs such as those prompting to save files.

```
Quit -c
```

Quits MPW, unless any confirmation dialogs occur and dialog boxes are displayed.

See also Shutdown command.

Quote—quote parameters

Syntax Quote [-n] [parameters...]

Description Quote writes its parameters, separated by spaces and terminated by a return, to standard output. Parameters containing characters that have special meaning to the Shell's command interpreter are quoted with single quotation marks. If no parameters are specified, only a return is written.

Quote is identical to Echo except that Quote quotes parameters that contain special characters. Quote is especially useful when using Shell commands to write a script.

The following special characters are quoted:

Space Tab Return Null

; & | ( ) ∂ ' " / \ { } ` ? ≈ [ ] + * • ® < > ≥ ...

Type Built-in.

Input None.

Output Parameters are written to standard output and are enclosed in single quotation marks if they contain special characters.

Diagnostics None.

Status Status code 0 (no problem) is always returned.

Option

-n Don't write a return following the last parameter. The insertion point remains at the end of the output. The **-n** isn't written to standard output.

Examples

```
Echo ≈.a
Quote ≈.a

Sample.a Count.a My Program.a
Sample.a Count.a 'My Program.a'
```

Echo and Quote behave slightly differently for parameters that contain special characters. The first line above was produced by Echo; the second by Quote.

```
Quote  Notice what happens to single quotes: "--'--"
Notice what happens to single quotes: '--'∂''--'
```

Because single quotes can't appear within single quotes, they are replaced with '∂'' which closes the original single quote, adds a literal quote, and reopens the single quotes.

```
For file In ≈.a
    Quote Print "{file}"
End

Print Sample.a
Print Count.a
Print 'My Program.a'
```

The For loop shown above writes a Print command for each file that matches the pattern ≈.a. These commands can then be selected and executed. Notice the quotation marks in the last Print command.

See also Echo and Parameters commands.

Rename—rename files and directories

Syntax Rename [-y | -n | -c] *name newName*

Description The file, folder or disk specified by *name* is renamed *newName*. A dialog box requests a confirmation if the rename would overwrite an existing file or folder. The **-y**, **-n**, or **-c** options can be used to avoid this interaction.

- *Note:* You can't use the Rename command to change the directory a file is in. To do this, use the Move command.

- *Note also:* Wildcard renames in the following form *will not work:*

```
Rename ≈.text ≈.p
```

  This is because the Shell expands the filename patterns "≈.text" and "≈.p" before invoking the Rename command. In order to gain the desired effect, you would need to execute a command such as the one shown in the fifth example below.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

0 Successful rename.
1 Syntax error.
2 *Name* does not exist.
3 An error occurred.
4 Cancel was selected or implied by the **-c** option.

Options

-y	Answer "Yes" to any confirmation dialog that may occur, causing the conflicting file or folder to be deleted.
-n	Answer "No" to any confirmation dialog that may occur, stopping the rename if *newName* already exists.
-c	Answer "Cancel" to any confirmation dialogs, aborting the rename if *newName* already exists.

Examples

```
Rename File1 File2
```

Changes the name of File1 to File2.

```
Rename HD:Programs:Prog.c  Prog.Backup.c
```

Changes the name of Prog.c in the directory HD:Programs to Prog.Backup.c in the same directory.

```
Rename Untitled: Backup:
```

Changes the name of the disk Untitled to Backup.

```
Rename -c File1 File2
```

Changes the name of File1 to that of File2; if a conflict occurs, it cancels the operation and returns a status of 4.

To perform a wildcard rename, you could execute the following set of commands:

```
For Name In ≈.text
    ( Evaluate "{Name}" =~ /(≈)®1.text/ ) > Dev:Null
    Rename "{Name}" "{®1}.p"
End
```

The Evaluate command is executed only for its side effect of permitting regular expression processing. (The expression operator =~ indicates that the right side of the expression is a regular expression.) Thus, you can use the regular expression capture mechanism, (*regularExpr*)®*n*. Evaluate's output is tossed in the bit bucket (Dev:Null).

See also

Move command.

Replace—replace the selection

Syntax Replace [-c *count*] *selection replacement* [*window*]

Description Replace finds *selection* in the specified window and replaces it with *replacement*. If no window is specified, the command operates on the target window (the second window from the front). It's an error to specify a window that doesn't exist. If a count is specified, the Replace command is repeated *count* times.

For a definition of *selection*, see "Selections" in Chapter 6. A summary of the selection syntax is contained in Appendix B.

You can include references to parts of the selection in the *replacement* by using the ® operator. The expression ®*n*, where *n* is a digit, is replaced with the string of characters that matches the regular expression tagged by ®*n* in the selection. (See "Tagging Regular Expressions With the ® Operator" in Chapter 6.)

The *selection* is a selection expression while *replacement* is a string (that could contain the ® operator). If *replacement* contains spaces or special characters, enclose it in quotation marks.

All searches are by default not case sensitive. To specify case-sensitive matching, set the {CaseSensitive} variable before executing the command.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

0 At least one instance of the selection was found.
1 Syntax error.
2 Any other error.

Option

-c *count* Repeat the command *count* times. As a convenience, ∞ (Option-5) can be specified in place of a number. **-c** ∞ replaces all instances of the selection from the current selection to the end of the document (or to the start of the document, for a backward search).

Examples

```
Replace -c ∞  /myVar/ 'myVariable'  Prog.p
```

Replaces every subsequent instance of the selection with the string in single quotation marks.

```
Replace -c 5  /•[ ∂t]+/  ''
```

Strips off all the spaces and tabs at the front of the next five lines in the file (and replaces them with the null string). This action takes place in the target window.

```
Set HexNum "[0-9A-F]+"
Set Spaces "[ ∂t]+"
Replace -c ∞  /({HexNum})®1{Spaces}({HexNum})®2/  ®1∂n®2
```

Defines two variables for use in the subsequent Replace command, and converts a file that contains two columns of hex digits (such as the icon list from ResEdit) into a single column of hex digits.

See also

Find and Clear commands.

Chapter 6.

Appendix B.

Request—request text from a dialog box

Syntax Request [-q] [-d *default*] [*message...*]

Description Request displays an editable text dialog box with OK and Cancel buttons and the prompt *message*. If you select the OK button, any text you type into the dialog box is written to standard output. The **-d** option lets you set a default response to the request.

Type Built-in.

Input Reads standard input for the message if no parameters are specified.

Output Text from the dialog box is written to standard output.

Diagnostics None.

Status Request may return the following status codes:

0 The OK button was selected.
1 Syntax errors.
4 The Cancel button was selected.

Options

-d *default* The editable text field of the dialog box is initialized to *default*. The default text appears in the dialog box; if the OK button is selected without changing the response, the default is written to standard output.

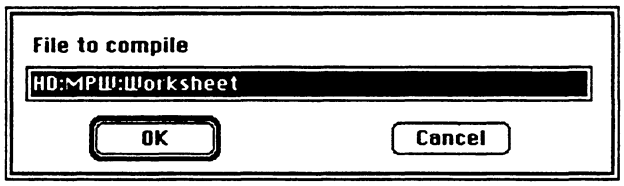
-q Makes Request quiet—that is, Request always returns a status of either zero or one. This is useful in scripts.

Examples

```
Set Exit 0
Set FileName "`Request 'File to compile' -d "{Active}"`"
If {FileName} ≠ ""
Pascal "{FileName}" ≥≥ "{WorkSheet}"
End

Set Exit 1
```

Displays a dialog box that lets the user enter the name of a file to be compiled. Sets the default to be the name of the active window, as follows:



See also Alert and Confirm commands.

ResEqual—compare resources in files

Syntax ResEqual [*option*] *file1 file2*

Description ResEqual compares the resources in two files and writes their differences to standard output.

ResEqual checks that each file contains resources of the same type and identifier as the other file; that the size of the resources with the same type and identifier are the same; and that their contents are the same.

Type Tool.

Input The *file1* and *file2* parameters specify the two files whose resources are to be compared.

Output Descriptions of the differences in the resources of the two files are written to standard output.

The following messages appear when reporting differences:

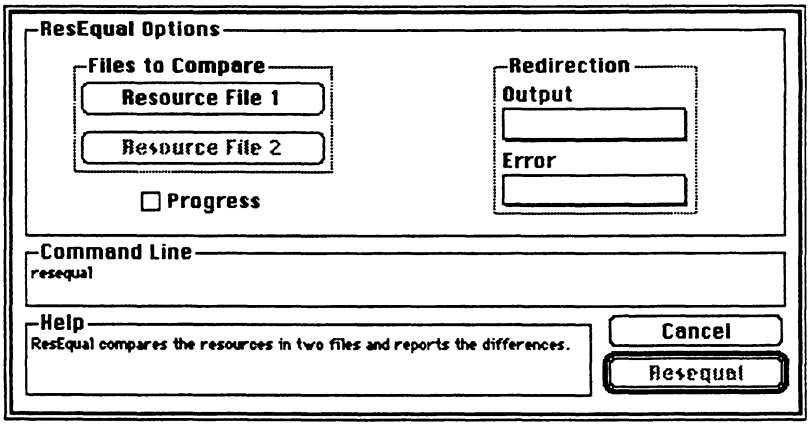
- In 1 but not in 2
  —the resource type and ID are displayed—
- In 2 but not in 1
  —the resource type and ID are displayed—
- Resources are different sizes
  —the resource type and ID are displayed—
  —the size of the resource in each file is displayed—
- Resources have different contents
  —the resource type and ID are displayed—
  Contents of resource in file 1 at offset
  —offset to the differing bytes from the start of the resource is displayed—
  —16 bytes at the offset are displayed—
  Contents of resource in file 2 at offset
  —offset to the differing bytes from the start of the resource is displayed—
  —16 bytes at the offset are displayed—

Diagnostics Parameter errors are written to diagnostic output.

Status The following status codes may be returned:

0 Resources match.
1 Parameter or option error.
2 Files don't match.

Option **-p** Write progress information to diagnostic output.



Example

```
Resequal Sample Sample.rsrc
```

Compares the resources in Sample and Sample.rsrc, writing the results to standard output.

Limitations When the contents of resources are compared and a mismatch is found, the mismatch and the subsequent 15 bytes are written. ResEqual then continues the comparison, starting with the byte following the last displayed.

If more than ten differences are detected in the same resource, the rest of the resource is skipped and processing continues with the next resource.

See also Equal command. (The **-r** option of Equal compares the resource forks of files on a byte-by-byte basis, including the resource map.)

Revert—revert to saved files

Syntax Revert [-y] [*window...*]

Description Reverts the specified windows to their previously saved states. If no window is specified, Revert works on the target window. Revert displays a confirmation dialog box, but you can avoid this dialog box by using the **-y** option to revert unconditionally to the last saved version of the document.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status The following status codes may be returned:

0 No errors.
1 Parameter or option error.
2 The specified window does not exist.
3 A system error occurred.
4 The Cancel button was selected.

Option **-y** Unconditionally revert all named windows to their previously saved states.

Examples

```
Revert
```

Displays a confirmation dialog box for reverting the target window to its last saved state.

```
Revert -y "{Worksheet}"
```

Reverts unconditionally to last saved worksheet.

See also Close and Save commands.

Rez—resource compiler

Syntax Rez [*option...*] [*resourceDescriptionFile...*]

Description Rez compiles the resource fork of a file according to a textual description. The resource description file is a text file that has the same format as the output produced by DeRez, the resource decompiler. The data used to build the resource file can come directly from the resource description file(s) as well as from other text files (via `#include` and `read` directives in the resource description file) and from other resource files (via the `include` directive).

Rez includes macro processing, full expression evaluation, and built-in functions and system variables. For information about Rez and the format of a resource description file, see Chapter 11. For a summary of the format of a resource description file, see Appendix D.

Type Tool.

Input Standard input is processed if no filenames are specified.

For all input files on the command line, the following search rules are applied:

1. Try to open the file with the name specified "as is."
2. If the preceding rule fails, and the filename contains no colons or begins with a colon, append the filename to each of the pathnames specified by the {RIncludes} variable and try to open the file.

Output No output is sent to the standard output file. By default, the resource fork is written to the file Rez.out. You can specify an output file with the **-o** option.

Diagnostics If no errors or warnings are detected, Rez runs silently. Errors and warnings are written to diagnostic output. If an error is detected, Rez sets the output file's modification date to zero.

Status Rez may return the following status codes:

0 No errors.
1 Error in parameters.
2 Syntax error in file.
3 I/O or program error.

Options

-align *word* [*longword*]

Align resources along word or longword boundaries. This may allow the Resource Manager to load these resources faster. The **-align** option is ignored when the **-a** option is in effect.

-a[ppend] Append Rez's output to the output file rather than replacing the output file.

▲ **Warning** Rez overwrites any existing resource of the same type and ID without any warning message. Rez cannot append resources to a resource file that has its Read Only bit set. Also, Rez cannot replace a resource that has its protected bit set unless the **-ov** option is specified. Although it is possible to append a resource directly to a running system file, this is not recommended. See also the **-ov** option that follows.▲

-c[reator] *creatorExpr*

Set the output file creator. (The default value is `'????'`.) Note that *creatorExpr* is a Rez expression, such as

```
-c "3*200+5"
```

If the creator begins with a letter and contains no fancy characters, you can simply enter it. For example,

```
-c APPL
```

Otherwise, you can enter the creator as a numeric expression or as a literal expression, such as

```
-c " '§§§' "
```

-d[efine] *macro*[=*data*]

Define the macro variable *macro* to have the value *data*. If *data* is omitted, *macro* is set to the null string—note that this still means that *macro* is defined. Using the **-d** option is the same as writing

`#define` *macro* [*data*]

at the beginning of the input.

-i *pathname(s)*

Search the following pathnames for `#include` files. This option can be specified more than once. The paths are searched in the order they appear on the command line.

```
rez -i {mpw}myStuff:    -i hd:tools...
```

-m[odification]

Don't change the output file's modification date. If an error occurs, the output file's modification date is set to zero, even if you use this option.

-o *outputFile*

Place the output in *outputFile*. The default output file is Rez.out.

-ov Override the protected bit when replacing resources with the **-a** option.

-p[rogress] Write version and progress information to diagnostic output.

-rd Suppress warning messages if a resource type is redeclared.

-ro Set the mapReadOnly flag in the resource map.

-s *pathname(s)*

Search the following pathnames for resource `include` files.

-t[ype] *typeExpr*

Set the type of the output file. The default value is 'APPL'. Note that *typeExpr* is a Rez expression, such as

```
-t "3*200+5"
```

If the type begins with a letter and contains no fancy characters, you can simply enter it. For example,

```
-t MPST
```

Otherwise, you can enter the type as a numeric expression or literal expression, such as

```
-t " 'ŞŞŞ' "
```

-u[ndef] *macro*

Undefine the macro variable *macro*. This is the same as writing

`#undef` *macro*

at the beginning of the input. It is meaningful to undefine only the preset macro variables.

Example

```
Rez Types.r Sample.r  -o Sample
```

Generates a resource fork for the file Sample, based on the descriptions in Types.r and Sample.r.

See also

DeRez and RezDet commands.

Chapter 11 and Appendix D.

Standard resource type declarations in the directory {RIncludes}:

- Types.r
- SysTypes.r
- MPWTypes.r
- Pict.r

RezDet—detect inconsistencies in resources

Syntax RezDet [-b] [-q | -s | -d | -r | -l] *resourceFile...*

Description If no options are specified, RezDet investigates the resource fork of each file for damage or inconsistencies. The specified files are read and checked one by one. Output is generated according to the options specified.

RezDet checks for the following conditions:

- The resource fork is at least the minimum size. (There must be enough bytes to read a resource header.)
- There is no overlap or space between the header, the resource data list, and the resource map. There should be no bytes between the EOF and the end of the resource map.
- Each record in the resource data list is used once and only once. The last data item ends exactly where the data list ends.
- Each item in the resource type list contains at least one reference; each sequence of referenced items starts where the previous resource type item's reference list ended; and each item in the reference list is pointed to by one and only one resource type list item.
- There are no duplicates in the resource type list.
- Each name in the name list has one and only one reference, and the last name doesn't point outside the name list.
- There are no duplicate names in the name list. Duplicate names cause an advisory warning rather than a true error. This warning is given only if the **-s**, **-d**, or **-r** option is selected.
- Each reference list item points to a valid data item and either has a name list offset of –1 or points to a valid name list offset.
- Bits 7 (Unused), 1 (Changed), or 0 (Unused) should not be set in the resource attributes.
- All names have a nonzero length.

Fields are displayed as hexadecimal or decimal for numeric values, or as a hex dump with associated printable Macintosh characters. The characters newline ($0D), tab ($09) and null ($00) are displayed as "¬", "Δ", and ".", respectively.

◆ *Note:* RezDet does not use the Resource Manager and should not crash, no matter how corrupt the resource fork of the file.

Type Tool.

Input RezDet does not read from standard input.

Output Information describing the resource fork is written to standard output (together with any other information generated by the **-s**, **-d**, **-l**, or **-r** options).

Diagnostics Error messages go to diagnostic output.

Status The following status codes may be returned:

0 No errors detected.
1 Invalid options or no files specified.
2 Resource format error detected.
3 Fatal error—an I/O or program error was detected.

Options Only one of the following options can be used at one time:

-q[uiet] Don't write any information to standard output. This option suppresses all resource file format errors normally generated.

-s[how] Write information about each resource to standard output.

-d[ump] Same as **-show** but also generates detailed information about headers, name lists, data lists, and so forth.

-r[awdump] Same as **-dump** but also dumps contents of data blocks, and so forth.

- *Note:* This option can generate *huge* amounts of output.

-l[ist] List resource types, IDs, names, attributes, and resource sizes to standard output in the following format:

'type' (ID,name,attributes) [size]

The following option can be used by itself or with other options:

-b[ig] Read the data for each resource into memory one resource at a time, instead of all at once (used for huge resource files). If RezDet tells you that it ran out of memory, try using this option.

Examples

```
RezDet "{SystemFolder}System"
```

Checks the System file for damage.

```
RezDet -q Foo || Delete Foo
```

Removes the file Foo if the resource fork is damaged.

Limitations

Duplicate resource name warnings are generated even if the names belong to resources of different types.

The file attributes field in the resource map header is not validated.

The Finder-specific fields in the header and resource map header are ignored.

RotateWindows—rotate between windows

Syntax RotateWindows

Description RotateWindows places the front MPW window in the back and brings the second window to the front. Multiple calls to RotateWindows rotate through all open MPW windows. RotateWindows brings only MPW windows to the front (desk accessory windows are not rotated). You might want to add this command to a menu, along with a command key equivalent. For example:

```
AddMenu 'Extras' 'RotateWindows/®' 'RotateWindows'
```

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status RotateWindows may return the following status codes:

0 No errors.
1 Syntax error (error in parameters).

Options None.

Example

```
RotateWindows
```

Puts the front MPW window in back, and brings the target MPW window to the front.

See also StackWindows, SizeWindow, MoveWindow, and ZoomWindow commands.

Save—save windows

Syntax Save [-a | *window...*]

Description Saves the contents of *window* or a list of *windows* to disk without closing them. The **-a** option saves all open windows. Save without any parameters saves the target window (the second window from the front).

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status Save may return the following status codes:

0 No errors.
1 Syntax error.
2 Specified window does not exist.

Option

-a Save all open windows. This option cannot be used when any windows are specified.

Examples

```
Save -a
```

Saves all open windows.

```
Save "{Active}" "{Worksheet}"
```

Saves the Worksheet window and the contents of the active window.

See also Close and Revert commands.

Search—search files for a pattern

Syntax Search [-s | -i] [-r] [-q] [-f *file*] /*pattern*/ [*file*...]

Description Searches the input files for lines that contain a pattern and writes those lines to standard output. If no file is given, standard input is searched. When reading from files, the filenames and line numbers of matching lines are prepended to each line of output.

Pattern (defined in "Pattern Matching" in Chapter 6 and in Appendix B) is a regular expression, optionally enclosed in forward slashes (/).

Type Tool.

Input Standard input is read if no files are specified.

Output Each matching line is written to standard output.

Diagnostics Error messages are written to diagnostic output.

Status The following status codes may be returned:

0 No error.
1 Syntax error.
2 Pattern not found.

Options

-r Write the lines **not** matching the pattern to standard output.

-q Write only the matching lines to standard output. Do not prepend filename and line number.

-s Case-sensitive search, overriding {CaseSensitive} variable.

-i Case-insensitive search, overriding {CaseSensitive} variable.

-f *file* All lines that do not get written to standard output are written into this file.

Examples

```
Search /procedure/ Sample.p
```

Searches the file Sample.p for the pattern "procedure". All lines containing this pattern are written to standard output.

```
Search /Export/ "{MPW}"StartUp "{MPW}"UserStartUp
```

Lists the Export commands in the StartUp and UserStartup files.

```
Search /PROCEDURE [a-zA-Z0-9_]*;/ "{PInterfaces}"≈
```

Searches for the procedures with no parameters in the Pascal interface files supplied with MPW Pascal. Because more than one input file is specified, a filename will precede each line in the output.

```
Search -f file.nonmatch /pattern/ file
```

All lines of "file" that contain "pattern" are written to standard output. All other lines will be placed in file.nonmatch. This, in effect, splits the file in two pieces, using "pattern" as the key.

```
Search -r -f file.nonmatch /pattern/ file
```

This does the opposite of the preceding example. All lines that do not contain "pattern" are echoed to standard output, and all other lines (that is, those containing "pattern") are written to file.nonmatch.

See also

Find command.

"Pattern Matching (Using Regular Expressions)" in Chapter 6.

Set—define or write Shell variable

Syntax Set [*name* [*value*]]

Description Set assigns the string *value* to the variable *name*. If *value* is omitted, Set writes the name and its current value to standard output. If both *name* and *value* are omitted, Set writes a list of all variables and their values to standard output. (This output is in the form of Set commands.)

- *Note:* To make variable definitions available to enclosed scripts and programs, you must use the Export command.

Type Built-in.

Input None.

Output If *value* or both *name* and *value* are omitted, variable names and their values are written to standard output.

Diagnostics Error messages are written to diagnostic output.

Status These status codes may be returned:

0 No error.
1 Syntax error.
2 Variable "name" does not exist.

Options None.

Examples

```
Set CIncludes "{MPW}CFiles:CIncludes:"
```

Redefines the variable CIncludes.

```
Set CIncludes
```

Displays the new definition of CIncludes.

```
Set Commands ∂
    ":,{MPW}Tools:,{MPW}Applications:,{MPW}ShellScripts:"
```

Redefines the variable {Commands} to include the directory "{MPW}ShellScripts:". (See Chapter 5 for a complete list of predefined variables.)

```
Set > SavedVariables
    # ... other commands
Execute SavedVariables
```

Writes the values of all variables to file SavedVariables. Because the output of Set is actually Set commands, the file can be executed later to restore the saved variable definitions. This technique is used in the Suspend and Resume scripts to save and restore variable definitions, as well as exports, aliases, and menus.

See also

Export, Unexport, and Unset commands.

"Defining and Redefining Variables" in Chapter 5.

"The Startup and UserStartup Files" in Chapter 5.

SetDirectory—set the default directory

Syntax SetDirectory *directory*

Description SetDirectory sets the default directory and adds the new default directory to the Directory menu if it is not already present. The *directory* parameter must be specified.

- *Note:* Directory names should not contain any of the special characters shown below. These characters all have special meaning when they appear in menu items:

  ```
  - ; ^ ! < / (
  ```

The SetDirectory script is used to implement the Set Directory menu item in the Directory menu.

Type Script.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status These status codes may be returned:

0 Successful completion.
1 Parameter error or unable to set directory.

Options None.

Examples

```
SetDirectory "{MPW}"Scripts:
```

Sets the default directory to the Scripts folder in the {MPW} directory and adds "{MPW}"Scripts: to the Directory menu if it's not already there.

```
SetDirectory...
```

Uses the Commando dialog box to select the default directory interactively.

See also Directory, DirectoryMenu, and Files commands.

SetFile—set file attributes

Syntax SetFile [*option...*] *file...*

Description Sets attributes for one or more files. The options apply to all files listed.

Type Built-in.

Input None.

Output None.

Diagnostics Error messages are written to diagnostic output.

Status These status codes may be returned:

0 The attributes for all files were set.
1 Syntax error.
2 An error occurred.

Options

-c *creator* Set the file creator. *Creator* must be four characters; for example,

```
-c 'MPS '
```

-t *type* Set the file type. *Type* must be four characters; for example,

```
-t 'TEXT'
```

-d *date* Set the creation date. *Date* is a string in the form

"*mm/dd/yy* [*hh:mm*[:ss] [AM | PM]]"

representing month, day, year (0–99), hour (0–23), minute, and second. The string must be quoted if it contains a space. A period (.) indicates the current date and time.

-m *date* Set the modification date: same format as above. A period (.) indicates the current date and time.

-l *h,v* Set the icon location. *h* and *v* are positive integer values and represent the horizontal and vertical pixel offsets from the upper left corner of the enclosing window.

-a *attributes* Set the file attributes. The string *attributes* is composed of the characters listed below. Attributes that aren't listed remain unchanged.

L	Locked
V	Invisible
B	Bundle
S	System
I	Inited
D	On Desktop
M	Shared (can run multiple times)
A	Always switch launch (if possible)

Uppercase letters set the attribute to 1; lowercase letters set it to 0. For example,

```
Setfile -a vB Filename
```

clears the invisible bit and sets the bundle bit.

- *Note:* These attributes are described in the chapter "File Manager" of *Inside Macintosh.*

Examples

```
SetFile -c "MPS " -t MPST ResEqual
```

Sets the creator and type for the MPW Pascal tool ResEqual.

```
SetFile Foo -m "2/15/86 2:25"
```

Sets the modification date of file Foo.

```
SetFile Foo Bar -m .
```

Sets the modification date to the current date and time (the period is a parameter to **-m**, indicating current date and time). Setting the date is useful, for instance, before running Make.

See also Files command. (The **-l** and **-x** options display file information.)

SetPrivilege—set access privileges for folders on file server

Syntax SetPrivilege [-f *priv*] [-d *priv*] [-c *priv*] [-o *owner*] [-g *group*] [-r] [-i] *folder...*

Description Using SetPrivilege is equivalent to using the access privileges desk accessory. *Priv* is a character string (one, two or three characters long) that specifies privileges for the owner, the group, and everyone (**o**, **g**, and **e**, respectively). An uppercase letter enables the privilege; a lowercase letter disables the privilege. If a specific character is not in the string, the respective privilege is not changed.

Type Tool.

Input None.

Output When the **-i** option is used, folder information is written to standard output.

Diagnostics Errors are written to diagnostic output.

Status These status codes may be returned:

0 No error.
1 Syntax error.
2 Folder not found, or folder not an AppleShare folder.
3 User is not owner; could not modify privileges.

Options

-o *new owner*
Change owner of the folder to *new owner.*

-g *new group*
Change group of the folder to *new group.*

-r Recursively apply changes to enclosed folders.

-f *priv* See files. Set the privileges with respect to seeing files within folders (equivalent to read access).

-d *priv* See folders. Set the privileges with respect to seeing folders and directories and listing their contents.

-m *priv* Modify. Set privileges allowing users to make changes to files and directories.

-i Write folder information (owner, group, and access privileges) to standard output. The output is in the form of a SetPrivilege command. The **-r** option is the only option that may be used in conjunction with the **-i** option.

Examples

```
SetPrivilege -r -f OGe -d OGe -m Oge ∂
"Server:personal:peter"
```

This gives everyone in your group the ability to see files within Server:personal:peter without being able to change them. Anyone outside the group cannot see the files or folders or make changes. The owner can do everything.

Here is the easiest way to use the SetPrivilege command: Use the **-i** option to get information on folders and edit the privileges as desired. Then execute the resulting command. For example, to change the privileges for Server:Private, follow these steps:

1. Execute this command to obtain the current privileges:

```
SetPrivilege -i Server:Private
SetPrivilege Server:Private -o Joe -g Team -d OGE -f OGE -m OGE
```

- *Note:* These privileges show that Joe, the group Team, and everyone else has all privileges to the folder Private.

2. Now edit the output, adjusting the privileges as desired. For example,

```
SetPrivilege Server:Private -o Joe -g Team -d Oge -f  Oge -m O
```

- *Note:* Now only Joe, the owner, can see directories and files. Only Joe can make changes; all other users have no privileges.

3. Execute the resulting command.

SetVersion—maintain version and revision number

Syntax SetVersion [*option*...] *file*

Description SetVersion generates and maintains (sets or increments) the individual components making up a version number for a file. There are two forms of version numbering supported by SetVersion:

ver.rev The first version numbering form is "*ver.rev*", where *ver* is a version number and *rev* a revision number. The component values are kept in a private resource generated and maintained by SetVersion itself. The resource is generally used only by applications (for example, in their About box) and MPW tools (for example, when an MPW tool's **-p** option is used) that contain code to read the resource. It is also recognized by Commando to be displayed just below the Do It button of a Commando dialog box[1].

In this form of version numbering, the resource is maintained as a Pascal string with the resource type `'MPST'` and a resource ID of 0 (you can use the **-t** and **-i** options to specify another resource type and ID number if desired). The resource has the following layout (described as Rez input):

```
type 'MPST' as 'STR ';
resource 'MPST' (0) {
  "Version ver.rev"           /* a Pascal string */
};
```

The resource is created by SetVersion if it is not already there. The string always contains the characters "Version *ver.rev*", where *ver* and *rev* are digits. The version can optionally be prefixed with an arbitrary string (**-prefix**), and the revision can be similarly suffixed with an arbitrary string (**-suffix**) for more complex version numbering (such as "Version x1.23B2").

ver.rev.bugfix.relletr.nonrel The second version numbering form is "*ver.rev.bugfix.relletr.nonrel*", where *bugfix* is a bug fix level, *relletr* is a release letter (*d* for a development release, *a* for alpha, *b* for beta, omitted for final), and *nonrel* is an additional nonrelease development level. For a final release, *nonrel* is suppressed. For a *bugfix* level of 0, it is suppressed along with its leading period. The following table shows a few examples:

1 Commando only uses the SetVersion string resource if a "VersionDialog" is specified as part of the Commando resources. If omitted, Commando will look for a 'vers' resource(s).

Event	Version	Stage
first versions of the product	1.0d1, 1.0d2...	development
product features are defined—being tested	1.0a1, 1.0a2 ...	alpha
product is stable—begin final testing	1.0b1, 1.0b2 ...	beta
first released version	1.0	release
first revision	1.1d1 ...	
first bug fix to first revision	1.1.1d1 ...	
first major revision	2.0d1 ...	

As with the first form, the component values are kept in a resource generated and maintained by SetVersion. Thus the values may be displayed by applications and MPW tools that read the resource. The resource is also recognized by Commando.[2] However, the resource generated for the second form is also recognized by the Finder (version 6.1 and beyond) and used to generate the version information for the Finder's Get Info display. Therefore, this form of version resource may be added to *arbitrary* files (such as text or data files) to allow version number displays from their Get Info windows.

The resource for this form of version numbering always has the resource type `'vers'` and a resource ID of 1 or 2 (why there are two values will be explained shortly). The resource format is predefined in SysTypes.r for Rez which contains the following template:

```
#include "SysTypes.r"  /* for country codes and 'vers'  template */
type 'vers'
{
     hex byte;                    /* Major revision in BCD        */
     hex byte;                    /* Minor/bug-fix revision in BCD*/
     hex byte development = 0x20, /* Release stage                */
              alpha       = 0x40,
              beta        = 0x60,
              final       = 0x80, /* or */ release = 0x80;
     hex byte;                    /* Non-final release #          */
     integer Country;             /* Country code                 */
     pstring;                     /* Short version number string  */
     pstring;                     /* Long version number string   */
};
```

2 When Commando uses a `'vers'` resource, it first will look for a `'vers'` ,1 resource, and if not present, a `'vers'` ,2 resource. The short version string is displayed below the Do It button. Clicking this version number causes the long version string to be displayed in the "help" box. The two `'vers'` resources as well as the strings they contain are described when the `'vers'` resource format is described.

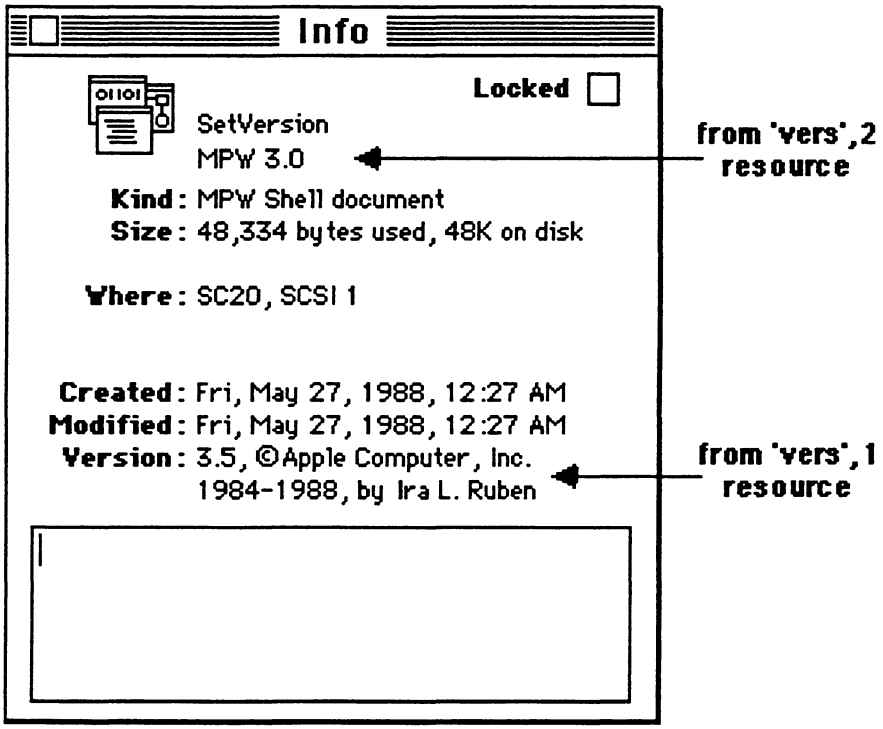
In the preceding resource description, the short version number message string contains just the version number, such as 1.0. The long version message can also include a copyright notice, release data, or other information, but should not include the program name. The following example illustrates the proper use of the Rez template:

```
resource 'vers' (1) {
  0x01, 0x23, beta, 0x45, verUS,
  "1.2.3b45",
  "1.2.3b45, ©Apple Computer, Inc. 1988"
};
```

These conventions have been imposed on the long version message because of the way the Finder uses this resource. Actually, each file can contain one, two, or no `'vers'` resources. A `'vers',1` resource, if present, identifies the version of the file itself. A `'vers',2` resource, if present, identifies the version and name of a set of files with which the file is shipped, linking files that make up the set. The Finder displays the long message from `'vers',1` and `'vers',2`, if they are present, in the Get Info window for a file. The Finder ignores the rest of the `'vers'` resource. Here is an example of the `'vers'` resources for the beta release of SetVersion itself, and the Finder's Get Info window for the SetVersion file:

```
resource 'vers' (1) {
  0x03, 0x50, beta, 0x00, verUS,
  "3.5",
  "3.5, ©Apple Computer, Inc. 1984-1988, by Ira L. Ruben"
};

resource 'vers' (2) {
  0x03, 0x00, beta, 0x01, verUS,
  "3.0b1",
  "MPW 3.0b1"   /* SetVersion "belongs" to MPW */
};
```



The other fields of a `'vers'` resource (besides the long message) are often useful to applications other than the Finder. Use the short version number to display the version of a particular file, as the Finder does for the System and Finder in the "About the Macintosh™ Finder" window. (SetVersion also uses the short message string to determine the version number components). The BCD version number is well suited for checking for a desired version number, or for comparing two versions. Note that this BCD numbering scheme represents a newer version with a greater number than the older version, so a numeric comparison between 4-byte values is all that is needed to find out which is newer.[3]

3 The comparison of the BCD field is only valid if the version number components don't exceed the limitations imposed by the resource. Specifically, the version and nonrelease values are limited to two BCD digits, while the revision and bug fix values are limited to one digit. Because of these limitations, SetVersion does not use the BCD value. SetVersion does, however, place the low-order digits of the actual version components (maintained in the short message) into the BCD fields. The BCD field is thus valid until the version counts exceed the corresponding BCD limitations.

SetVersion can increment (**-v**, **-r-**, **-b**, **-x**) or set (**-sv**, **-sr**, **-sb**, **-sx**) the various components of the version number. In the case of the `'vers'` resources it can set the country code (**-country**) and long version message string (**-verstring**) as well. The `'MPST'` resource (first SetVersion form) or the `'vers'` resource(s) (second Finder form) attached to the file is considered *the* location of the version number. If you attach the resource to the actual file, it will "go" wherever the file goes! Thus a filename is a required parameter to SetVersion. However, the values contained in the `'MPST'` resource or the short message of the `'vers'` resource can be used to set a corresponding string constant in a source file used to generate an application or tool (**-csource**, **-[p]source**, **-rezsource**). This feature is optional, but it should be used for two reasons: first, it explicitly allows the source to reflect the version numbers in the resource; second, if for any reason the resource cannot be accessed, the constant can be used. This is illustrated in the examples in the following paragraphs.

The following code fragments illustrate how each version resource and its corresponding source string constant can be used to access the version of an MPW tool:

To access the `'MPST'` resource...

```
- - -
CONST
  Version = '1.2';     {ver.rev string const.}
- - -
PROCEDURE GetVersion(VAR VerStr: Str255);
  VAR
    H: StringHandle;
    i: Integer;
  BEGIN {GetVersion - return version string in VerStr}
    H := StringHandle(Get1Resource('MPST', 0));{Get 'MPST' resource }
    IF H = NIL THEN   {Use string const.    }
      VerStr :=  Version    {if not found         }
    ELSE
     BEGIN
      i := Pos('Version', H^^) + 8;{Start of ver.rev     }
      VerStr := Copy(H^^, i, Length(H^^)-i+1);{Extract from resource}
     END;
  END; {GetVersion}
```

To access the 'vers' resource...

```
- - -
USES Files.p;   {Defines 'vers' layout}
- - -
CONST
  Version = '1.2.3b4';      {v.r.bsx string const.}
- - -
PROCEDURE GetVersion(Id: Integer; Short: Boolean; VAR VerRev: Str255);


  VAR
    H: VersRecHndl;
    p: StringPtr;


  BEGIN {GetVersion - return long or short version string in VerStr}
    VerRev := Version;      {assume failure!!      }
    IF (Id = 1) | (Id = 2) THEN   {if valid id specified }
      BEGIN
      H := VersRecHndl(Get1Resource('vers',Id));{Get 'vers',id resource}
      IF H <> NIL THEN      {if we got resource... }
        BEGIN
          p := StringPtr(@H^^.shortVersion[0]);{...point at short msg }
          IF NOT Short THEN {if we want long msg...}
            p := StringPtr(Ord(p)+Length(p^)+1);{...point at long msg  }
          VerStr := Copy(p^, 1, Length(p^));{Copy short or long msg}
        END;
      END;
  END; {GetVersion}
```

Normally, SetVersion is used as part of a makefile to automatically increment a specific version number component (or components) each time an application or tool is rebuilt, or in the 'vers' case, possibly when some kind of associated file is generated. Note that when SetVersion modifies a file or updates a source file, the modification date is *not* changed. Therefore, makefiles will not be affected by the use of SetVersion.

Type Tool.

Input The *file* parameter specifies the filename of a file containing the 'MPST' string resource or 'vers' resource(s).

Output None.

Diagnostics Errors are written to the diagnostic file.

Status SetVersion may return these status codes:

0 Normal termination.
1 Parameter or option error.

Options[4] **-b** Increment the bug fix level component by one. This option is allowed only when **-t** `'vers'` is specified to manipulate a Finder `'vers'` resource.

-country *name*

A Finder `'vers'` resource contains the International Utility's country code. The default is to use the code for the current country. The country option specifies the appropriate country. The following countries' names are allowed:[5]

Arabia	FrSwiss	Malta
Australia	Germany	Netherlands
BelgiumLux	Greece	Norway
Britain	GrSwiss	Portugal
China	Iceland	Spain
Cyprus	Ireland	Sweden
Denmark	Israel	Taiwan
Finland	Italy	Thailand
France	Japan	Turkey
FrCanada	Korea	US
		Yugoslavia

4 See the **-t** option for a summary of which options are valid as a function of which resource (SetVersion's string or Finder's `'vers'` resource) is being manipulated.

5 The country names are spelled exactly as specified in *Inside Macintosh* for the International Utilities.

-csource *file*

Update the string constant in the C source specified by the *file*. The constant is set to be the same as that specified by SetVersion's string resource or the short string from a `'vers'` resource. It is assumed that the constant is defined as a string constant in a `#define`, somewhere in the first 12,800 characters (twenty-five 512-byte blocks) of the file, as follows:

```
#defineΔVersion "ver.rev..."ΔΔΔΔΔΔΔΔΔΔ/*some comment*
```

The Δs indicate required spaces. There may be any number of spaces before the *required* comment. However, because SetVersion edits the line in-place, there must be enough room to allow for changes in the size of the version component values—otherwise an error will be reported to the diagnostic file. Case is ignored, and C comments are skipped when searching for the characters "#defineΔVersion" in the source. The **-verid** option may be used to search for a different `#define` identifier if desired.

-d

Write the (updated) version component values contained in the resource string to the standard output file. For `'vers'` resources, the display indicates which values go with which resource (that is, `'vers'`,1 and/or `'vers'`,2).

-fmt *nf.mf*

For the SetVersion string resource only, format the version and revision values according to the specified format. The format of the resource is changed only if the version and/or revision is actually changed (**-sv, -v, -sr, -r**). The format is specified as *nf.mf*, where *f* is either of the letters D or Z, and *n* and *m* are integer values from 1 to 10, which specify the field widths of the version and revision numbers, respectively. If the version or revision value is larger than the specified field width, the width is enlarged to contain the entire value. Each field is independently padded up to the specified width with leading zeros or blanks according to the setting of *f*. D indicates leading blanks, and Z indicates leading zeros. For example, a format of 1Z.3Z for a version/revision value of 10.2 would be formatted as d10.002. The default format is 1Z.1Z. Only the version format (*nf*) or revision format (*.mf*—the period is required) need be specified, allowing the other value to format according to the default.

-i *resid* The resource ID is the specified *resid*. The default is to use a resource ID of 0 if the **-t** option does not specify that a `'vers'` resource is to be processed. If the **-t** option does specify `'vers'`, then **-i** must specify 1 or 2. With **-t** `'vers'`, omitting **-i** implies that *both* `'vers'`,1 and `'vers'`,2 are to be updated. Indeed, this option must be omitted when the **-sync** option is specified.

-p Write SetVersion's version number to the diagnostic output file. You can use the **-d** option just to output the resource information to the standard output file. The **-p** option also displays this information, but to the diagnostic output file.

-prefix *prefix*

Set the prefix string on the version. The *prefix* can be any sequence of characters that does not contain a "." or "/" and that does not end with a digit (0–9), a blank., "%", or "Δ" (a blank could be inserted by choosing an appropriate **-fmt** format with leading blanks for the version number). Once the prefix is set, you can change it only by specifying another **-prefix** string. Alternatively, you can remove the prefix by specifying the *prefix* as a period (".). Note that the **-prefix** option is allowed only for SetVersion's string resource.

-[p]source *file*

Update the string constant in the Pascal source specified by the *file*. The constant is set to be the same as that specified by SetVersion's string resource or the short string from a `'vers'` resource. It is assumed that the constant is defined in a CONST section somewhere in the first 12,800 characters (twenty-five 512-byte blocks) of the file as follows:

```
Version = 'ver.rev...';ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ{some comment}
```

The Δs indicate required spaces (spaces or tabs may surround the "="). There may be any number of spaces before the *required* comment. However, because SetVersion edits the line in-place, there must be enough room to allow for changes in the size of the version component values—otherwise an error will be reported to the diagnostic file. Case is ignored and Pascal comments are skipped when searching for the "Version" identifier in the source. The **-verid** option can be used to search for a different identifier if desired.

-r Increment the revision component by 1.

-rezsource *file*

Update the resource definition in the resource compiler source specified by the *file*. The definition is set to be the same as that specified by SetVersion's resource string or `'vers'` resource definition. It is assumed that the definition is somewhere in the first 12,800 characters (25 512-byte blocks) of the file and is specified as described in the following general format:

For a SetVersion string resource,

```
type 'MPST' as 'STRΔ';
resourceΔ'MPST' (0) {
  "Version ver.rev..."ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ/*some comment*
};
```

For a Finder `'vers'` resource (see the Description section above for the meaning of these fields),

```
resourceΔ'vers'Δ(i) {
  0xVV,Δ0xRB,ΔS,Δ0xXX, countryCode,ΔΔΔΔ/*some comment*
  "ver.rev...",ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ/*some comment*/
  "long version message"ΔΔΔΔΔΔΔΔΔΔΔΔΔΔ/*some comment*/
};
```

The Δs indicate required spaces. There may be any number of spaces before the *required* comment(s). Because SetVersion edits the line in-place, there must be enough room to allow for changes in the size of the fields—otherwise an error will be reported to the diagnostic file. Case is ignored and Rez comments are skipped, when searching for the characters "resourceΔ'MPST'" or "resourceΔvers'Δ(*i*)" in the source. Note that because this is a resource definition destined to be placed in a file's resource fork, this option defines the actual resource that SetVersion will seek in the file. The "Version" in the `'MPST'` resource here is fixed and *not* controlled by the **-verid** option.

-sb *bugfix* Set the bug fix component to the specified *bugfix* integer value. This option is only allowed when **-t** `'vers'` is specified to manipulate a Finder `'vers'` resource. The bug fix component is suppressed if its value is 0.

-sr *revision* Set the revision component to the specified *revision* integer value.

-stage *stage* Set the release stage for a `'vers'` resource. The *stage* can be specified as **dev[elopment]**, **alpha**, **beta**, or **rel[ease]**. This is used to set the release letter in the version number as *d*, *a*,*b*, or null, respectively. For the release stage, the nonrelease level component (as modified by the **-x** or **-sx** options) is suppressed.

-suffix *suffix*

Set the suffix string on the revision. The *suffix* can be any sequence of characters that does not contain a "." or "/" and does not begin with a digit (0-9), a blank., "%", or "Δ". Once the suffix is set, it can be changed only by specifying another **-suffix** string, and it can be removed by specifying the *suffix* as a period (**.**"). Note that the **-suffix** option is allowed only for SetVersion's string resource.

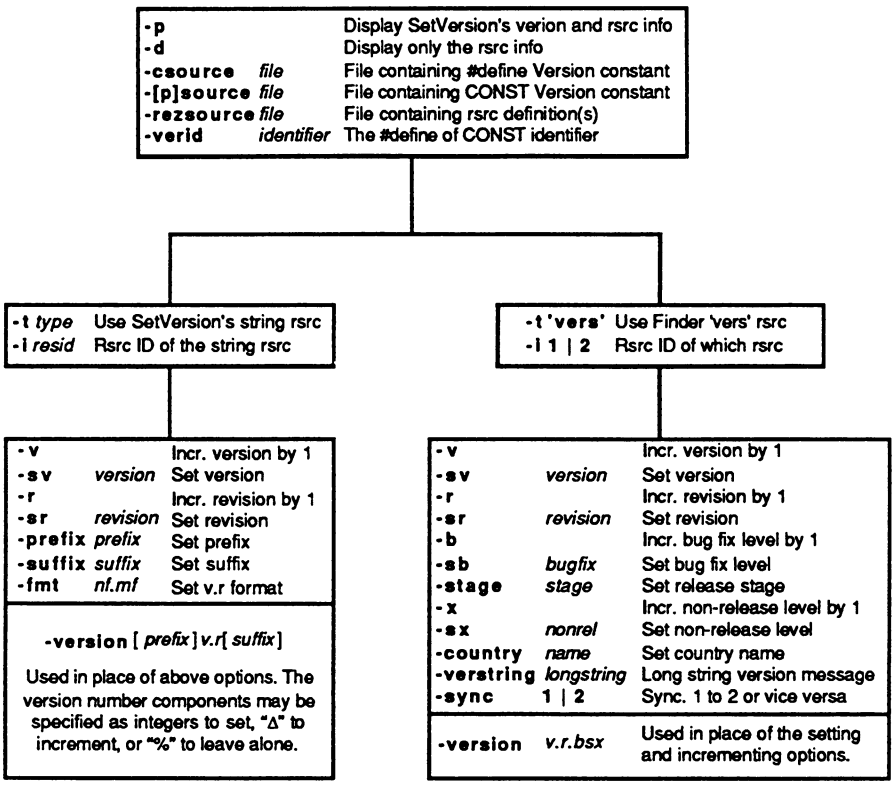
-sv *version* Set the version component to the specified *version* integer value.

-sx *nonrel* Set the nonrelease component to the specified *nonrel* integer value. This option is allowed only when **-t** `'vers'` is specified to manipulate a Finder `'vers'` resource.

-sync 1 | 2 Synchronize the `'vers'`,1 resources with `'vers'`,2 resource or vice versa. Specify 1 or 2 to indicate which `'vers'` resource is to be used as the *master* copy. The short version string and BCD values are copied from the master. The country code is not changed. If there is a version number in the long string, it too is modified to be the same as that of the master copy. Other options can also be specified to modify the master copy prior to copying it to the other resource. Because both `'vers'`,1 and `'vers'`,2 are modified using this option, the **-i** option must be omitted, implying that both resources are to be changed.

-t *type* Use the specified resource *type* instead of `'MPST'` for SetVersion's string resource or `'vers'` to indicate that a Finder `'vers'` resource is to be manipulated. You can use the **-i** option to specify the resource ID. For `'vers'`, **-i** *must* specify 1 or 2 or be omitted entirely if a Finder `'vers'` resource is to be manipulated. Since the **-t** option controls which resource type is to be processed, it also controls which options are valid. Because SetVersion has so many options, the following chart summarizes which options are accepted for which resource type:

SetVersion's Option Summary as a Function of the -t option



-v Increment the version component by 1.

-verid *identifier*

Use the specified constant identifier when searching for the **-[p]source** `CONST` identifier or **-csource** `#define` identifier.

-version *fmtstring*

This option can be used *in place* of the individual version number component specification options (**-sv**, **-v**, **-sr**, **-r**, **-sb**, **-b**, **-stage**, **-sx**, **-x**, **-prefix**, and **-suffix**). Its use is mainly intended for explicit use of the SetVersion tool, while the individual component options are more useful in scripts and makefiles where macros can be used to define individual component parameters (see examples below). The format string *fmtstring* specifies the format of the version number and whether the individual components are to be set, incremented, or left alone. The format string has the general format "v.rbsx", where *v* is the version component, *r* the revision component, *b* the bug fix level component, *s* the release stage letter (*d,a,b*, or omitted for release stage), and *x* the nonrelease level component (omitted for release stage). For SetVersion's string resource, the format string may include a prefix and/or suffix as well, but only a version and revision component are allowed. Each of the component fields (except a prefix and suffix) can be any of the following:

- An integer (or appropriate letter for the stage component). This corresponds to using the **-sv**, **-sr**, **-sb**, **-stage**, and **-sx** options.
- The character "%" to indicate that the corresponding component is to be left alone.
- The character "Δ" to indicate that the corresponding component is to be incremented. This corresponds to using the **-v**, **-r**, **-b**, and **-x** options.

-verstring *longstring*

Set the long version string of a Finder `'vers'` resource to the string specified by the *longstring*. A"^" character placed in this string indicates where to place the version number each time SetVersion updates it. The short message string is used as the string to insert into the position specified by the "^".

-x

Increment the nonrelease level by 1. This option is only allowed when **-t** `'vers'` is specified to manipulate a Finder `'vers'` resource. The nonrelease component is suppressed for the release stage (**-stage release**).

Examples `setversion -d -sv 1 -r Example -psource Globals -rezsource Example.r`

The MPW tool Example contains a SetVersion `'MPST'` string resource. The above command line increments the revision for the tool (**-r**) in the resource fork of the file Example. The version is fixed at 1 (**-sv**), so that Example displays the version and revision as "1.*rev*". The Pascal `include` file, Globals, contains the tool's global declarations, including the Version string. This `include` file is updated to match the `'MPST'` resource (**-psource**). The resource definitions for the tools, in Example.r, will be similarly updated (**-rezsource**). Finally, this command displays the new version of the standard output file (**-d**).

```
setversion -d -version 1.Δ Example -psource Globals -rezsource Example.r
```

Same as previous example, but here we illustrate how the **-version** option serves the same purpose as the **-sv** and **-r** options. Here the "Δ" indicates that the revision is to be incremented.

```
setversion -d -version 1.2.%bΔ Example -psource Globals ∂
        -rezsource Example.r
```

Again an `'MPST'` SetVersion string resource is to be generated. But here we use a more complex version number. The version is set to 1, the revision to 2, the bug fix level is left alone ("%"), this is a beta (*b*) release, and finally the nonrelease level is to be incremented.

```
SetVersion SetVersion -psource SetVersion.p -version 3.Δ -t vers -i 1 -d
        -verstring "^, ©Apple Computer, Inc. 1984-1988, by Ira L. Rubei
SetVersion SetVersion -version 3.0b1 -t vers -i 2 -verstring "MPW 3.0b1"
```

This pair of SetVersion commands generates both Finder `'vers',1` and `'vers',2` resources. The Finder Get Info display shown earlier illustrates the result of using these commands. The MPW tool, SetVersion, has its own version number, 3.Δ (the revision is incremented for version 3) set as a `'vers',1` resource (**-t** `'vers'`, **-i** 1). A long version message is specified by the **-verstring** option. The version number from the short message string is inserted into the long string at the position indicated by the "^" character. The generated version number is displayed to the standard output (**-d**) file. It is also used to update the Pascal source file constant (**-psource**).

The second SetVersion command set the `'vers',2` resource (**-t** `'vers'`, **-i** 2). The version is set unconditionally to 3.0b1 and the long message string to "MPW 3.0b1". MPW 3.0b1 is the MPW release, and SetVersion is just one of the files that belong to this release.

The last example illustrates how both `'vers'` resources should be used. The `'vers',1` resource is the individual file version while the `'vers',2` is the version release of a product that "owns" the file. The last example also should give some idea of how to arrange makefiles, specifically makefile macro definitions, to make the version numbering automatic and general. The following example illustrates this. It is the actual macro definitions and the SetVersion calls used to build SetVersion itself. They are taken as is from SetVersion's makefile.

```
MPWversion     = 3.0b1                  # product release version
Copyright      = ©Apple Computer, Inc.  # copyright notice
ver2           = MPW {MPWversion}       # long msg string for 'ver',2
ver1           = ^, {Copyright}         # long msg string for 'ver',1
- - -
SetVersionVer = -sv 3 -r                # SetVersion's component controls
Stage          = -stage rel -sb 0       # Stage used by tools in makefile
- - -
SetVersion {LinkedTools}SetVersion -psource {ToolsDir}SetVersion.p ∂
           {SetVersionVer} -t vers -i 1 {stage} -d ∂
           -verstring "{ver1} 1984-1988, by Ira L. Ruben"
SetVersion {LinkedTools}SetVersion -version {MPWversion} -t vers -i 2 ∂
           -verstring "{ver2}"
- - -
```

The macro definitions specify the common aspects of the build; that is,

- {MPWVersion}—the MPW release (which can be changed by a Make **-d** option when Make is called).
- {Copyright}—the copyright string (which is concatenated into the `'vers',1` long message string).
- {ver1}—the long string for the `'vers',2` resource (note it uses the MPW release string—we could have used a "^" here), which is to be displayed at the top of the Finder's Get Info window.
- {ver2}—the long string for the `'vers',1` resource (here we do use the "^"), which is to be displayed as the tools' individual version number (we use only version and revision numbers).
- {SetVersionVer}—a macro that defines the numbering control for the individual tool (the makefile is used to make other tools so there is one of these for each individual tool made).
- {Stage}—Used just to insure that only *ver.rel* is generated in the `'vers',1` resource.

The two SetVersion calls are similar to the previous example, but here they are part of a makefile, and we use the macros.

Shift—renumber script parameters

Syntax Shift [*number*]

Description Shift renames the command script positional parameters {*number*+1}, {*number*+2}... to {1}, {2}, and so on. If *number* is not specified, the default value is 1. Parameter 0 (the command name) is not affected. The variables {Parameters}, {"Parameters"}, and {#} variables are also modified to reflect the new parameters.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status These status codes may be returned:

0 Success.
1 Syntax error.

Options None.

Examples The following script repeats a command once for each parameter:

```
### Repeat - Repeat a command for several parameters ###
#
#          Repeat command parameter...
#          Execute command once for each parameter in the
#          parameter list. You can specify options by
#          including them in quotes with the command name.
#
Set cmd "{1}"
Loop
          Shift
          Break If "{1}" == ""
          {cmd} "{1}"
End
```

In the preceding example, the Shift command is used to step through the parameters. The Break command tells the loop when all the parameters have been used. You might, for example, use the following Repeat script to compile several C programs with progress information:

```
Repeat 'C -p' Sample.c Count.c Memory.c
```

See also "Parameters" in Chapter 5.

Shutdown—shutdown or software reboot

Syntax Shutdown [-y | -n | -c] [-r]

Description Shutdown quits MPW and then either shuts down or reboots the Macintosh. The default is shutdown. Before rebooting the computer, the system executes standard quit procedures, asking for confirmation to save modified files, close all windows, and so on.

- *Note:* Under MultiFinder, Shutdown does not give other active applications the chance to save their documents.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status These status codes may be returned:

1 Syntax error.
2 Command aborted.

- *Note:* Shutdown cannot return a status of 0 because if there are no errors the command never returns.

Options

-y Answer "Yes" to any confirmation dialog that occurs.

-n Answer "No" to any confirmation dialog that occurs.

-c Answer "Cancel" to any confirmation dialog that occurs.

-r Restart the machine.

Example

```
Shutdown -y
```

Shuts down the machine, answering "Yes" to any dialogs such as those prompting to save files.

See also Quit command.

SizeWindow—set a window's size

Syntax SizeWindow [**h v**] [*window*]

Description Sets the size of the specified *window* to be **h** by **v** pixels, where **h** and **v** are nonnegative integers referring to the horizontal and vertical dimensions, in that order. (Use a blank space to separate the numbers **h** and **v** on the command line.) The default *window* is the target (second from the front) window; a specific *window* can optionally be specified. If the size specified would cause the window to be too big for the screen, an error is returned.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status SizeWindow may return the following status codes:

0 No errors.
1 Syntax error (error in parameters).
2 The specified window does not exist.
3 The **h v** size specified is too big.

Options None.

Examples

```
SizeWindow 200 200
```

Makes the target window 200 pixels square in size.

```
SizeWindow {Active}
```

A SizeWindow command with no parameters displays the size of the specified window:

```
SizeWindow 500 100 "{Worksheet}"
```

Makes the Worksheet window 500 x 100 pixels in size.

See also MoveWindow, RotateWindows, StackWindows, TileWindows, and ZoomWindow commands.

Sort—sort or merge files

Syntax Sort [*options...*] [*files...*]

Description Sort sorts or merges the specified files and prints the result on the standard output. If no input files are specified, standard input is assumed.

Fields and Field Specifications

The **-f** option (see "Options") precedes a comma-separated list of field specifications. Lines are sorted by extracting and comparing the fields in the order specified until a comparison yields inequality. If a field exists in one line but not the other, the line that possesses the field wins. If neither line has a field, the lines are considered equal. Fields not sorted are output randomly (Sort is not a stable sort).

Each of the field specifications takes one of the forms:

```
[F] [.C] [-K] [modifiers]
[F] [.C] [+N] [modifiers]
```

F is a field number, C and -K are column numbers, and +N is a character count. Any of the items may be omitted, provided that at least one item appears. The numbers -K and +N are mutually exclusive. Spaces can appear anywhere in the specification (except within numbers), but they must be Shell-quoted.

Fields are numbered from 1. A **field** is a string of characters surrounded by newlines or field separator characters (usually whitespace; see the **-fs** option). Typically field 1 would be the first word on the line, field 2 the second word, and so on. Field 0 represents the entire line and is the default if a field number is not specified. Field separator characters are treated as normal text (not separators) in field 0.

Columns are numbered from 1. If .C is specified, it represents a starting offset into the field, taking into account the (file-dependent) varying width of tab characters, if necessary. .C defaults to 1 if it is not specified.

If -K is specified it represents the last column to be included in the field. It defaults to infinity (the maximum K possible) if not specified. Except for field 0, fields are always terminated by field-separator characters, so a large K does not mean "the rest of the line."

If +N is specified, it represents the number of characters to be included in the field (this differs from -K in that tabs are always counted as single characters). It defaults to infinity (the maximum N possible) if not specified.

Here is a short description of all possible field specifications:

F	The entirety of field F.
F.C	Columns C...∞ in field F.
F.C-K	Columns C...K in field F.
F.C+N	N characters starting at column C in field F.
F-K	Columns 1...K in field F.
F+N	The first *N* characters in field F.
.C	Columns C...∞ in the whole line.
.C-K	Columns C...K in the whole line.
.C+N	*N* characters starting at column C in the whole line.
-K	Columns 1...K in the whole line.
+N	The first *N* characters of the whole line.

A field specification may be followed by one or more modifier characters:

r	Reverse order of comparison (reverses **-r**).
b	Ignore leading blanks (reverses **-b**).
q	Interpret quotes when extracting field (reverses **-quote**).
d x t l u	Treat field as decimal (**d**), hexadecimal (**x**), normal text (**t**), lowercase text (**l**) or uppercase text (**u**). These modifiers are mutually exclusive.

These modifiers override the corresponding command line options on a field-by-field basis (**r**, **q**, and **b** flip the meaning of **-r**, **-quote**, and **-b**).

When sorting multiple files, each file can have its own tab setting. When comparing column-aligned fields, Sort correctly handles tabs of varying width, even when comparing records from different files.

Type

Tool.

Input

The specified files, or standard input if no files are specified.

Output

If sorting or merging, the concatentation of all specified input files is performed, sorted by the specified fields.

If **-check** is specified, no output is generated; the exit code of the tool indicates whether the input was presorted.

Diagnostics

Errors are written to diagnostic output.

Status

These status codes may be returned:

0 No errors.
1 Syntax error on command line.
2 Any other error.
4 Out of memory.
5 Input is not sorted.

Options

-b Skip leading blanks in each field.

-check Do not sort, but check if the input is already sorted. Exit with status 0 if the input is sorted; exit with status 5 if the input is not sorted. No output is generated.

-d Sort fields as decimal numbers. The numbers can be of arbitrary length.

-f *field1*[,*field2*...]
Specify fields to sort by. The default field specification is to sort entire lines as text. (See the discussion on field specifications above.)

-fs *string* Specify the field-separator characters. The default field separators are space, tab, backspace, and form feed. Fields may not cross newlines. This switch completely replaces the default set of separators with the specified set.

-l Convert characters to lowercase before comparing them.

-merge Assume each input file is already sorted and merge the input files into the output file. If one or more of the input files is not sorted, the output will not be sorted.

-o *file* Specify the output file (default is standard output). With this option it is possible to sort (though not to merge) a file "in place;" the output file can be one of the input files.

-p Print version and progress information.

-quote Modify field extraction by ignoring field separators enclosed in single and double quotation marks. Characters preceded by the Shell quoting character (∂) are properly escaped. Quotation marks themselves are ignored in comparisons, and sets of alternating quotes (such as ' " ' ...stuff... ' " ') can be nested to any depth. If a quote "dangles" (there is no matching quote before the end of the line), the field extends to the end of the line.

-stdin This option serves as a place holder for the standard input, making it possible to sort or merge standard input with other files.

-r Reverse order of comparison.

-t Sort fields as text (default).

-u Convert characters to uppercase before comparing them.

-unique Output lines that are identical (with respect to the fields specified with the **-f** option) are printed only once.

-x Sort fields as hexadecimal numbers (upper- or lowercase). The numbers can be of arbitrary length. A leading dollar sign (`$`) or '`0x`' is ignored as whitespace.

Examples

```
Sort Able -stdin Baker -o Output
```

Sort the files Able, Baker, and the standard input, with output to file Output.

```
Sort -x -f '2.2+8, ltr' Frog
```

Sort the file Frog. The first key to sort on consists of eight characters starting at the second column of the second field, treated as a hexadecimal number. The second key to sort on is merely the text of the first field, in reverse order.

```
Sort -p -merge -u one two three infinity
```

Merge the specified files, treating lowercase characters as uppercase. Print version and progress information.

StackWindows—arrange windows diagonally

Syntax StackWindows [-h *num*] [-r t, l, b, r] [-v *num*] [-i | *window...*]

Description Automatically sizes and moves all of the open Shell windows (except the Worksheet) so that they are staggered diagonally across the screen. Use StackWindows when selecting windows from the Window menu; this makes dealing with many open windows easier.

If no *windows* are specified, all open Shell windows (except the Worksheet) are stacked up. Additionally you can specify the horizontal and vertical staggering constants; otherwise staggering defaults to five pixels horizontally and 20 pixels vertically. You can also include the Worksheet by using the **-i** option.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status These status codes may be returned:

0 No errors.
1 Syntax error (in parameters).

Options

-i Include the Worksheet window when stacking if there is no list of windows specified.

-h *num* Stack the specified windows with a horizontal spacing of *num* pixels. The default horizontal spacing is five pixels wide. Negative values are not allowed; they return a syntax error.

-r *t,l,b,r* Stack the specified windows within the specified rectangle. The rectangle is specified by the coordinates for top, left, bottom, and right. The default rectangle is the entire screen, minus the menu bar. The coordinates of the rectangle are separated by commas. If spaces are included, the rectangle must be enclosed in quotation marks, such as "10, 10, 300, 500". The coordinates (0,0) are located at the left side of the screen below the menu bar.

-v *num* Stack the specified windows with a vertical spacing of *num* pixels. The default vertical spacing is 20 pixels high—the height of a window's title bar. Negative values are not allowed; they return a syntax error.

Example

```
StackWindows
```

Stacks all of the Shell windows, excluding the Worksheet, in a neat and orderly fashion.

```
StackWindows -i -v 20 -h 10 "{active}" "{target}"
```

Stacks the top two windows, including the Worksheet, with a vertical spacing of 20 pixels and a horizontal spacing of 10 pixels.

See also MoveWindow, RotateWindow, SizeWindow, TileWindows, and ZoomWindow commands.

Target—make a window the target window

Syntax Target *name*

Description Makes window *name* the target window for editing commands (that is, the second window from the front). If window *name* isn't already open, then file *name* is opened as the target window. If *name* doesn't exist, an error is returned.

Type Built-in.

Input None.

Output None.

Diagnostics Error messages are written to diagnostic output.

Status These status codes may be returned:

0 No errors.
1 Error in parameters.
2 The specified file does not exist.
3 System error.

Options None.

Example

```
Target Sample.a
```

Makes the window Sample.a the target window.

See also Open command.

"Editing With the Command Language" in Chapter 5.

TileWindows—arrange windows in tile pattern

Syntax TileWindows [-h | -v] [-r *t,l,b,r*] [-i | *window...*]

Description TileWindows automatically sizes and moves the specified Shell windows so that they are all visible on the screen at once. If no windows are specified, all open windows are tiled (except the Worksheet). Arranging your open windows like tiles and then zooming a selected window to full size makes dealing with many open windows much easier.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status TileWindows may return these status codes:

0 No errors.
1 Syntax error (error in parameters).

Options

-i Include the Worksheet window when tiling if you have not specified a list of windows.

-h Tile the specified windows in a horizontal fashion, allowing the full width of the screen to be used by each window. The result is a stack of short, wide windows.

-r *t,l,b,r* Tile the specified windows within the specified rectangle. The rectangle is specified by the coordinates for top, left, bottom, and right. The default rectangle is the entire screen, minus the menu bar. The coordinates of the rectangle are separated by commas. If spaces are included, the rectangle must be enclosed in quotation marks, such as "10, 10, 300, 500". The coordinates (0,0) are located on the left side of the screen below the menu bar.

-v Tile the specified windows in a vertical fashion to see more lines of a document. The result is a row of tall, thin windows.

Examples

```
TileWindows
```

Arranges all of the Shell windows in a tile pattern, so that all are visible.

```
TileWindows -h hd:new:main.c hd:new:foo.c
```

Arranges the specified windows as two long horizontal strips.

```
TileWindows -v "{active}" "{target}"
```

Arranges the top two windows vertically.

See also MoveWindow, RotateWindow, SizeWindow, StackWindows, and ZoomWindow commands.

TransferCkid—move projector information

Syntax TransferCkid *sourcefile destinationfile*

Description Move the Projector 'CKID' resource from *sourcefile* to *destinationfile.*

See Chapter 7 for complete definitions of the terms and symbols used in Projector commands.

Type Script.

Input None.

Output None.

Diagnostics Errors and warnings are written to diagnostic output.

Status The following status codes may be returned:

0 No errors.
1 Syntax error.
2 Error in processing.

Options None.

See Also OrphanFiles.

Translate—convert selected characters

Syntax Translate [*option...*] *src* [*dst*]

Description Standard input is copied to standard output, with input characters specified in the *src* (*source*) parameter string mapped into the corresponding characters specified by the *dst* (*destination*) parameter string; all other characters are copied as is. If *dst* is omitted, all characters represented by the *src* are deleted. If the *dst* string is shorter than the *src*, all characters in the *src* that would map to or beyond the last character in the *dst* are mapped into the last character in *dst*, and adjacent instances of such characters in the input are represented by a single instance of the last character in *dst*.

Both *src* and *dst* are specified as a standard Shell *character list* but not enclosed in square brackets. Thus the *src* and *dst* are sequences of one or more characters (that is, an abcde) or a range of characters separated by a minus sign (that is, a–z, 0–9). Standard escape characters (such as ∂t, ∂n, ∂f) are processed by the Shell command interpreter. In order to specify a minus sign, place it last in the character list. Finally, the *src* character list may be preceded by a ¬ to negate the list; that is, all characters *except* those in the *src* are taken as the *src* string. Thus they are all deleted if *dst* is absent, or collapsed if the last character in *dst* is present.

- *Note:* Case sensitivity of letters specified in the *src* list are governed by the {CaseSensitive} Shell variable. If CaseSensitive is set to 1, then only letters specified in the *src* are mapped or deleted. If CaseSensitive is 0, then uppercase and lowercase letters not explicitly mapped into *dst* characters are mapped identically.

Type Tool.

Input All input is read from the standard input file.

Output The translated input file is written to standard output.

Diagnostics Errors are written to diagnostic output.

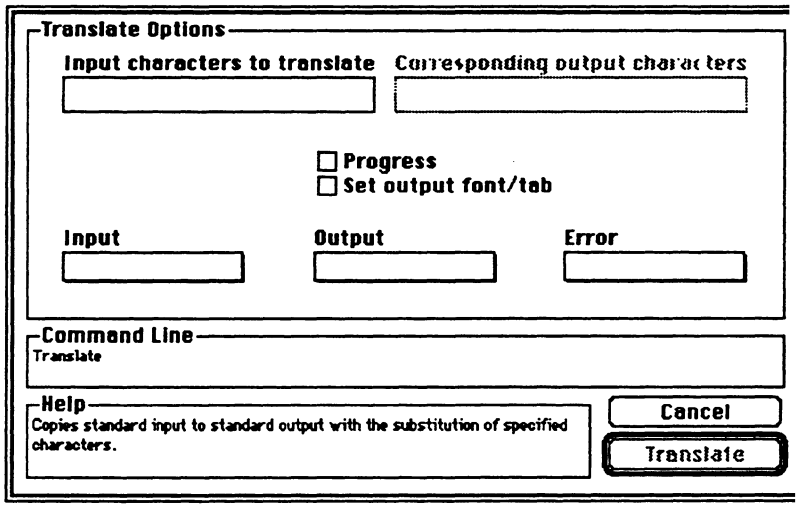
Status Translate may return these status codes:

0 Normal termination.
1 Parameter or option error.

Options

-p Write Translate's version information to the diagnostic file.

-s Set the output file's tab, font, and font size to the same as those of the input file.



Examples

```
translate a-z A-Z <origFile >ucFile
```

Converts all lowercase letters in origFile to uppercase and writes the translated file to ucFile.

```
translate 0-9 9 <origFile >outFile
```

Converts each string of digits in origFile to the single digit 9 in outFile.

```
translate " ∂t∂n" ∂n <origFile >outFile
```

Converts each run of blanks, tabs, or newline (return) characters in origFile to a single newline character in outFile. This effectively produces an output with just one word on each line. Note that the *src* string had to be quoted to specify the blank.

```
translate ¬a-zA-Z∂n " " <origFile >outFile
```

Removes all punctuation and isolates words by spaces on each line. Here we negated the *src* character list. Thus all characters except letters and newline characters are replaced with spaces.

Unalias—remove aliases

Syntax Unalias [*name...*]

Description Removes any alias definition associated with the alias *name*. (It is not an error if no definition exists for *name*.)

▲ **Caution** If no names are specified, all aliases are removed. ▲

The scope of the Unalias command is limited to the current script; that is, aliases in enclosing scripts are not affected. If you are writing a script that is to be completely portable across various users' configurations of MPW, you should place the command

```
Unalias
```

at the beginning of your script to make sure no unwanted substitutions occur.

Type Built-in.

Input None.

Output None.

Diagnostics None.

Status A status code of 0 (no problem) is always returned.

Options None.

Example

```
Unalias File
```

Remove the alias "File". (This alias is defined in the Startup file.)

See also Alias command.

"Command Aliases" in Chapter 5.

Undo—undo last edit

Syntax Undo [*window*]

Description Undo is the scriptable equivalent of choosing Undo from the Edit menu to reverse the last editing operation. Undo without any parameters acts on the target (that is, the second from the front) window. Optionally a named window can be specified.

- *Note:* Remember that Undo is maintained on a window-by-window basis. Therefore using this command will undo the last edit operation that was performed in the specified window, which may or may not be the last operation actually performed.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to diagnostic output.

Status Undo may return these status codes:

0 No errors.
1 Syntax error (error in parameters).
2 Any other error.

Options None.

Examples

```
Undo
```

Reverses the last edit operation in the target window.

```
Undo "{Worksheet}"
```

Reverses the last edit operation in the Worksheet window.

See also Cut, Copy, and Paste commands.

Unexport—remove a variable definition from export

Syntax Unexport [-r | -s | *name...*]

Description Removes the specified variables from the list of exported variables. The list of exported variables is local to a script, so unexported variables are removed only from the local list.

If no names are specified, a list of unexported variables is written to standard output. The default output of Unexport is in the form of Unexport commands. (A variable that is not exported is considered unexported.)

Type Built-in.

Input None.

Output If no names are given, Unexport writes a list of unexported variables to standard output.

Diagnostics Errors are written to diagnostic output.

Status Unexport may return these status codes:

0 No error.
1 Syntax error.

Options

-r Reverse the sense of the output, causing Unexport to generate Export commands for all unexported variables.

-s Suppress the printing of "Unexport" before the unexported variables.

Examples

```
Set SrcDir "HD:source:"
Export SrcDir # SrcDir is available to scripts and tools
...
Unexport SrcDir
```

Now the variable SrcDir is no longer available to scripts and tools.

```
Unexport -r
Export var1
Export var2
...
```

This example lists all the variables that are not exported. To export them, simply select and execute all the export commands.

To get a list of all the variables that have not been exported, execute this command:

```
Unexport -s
var1
var2
...
varx
```

See also Set and Export commands.

Unmark—remove a marker from a file

Syntax Unmark *name... window*

Description Unmark removes the marker(s) *name...* , from the list of markers available for *window*. When a window is the current active window, the Mark menu item(s) will be adjusted.

Type Built-in.

Input None.

Output None.

Diagnostics Errors are written to the diagnostic output.

Status These status codes may be returned:

0 No errors.
1 Syntax error.
2 Error in processing.
3 System error.

Options None.

Examples

```
Unmark `Markers` "{Target}"
```

Removes all markers associated with the target window.

```
Unmark Proc1 "{Active}"
```

Removes the "Proc1" marker from the active window's marker list. Because {Active} is by definition the current active window, the Mark menu is also adjusted to reflect the deletion of the "Proc1" marker.

Limitation Unmark does not support Undo.

See also "Markers" in Chapter 6.

Unmount—unmount volumes

Syntax Unmount *volume...*

Description Unmounts the specified volumes. A volume name must end with a colon (:). If *volume* is a number without a colon, it's interpreted as a disk drive number. The unmounted volumes cannot be referenced again until remounted. If you unmount the current volume (the volume containing the current directory), the boot volume becomes the current volume.

Type Built-in.

Input None.

Output None.

Diagnostics Error messages are written to diagnostic output.

Status These status codes may be returned:

0 The volume was successfully unmounted.
1 Syntax error.
2 An error occurred.

Options None.

Examples

```
Unmount Memos:
```

Unmounts the volume titled Memos.

```
Unmount 1 2
```

Unmounts the volumes in drives 1 (the internal drive) and 2 (the external drive). (The command `Eject 1 2` would unmount *and* eject the volumes.)

See also Eject and Mount commands.

UnmountProject—unmount mounted projects

Syntax UnmountProject *projects...*

Description Unmount projects mounted under Projector.

See Chapter 7 for complete definitions of the terms and symbols used in Projector commands.

Type Built-in

Input None.

Output None.

Diagnostics Errors and warnings are written to diagnostic output.

Status The following status codes may be returned:

0 No errors.
1 Syntax error.
2 Error in processing.
3 System error.

Option **-a** Unmount all mounted projects.

Examples To unmount all mounted projects use:

```
UnmountProject -a
```

To unmount the projects MyProject and YourProject use:

```
UnmountProject MyProject YourProject
```

See Also MountProject.

Unset—remove Shell variables

Syntax Unset [*name...*]

Description Removes any variable definition associated with *name.* (It's not an error if no definition exists for *name.*)

▲ **Caution** If no names are specified, all variable definitions are removed. This can have serious consequences. For example, the Shell uses the variable {Commands} to locate utilities and applications and uses several other variables to set defaults. The assembler and compilers use variables to help locate `include` files. (For details, see "Variables Defined in the Startup File" in Chapter 5.) ▲

The scope of the Unset command is limited to the current script; that is, variables in enclosing scripts are not affected.

Type Built-in.

Input None.

Output None.

Diagnostics None.

Status A status code of 0 is always returned.

Options None.

Example

```
Unset CaseSensitive
```

Removes the variable definition for {CaseSensitive}. This turns off case-sensitive searching for the editing commands.

See also Set, Export, and Unexport commands.

"Defining and Redefining Variables" in Chapter 5.

Volumes—list mounted volumes

Syntax Volumes [-l] [-q] [*volume...*]

Description For each volume named, Volumes writes its name and any other information requested to standard output. The output is sorted alphabetically. A volume name must end with a colon (:)—if *volume* is a number without a colon, it's interpreted as a disk drive number. If *volume* is not given, all mounted volumes are listed.

Type Built-in.

Input None.

Output Information about the specified volumes is written to standard output.

Diagnostics Errors are written to diagnostic output.

Status These status codes may be returned:

0 No errors.
1 Syntax error.
2 No such volume.

Options

-l List volumes in long format, giving volume name, drive ('–' if offline), capacity, free space, number of files, and number of directories.

-q Don't quote volume names that contain special characters. (The default is to quote names that contain spaces or other special characters.)

Examples

```
Volumes -l
```

will write information such as

```
Name        Drive        Size        Free        Files        Dirs
_____       _____        _____       _____       _____        _____
HD:             3       19171K      14242K          290          33
```

```
Files `Volumes 1`
```

Lists the files on the disk in drive 1 (the built-in 3.5-inch disk drive).

WhereIs—search for files in directory tree

Syntax

WhereIs [-c] [-d] [-v] [-s dir]... *pattern*

Description

Use WhereIs to find the location of all files that contain *pattern* as part of their filename. You can use WhereIs to find files hidden in the directory tree. *Pattern* is a full or partial filename. For example, a pattern of "test" will match TestProg.c, test.c, and Work:OutputTest. WhereIs starts searching in the root directory of the default volume and searches the entire disk. To constrain the search to a portion of a disk, or to specify different disks or multiple disks, use the **-s** option. To list any directories that contain *pattern*, use the **-d** option. To constrain the search to files that completely match *pattern*, use the **-c** option. The **-v** option prints the number of items matched with *pattern*. Matching is not case sensitive, and regular expressions are not supported.

WhereIs lists the full pathname of all files and directories found. Files that contain special characters are quoted.

Type

Tool.

Input

None.

Output

The full pathname of any file that contains *pattern* is written to standard output. Also, the total number of files and directories found is written to standard output.

Diagnostics

Error messages are written to diagnostic output.

Status

These status codes may be returned:

0 No errors.
1 Syntax error.
2 File system error during processing.
3 No matches were found.

Options

-c List only files that match *pattern* completely. (In other words, treat *pattern* as a filename.)

-d Match directories also.

-v Print a summary line that counts the number of items matched.

-s *dir* Normally, WhereIs starts searching in the root directory of the default volume. This option constrains the search to start in *dir*. Multiple starting directories can be specified. (Each directory must be preceded by **-s**.) Since searching a large hard disk may take several minutes, this option can speed up the search when you know the general location of a file.

Examples

```
WhereIs test
```

Find all files that have "test" in their filename. The output would be something like

```
HD:MPW:test.c
HD:MPW:test.c.o
HD:MPW:TestMenu.c
HD:MPW:TestProg.p
```

```
WhereIs -c test.c
```

Find files named test.c. The output (with the same files as the example above) would be

```
HD:MPW:test.c
```

```
WhereIs -d test
```

Find all files or directories that have "test" in their leafname. The output would be

```
HD:MPW:TestDir:
HD:MPW:test.c
HD:MPW:test.c.o
HD:MPW:TestMenu.c
HD:MPW:TestProg.p
```

```
WhereIs -s HD:MPW -s Disk2:Work test
```

Find all files that have "test" in their pathname. Search for the files starting in HD:MPW and also in Disk2:Work.

Which—determine which file the Shell will execute

Syntax

Which [-a] [-p] [*command*]

Description

Determines which command the Shell will execute when *command* is entered. Which looks for commands defined by aliases, Shell built-in commands, and commands accessible through the Shell variable {Commands} (the same order the Shell uses). If *command* is not specified, all paths in the {Commands} variable will be written to standard output, one directory per line. The directories are listed in the order in which the Shell would search for commands. In this case the **-a** and **-p** options have no meaning.

Type

Built-in.

Input

None.

Output

In the case of a tool, application, or script, the full path of the command is written to standard output. If *command* is an alias its definition is written to standard output. If *command* is a built-in command, it is simply echoed back to standard output.

Diagnostics

Errors are written to diagnostic output.

Status

These status codes may be returned:

0 No error.
1 Syntax error.
2 *Command* not found.
3 Other error.

Options

-a All paths to *command* are written to standard output. This option allows the user to determine if there are multiple commands with the same name.

-p Prints progress information as each directory in the variable commands is searched.

Examples

```
Which asm
```

This command outputs something like - `HD:MPW:Tools:asm`. The Shell then executes hd:MPW:Tools:asm when given asm.

```
Which -a makeit
Alias makeit 'make > tmp; tmp'
HD:MPW:Tools:makeit
HD:MPW:Scripts:makeit
```

In this case, there are three different "makeit" commands that the Shell could execute, as determined by current aliases and the {Commands} variable. The Shell executes the first one found (the alias).

```
Which newfolder
newfolder
```

In this case, newfolder is a Shell built-in command.

Windows—list windows

Syntax Windows [-q]

Description Writes the full pathname of each file currently in a window. The names are written to standard output, one per line, from backmost to frontmost.

Type Built-in.

Input None.

Output The list of open windows is written to standard output.

Diagnostics Errors are written to diagnostic output.

Status Windows may return these status codes:

0 No error.
1 Syntax error.

Option

-q Don't quote window names that contain special characters. (The default is to quote names that contain spaces or other special characters.)

Examples

```
Windows
```

Lists the pathnames of all open windows.

```
Print {PrintOptions} `Windows`
```

Prints the pathnames of the open windows, using the options specified by the {PrintOptions} variable. This example uses command substitution: Because the Windows command appears in backquotes (`...`), its output supplies the parameters to the Print command.

```
Echo "Open `Windows` || Set Status 0" > SavedWindows
```

Writes a script in the file SavedWindows that will reopen the current set of open windows. Notice how Echo is used to create the script. The conditional || execution operator restores the status to zero should an error occur while opening the remembered windows. This technique is used in the script Suspend to save the list of open windows.

ZoomWindow—enlarge or reduce a window

Syntax ZoomWindow [-s | -b] [*window*]

Description Zooms the specified *window* according to the option specified. The default *window* is the target (second from the front) window; a specific *window* can optionally be specified. The **-s** option forces the window to zoom back to its small size. The **-b** option forces the window to zoom to its full size. If no option is specified, the window toggles to the other size. ZoomWindow without any options mimics the operation of clicking in the window's zoom box. This command is especially valuable when used in conjunction with StackWindows or TileWindows.

The "full size" window is normally the entire screen. You can change it (for example, prevent it from covering the disk and trash icons) by specifying a rectangle in the Shell variable {ZoomWindowRect}.

Type Built-in.

Output None.

Diagnostics Errors are written to diagnostic output.

Status ZoomWindow may return these status codes:

0 No errors.
1 Syntax error (error in parameters).
2 The specified window does not exist.

Options

-s Zoom the specified window back to its original, smaller size.

-b Zoom the specified window to full screen size.

Examples

```
ZoomWindow
```

Zooms the target window to full screen size if the window was originally in the small size.

```
ZoomWindow -s "{Worksheet}"
```

Zooms the Worksheet window back to its small size.

See also

MoveWindow, RotateWindows, SizeWindow, StackWindows, and TileWindows commands.

{ZoomWindowRect} variable in Chapter 5.

THE APPLE PUBLISHING SYSTEM

This Apple® manual was written, edited, and composed on a desktop publishing system using Apple® Macintosh® computers and Microsoft® Word software. Proof and final pages were created on the Apple LaserWriter® IINTX printer. POSTSCRIPT®, the LaserWriter® page-description language was developed by Adobe Systems Incorporated. The illustrations were created using Adobe Illustrator and some were output to a Linotronic 300.

The illustration on the cover was generated using Adobe Illustrator 88 on a Macintosh® II computer. Some of the images were scanned using an Apple® Scanner and then manipulated in ImageStudio. Initial proofing was done using a QMS color printer. Color separations were done using Adobe separator and output to a Linotronic 300 at standard resolution.

Text type is Apple's corporate font, a condensed version of Garamond. Bullets are ITC Zapf Dingbats®. Some elements, such as programs listings, are set in Apple Courier, a fixed-width font.

APPLE COMPUTER, INC. SOFTWARE LICENSE

PLEASE READ THIS LICENSE CAREFULLY BEFORE USING THE SOFTWARE. BY USING THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, PROMPTLY RETURN THE UNUSED SOFTWARE TO THE PLACE WHERE YOU OBTAINED IT AND YOUR MONEY WILL BE REFUNDED.

1. **License.** The application, demonstration, system and other software accompanying this License, whether on disk, in read only memory, or on any other media (the "Apple Software") and related documentation are licensed to you by Apple. You own the disk on which the Apple Software is recorded but Apple and/or Apple's Licensor(s) retain title to the Apple Software and related documentation. This License allows you to use the Apple Software on a single Apple computer and make one copy of the Apple Software in machine-readable form for backup purposes only. You must reproduce on such copy the Apple copyright notice and any other proprietary legends that were on the original copy of the Apple Software. You may also transfer all your license rights in the Apple Software, the backup copy of the Apple Software, the related documentation and a copy of this License to another party, provided the other party reads and agrees to accept the terms and conditions of this License.

2. **Restrictions.** The Apple Software contains copyrighted material, trade secrets and other proprietary material and in order to protect them you may not decompile, reverse engineer, disassemble or otherwise reduce the Apple Software to a human-perceivable form. You may not modify, network, rent, lease, loan, distribute or create derivative works based upon the Apple Software in whole or in part. You may not electronically transmit the Apple Software from one computer to another or over a network.

3. **Support.** You acknowledge and agree that Apple may not offer any technical support in the use of the Software.

4. **Termination.** This License is effective until terminated. You may terminate this License at any time by destroying the Apple Software and related documentation and all copies thereof. This License will terminate immediately without notice from Apple if you fail to comply with any provision of this License. Upon termination you must destroy the Apple Software and related documentation and all copies thereof.

5. **Export Law Assurances.** You agree and certify that neither the Apple Software nor any other technical data received from Apple, nor the direct product thereof, will be exported outside the United States except as authorized and as permitted by the laws and regulations of the United States.

6. **Government End Users.** If you are acquiring the Apple Software on behalf of any unit or agency of the United States Government, the following provisions apply. The Government agrees:

(i) if the Apple Software is supplied to the Department of Defense (DoD), the Apple Software is classified as "Commercial Computer Software" and the Government is acquiring only "restricted rights" in the Apple Software and its documentation as that term is defined in Clause 252.227-7013(c)(1) of the DFARS; and

(ii) if the Apple Software is supplied to any unit or agency of the United States Government other than DoD, the Government's rights in the Apple Software and its documentation will be as defined in Clause 52.227-19(c)(2) of the FAR or, in the case of NASA, in Clause 18-52.227-86(d) of the NASA Supplement to the FAR.

7. **Limited Warranty on Media.** Apple warrants the disks on which the Apple Software is recorded to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of purchase as evidenced by a copy of the receipt. Apple's entire liability and your exclusive remedy will be replacement of the disk not meeting Apple's limited warranty and which is returned to Apple or an Apple authorized representative with a copy of the receipt. Apple will have no responsibility to replace a disk damaged by accident, abuse or misapplication. ANY IMPLIED WARRANTIES ON THE DISKS, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF DELIVERY. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

8. **Disclaimer of Warranty on Apple Software.** You expressly acknowledge and agree that use of the Apple Software is at your sole risk. The Apple Software and related documentation are provided "AS IS" and without warranty of any kind and Apple and Apple's Licensor(s) (for the purposes of provisions 8 and 9, Apple and Apple's Licensor(s) shall be collectively referred to as "Apple") EXPRESSLY DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. APPLE DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE APPLE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE APPLE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE APPLE SOFTWARE WILL BE CORRECTED. FURTHERMORE, APPLE DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE APPLE SOFTWARE OR RELATED DOCUMENTATION IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY. SHOULD THE APPLE SOFTWARE PROVE DEFECTIVE, YOU (AND NOT APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

9. **Limitation of Liability.** UNDER NO CIRCUMSTANCES INCLUDING NEGLIGENCE, SHALL APPLE BE LIABLE FOR ANY INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES THAT RESULT FROM THE USE OR INABILITY TO USE THE APPLE SOFTWARE OR RELATED DOCUMENTATION, EVEN IF APPLE OR AN APPLE AUTHORIZED REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

In no event shall Apple's total liability to you for all damages, losses, and causes of action (whether in contract, tort (including negligence) or otherwise) exceed the amount paid by you for the Apple Software.

10. **Controlling Law and Severability.** This License shall be governed by and construed in accordance with the laws of the United States and the State of California, as applied to agreements entered into and to be performed entirely within California between California residents. If for any reason a court of competent jurisdiction finds any provision of this License, or portion thereof, to be unenforceable, that provision of the License shall be enforced to the maximum extent permissible so as to effect the intent of the parties, and the remainder of this License shall continue in full force and effect.

11. **Complete Agreement.** This License constitutes the entire agreement between the parties with respect to the use of the Apple Software and related documentation, and supersedes all prior or contemporaneous understandings or agreements, written or oral, regarding such subject matter. No amendment to or modification of this License will be binding unless in writing and signed by a duly authorized representative of Apple.

7/15/91
001-0158-A