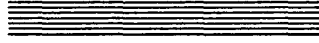




Macintosh®



Macintosh Programmer's
Workshop 2.0 Reference

🍏 APPLE COMPUTER, INC.

This manual and the software described in it are copyrighted, with all rights reserved. Under the copyright laws, this manual or the software may not be copied, in whole or part, without written consent of Apple, except in the normal use of the software or to make a backup copy of the software. The same proprietary and copyright notices must be affixed to any permitted copies as were affixed to the original. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased (with all backup copies) may be sold, given, or loaned to another person. Under the law, copying includes translating into another language or format.

You may use the software on any computer owned by you, but extra copies cannot be made for this purpose.

© 1985, 1986, 1987 Apple Computer, Inc.
20525 Mariani Ave.
Cupertino, California 95014
(408) 996-1010

Pascal Compiler © 1982, 1983, 1984, 1985, 1986, 1987 Apple Computer, Inc.
© 1981 SVS, Inc.

C Compiler © 1984, 1985, 1986, 1987 Green Hills Software, Inc.

Apple, the Apple logo, AppleTalk, ImageWriter, LaserWriter, Lisa, MacDraw, Macintosh, MacPaint, and MacWrite are registered trademarks of Apple Computer, Inc.

AppleShare, MacApp, SANE, and Switcher are trademarks of Apple Computer, Inc.


Motorola is a trademark of Motorola, Inc.

UNIX is a trademark of AT&T Bell Laboratories.


Simultaneously published in the United States and Canada.

MPW sample programs

Apple Computer, Inc. grants users of the Macintosh Programmer's Workshop a royalty-free license to incorporate Macintosh Programmer's Workshop *sample programs* into their own programs, or to modify the sample programs for use in their own programs, provided such use is exclusively on Apple computers. For any modified Macintosh Programmer's Workshop sample program, you may add your own copyright notice alongside the Apple copyright notice.



Contents



Figures and tables xvii

Preface xxi

Power tools for Macintosh programmers xxi

What's new in MPW 2.0 xxii

 New menus xxiii

 New Shell commands xxiii

 New tools xxiv

 Other new features xxvi

What you'll need xvii

 Hardware and system requirements xvii

 System Folder requirements xxvii

 Documentation xxviii

About this manual xxviii

 Syntax notation xxix

Part I Shell Reference 1-1

Chapter 1 System Overview 1-1

The MPW Shell 1-2

 Window commands 1-2

 File management commands 1-3

 Editing commands 1-3

 Structured commands 1-4

 Other built-in commands 1-4

MPW command scripts 1-5

MPW tools 1-5

 Assembler 1-6

 Pascal tools 1-6

 C Compiler 1-7

 Linker 1-7

 Make 1-7

 Resource Compiler and Decompiler 1-8

 Commando 1-8

 Conversion tools 1-8

 Performance measurement tools 1-8

Applications 1-9

 ResEdit 1-9

- Debugger 1-9
- Special command scripts 1-9
- Examples 1-10
 - Sample program files 1-10
 - Command-language examples 1-10
- Overview of MPW files and directories 1-11

Chapter 2 Getting Started 2-1

- Installing the system 2-2
- Starting up 2-3
- Selecting commands from menus 2-4
- Building a program: an introduction 2-4
 - The example programs 2-4
 - Two easy steps 2-5
- Building a new program 2-8

Chapter 3 Using the Shell Menus 3-1

- Features 3-2
- File format 3-2
- Menu commands 3-3
 - Apple menu 3-3
 - File menu 3-3
 - Edit menu 3-6
 - Find menu 3-8
 - Selection expressions 3-10
 - Window menu 3-11
 - Mark menu 3-12
 - Directory menu 3-13
 - Build menu 3-15
 - User-defined menus 3-16

Chapter 4 Using MPW: The Basics 4-1

- Editing 4-2
- Entering commands 4-2
 - Typing commands in a window 4-3
 - The Enter key 4-3
 - Executing several commands at once 4-4
 - Terminating a command 4-5
 - The Help command 4-5
- File management commands 4-7
- File and window names 4-8
 - Selection specifications 4-9
 - Directories and pathnames 4-9
 - Command search path 4-11
 - Changing directories 4-12
 - An aside: the Alias command 4-12
 - Pathname variables 4-12
 - Wildcards (filename generation) 4-13
- Commando dialogs 4-13
 - Invoking Commando dialogs 4-13
 - Using Commando dialogs 4-14

| | |
|--|------|
| Standard dialog box controls | 4-15 |
| Generic text parameters | 4-15 |
| Repeatable options | 4-15 |
| Radio buttons | 4-15 |
| Check boxes | 4-16 |
| Shadow pop-up menus | 4-16 |
| Other pop-up variations | 4-16 |
| Multiple input files | 4-17 |
| Multiple directories | 4-18 |
| Multiple files and/or directories | 4-18 |
| Single input or output file | 4-19 |
| Output file where a file or directory may be specified | 4-19 |
| New directories | 4-20 |
| Special dialog box controls | 4-20 |
| Nested dialog boxes | 4-20 |
| Redirecting output | 4-21 |
| Options dependent on other options | 4-22 |
| Three-state controls | 4-23 |

Chapter 5 Using the Command Language 5-1

| | |
|--|------|
| Overview | 5-2 |
| Types of commands | 5-2 |
| Entering and executing commands | 5-3 |
| Negative status | 5-3 |
| Structure of a command | 5-4 |
| Command name | 5-4 |
| Parameters | 5-4 |
| Command terminators | 5-5 |
| Command continuation | 5-5 |
| Comments | 5-6 |
| Simple versus structured commands | 5-6 |
| Running an application outside the Shell environment | 5-6 |
| Command scripts | 5-7 |
| Special scripts | 5-7 |
| The Startup and UserStartup files | 5-8 |
| Suspend, Resume, and Quit | 5-8 |
| Command aliases | 5-8 |
| Executable error messages | 5-9 |
| Variables | 5-9 |
| Predefined variables | 5-10 |
| Variables defined in the Startup file | 5-11 |
| Parameters to scripts | 5-13 |
| Defining and redefining variables | 5-13 |
| Exporting variables | 5-14 |
| Command substitution | 5-15 |
| Quoting special characters | 5-15 |
| How commands are interpreted | 5-18 |
| Structured commands | 5-19 |
| Control loops | 5-21 |
| Processing command parameters | 5-22 |
| Expressions | 5-23 |
| Filename generation | 5-25 |

| | |
|--------------------------------------|------|
| Redirecting input and output | 5-26 |
| Standard input | 5-27 |
| Terminating input with Command-Enter | 5-28 |
| Standard output | 5-28 |
| Diagnostic output | 5-29 |
| Pseudo-filenames | 5-30 |
| Editing with the command language | 5-31 |
| Defining your own menu commands | 5-32 |
| Sample scripts | 5-32 |
| "AddMenuAsGroup" | 5-32 |
| "CC" | 5-33 |

Chapter 6 Advanced Editing 6-1

| | |
|--|------|
| Editing commands | 6-2 |
| Selections | 6-3 |
| Current selection (§) | 6-5 |
| Selection by line number | 6-6 |
| Position | 6-7 |
| Extending a selection | 6-7 |
| Markers | 6-8 |
| Pattern | 6-9 |
| Pattern matching (using regular expressions) | 6-9 |
| Character expressions | 6-11 |
| Wildcard operators | 6-11 |
| Repeated instances of regular expressions | 6-12 |
| Tagging regular expressions with the @ operator | 6-12 |
| Matching a pattern at the beginning or end of a line | 6-13 |
| Inserting invisible characters | 6-13 |
| Note on forward and backward searches | 6-14 |
| Some useful examples | 6-14 |
| Transforming DumpObj output | 6-14 |
| Finding a whole word | 6-15 |

Chapter 7 Editing Resources With ResEdit 7-1

| | |
|-------------------------------------|------|
| About ResEdit | 7-2 |
| Uses | 7-2 |
| Extensibility | 7-2 |
| Using ResEdit | 7-3 |
| Working with files | 7-3 |
| Working within a file | 7-4 |
| Working within a resource type | 7-6 |
| Editing individual resources | 7-7 |
| 'CURS' (cursor) resources | 7-8 |
| 'DITL' (dialog item list) resources | 7-8 |
| 'FONT' resources | 7-9 |
| 'ICN#' (icon list) resources | 7-11 |
| Creating a resource template | 7-12 |

Chapter 8 Resource Compiler and Decompiler 8-1

- About the Resource Compiler and Decompiler 8-2
 - Resource Decompiler 8-2
 - Standard type declaration files 8-3
 - Using Rez and DeRez 8-3
- Structure of a resource description file 8-4
 - Sample resource description file 8-5
- Resource description statements 8-6
 - Syntax notation 8-6
 - Special terms 8-6
 - Include—include resources from another file 8-7
 - Syntax 8-7
 - AS resource description syntax 8-7
 - Resource attributes 8-8
 - Read—read data as a resource 8-8
 - Syntax 8-8
 - Description 8-8
 - Data—specify raw data 8-9
 - Syntax 8-9
 - Description 8-9
 - Type—declare resource type 8-9
 - Syntax 8-9
 - Description 8-10
 - Data-type specifications 8-10
 - Fill and align types 8-13
 - Array type 8-14
 - Switch type 8-14
 - Sample type statement 8-15
 - Resource—specify resource data 8-16
 - Syntax 8-16
 - Description 8-16
 - Data statements 8-16
 - Sample resource definition 8-17
- Preprocessor directives 8-19
 - Variable definitions 8-19
 - Include directives 8-19
 - If-Then-Else processing 8-20
 - Print directive 8-20
- Resource description syntax 8-21
 - Numbers and literals 8-21
 - Expressions 8-22
 - Variables 8-23
 - Strings 8-24
 - Escape characters 8-25

Chapter 9 Building an Application, a Desk Accessory, or an MPW Tool 9-1

- Overview of the build process 9-2
- The structure of a Macintosh application 9-4
- Linking 9-5
 - What to link with 9-6
 - Linking multilingual programs 9-6
- File types and creators 9-7
- Building a desk accessory or driver 9-7
 - Linking a desk accessory or driver 9-9
 - The desk accessory resource file 9-10
- Modifying the Build menu and makefiles 9-11
 - Variables 9-11
 - Scripts 9-11
 - Files 9-11
 - UserStartup 9-11
 - Modifying the makefiles 9-12
 - Include dependencies 9-12
 - Library object files 9-12
- Building an MPW tool 9-13
 - Linking a tool 9-13

Chapter 10 Advanced Programming Tools 10-1

- Using Make 10-2
 - Format of a makefile 10-2
 - Dependency rules 10-3
 - Double-*f* dependency rules 10-4
 - Default rules 10-5
 - Variables in makefiles 10-7
 - Shell variables 10-7
 - Defining variables within a makefile 10-7
 - Built-in Make variables 10-8
 - Quoting in makefiles 10-8
 - Line continuation character 10-8
 - Comments in makefiles 10-9
 - Executing Make's output 10-9
 - The order in which Make builds targets 10-9
 - Debugging makefiles 10-10
 - Problems due to command generation before execution 10-10
 - Problems with different specifications for the same file 10-10
 - Problems with default rules 10-10
 - An example 10-11
 - Notes on Make's makefile 10-13

- More about linking 10-14
 - Linker functions 10-14
 - Segmentation 10-15
 - Segments with special treatment 10-16
 - Setting resource attributes 10-16
 - Controlling the numbering of code resources 10-17
 - Resolving symbol definitions 10-17
 - Multiple external symbol definitions 10-17
 - Unresolved external symbols 10-17
 - Linker location map 10-18
 - Optimizing your links 10-19
- Library construction 10-19
 - Using Lib to build a specialized library 10-20
 - Removing unreferenced modules 10-20
 - Guidelines for choosing files for a specialized library 10-21

Chapter 11 Debugging With MacsBug 11-1

- About MacsBug 11-2
- Installing MacsBug 11-2
- Theory of operation: a technical aside 11-3
 - The boot process 11-3
 - Memory usage 11-4
 - MacsBug exceptions 11-4
- Using MacsBug 11-5
 - Assembly language 11-5
 - Declaration 11-5
 - Example calls 11-5
 - Pascal 11-6
 - Declaration 11-6
 - Example calls 11-6
 - MPW C 11-6
 - Declaration 11-6
 - Example calls 11-6
- The MacsBug command language 11-7
 - Numbers 11-7
 - Strings 11-7
 - Symbols 11-8
 - Expressions 11-8
- General commands 11-9
- Memory commands 11-10
- Break commands 11-12
- A-trap commands 11-14
- Heap zone commands 11-16
- Disassembler commands 11-18
- MacsBug summary 11-20

| | |
|-------------------|---|
| Chapter 12 | Performance Measurement Tools 12-1 |
| | About performance measurement tools 12-2 |
| | Components of performance tools 12-2 |
| | Requirements for using performance tools 12-3 |
| | How performance measurement works 12-3 |
| | Program Counter sampling 12-3 |
| | A restriction 12-4 |
| | Bucket counts 12-4 |
| | Using performance measurement tools 12-5 |
| | Performance reports 12-6 |
| | Performance output file 12-6 |
| | Analyzing the results with PerformReport 12-9 |
| | Adding identification lines to a data file 12-9 |
| | Interpreting the performance report 12-10 |
| | Implementation issues 12-10 |
| | Locking the interrupt handler 12-10 |
| | Segmentation 12-11 |
| | Dirty code segments 12-11 |
| | Movable code resources 12-11 |
| | Macintosh II ROMs and 24-bit addresses 12-11 |
| | |
| Chapter 13 | Writing an MPW Tool 13-1 |
| | Shell facilities 13-2 |
| | Parameters 13-2 |
| | Environment (Shell) variables 13-4 |
| | Standard I/O channels 13-4 |
| | I/O buffering 13-5 |
| | I/O to windows and selections 13-6 |
| | Signal handling 13-6 |
| | Exit processing 13-7 |
| | Status codes 13-7 |
| | Restrictions 13-8 |
| | Initialization 13-8 |
| | Memory management 13-9 |
| | Heap 13-10 |
| | Stack 13-10 |
| | Windows, graphics, and events 13-10 |
| | Conventions 13-11 |
| | |
| Chapter 14 | Creating a Commando Interface for Tools 14-1 |
| | About Commando 14-2 |
| | Using Commando 14-2 |
| | Resource description 14-3 |
| | Resource ID and name 14-3 |
| | Size of the dialog box 14-3 |
| | Tool description 14-4 |
| | Regular entry 14-5 |
| | Multiregular entry 14-5 |
| | Check boxes 14-6 |
| | Radio buttons 14-8 |

| | |
|---|-------|
| Boxes, lines, and text titles | 14-9 |
| Box | 14-9 |
| TextBox | 14-10 |
| TextTitle | 14-10 |
| Pop-up menus | 14-11 |
| Editable pop-up menus | 14-12 |
| Lists | 14-14 |
| Three-state buttons | 14-15 |
| Icons and pictures | 14-16 |
| Control dependencies | 14-16 |
| Direct dependency | 14-17 |
| Inverse dependency | 14-18 |
| Dependency on the Do It button | 14-19 |
| Multiple dependencies | 14-19 |
| Dependencies on radio buttons | 14-20 |
| Nested dialog boxes | 14-21 |
| Redirection | 14-22 |
| Files and directories | 14-23 |
| Individual files and directories | 14-23 |
| Multiple files and directories for input and output | 14-26 |
| Multiple files and directories for input only | 14-28 |
| Multiple new files | 14-30 |
| A Commando example | 14-30 |

Chapter 15 Writing a Desk Accessory or Other Driver Resource 15-1

| | |
|-------------------------------|------|
| The DRVRRuntime library | 15-2 |
| What your routines need to do | 15-3 |
| Programming hints | 15-4 |
| Sample desk accessory | 15-4 |

Appendix A Macintosh Workshop Files A-1

| | |
|-----------------------------------|-----|
| Distribution files | A-1 |
| Distribution disk MPW1: | A-1 |
| Distribution disk MPW2: | A-2 |
| Distribution disk MPW3: | A-3 |
| MPW Assembler files | A-3 |
| Distribution disk MPW Assembler1: | A-3 |
| Distribution disk MPW Assembler2: | A-3 |
| MPW Pascal files | A-4 |
| MPW C files | A-5 |
| Hard disk configuration | A-7 |

Appendix B Summary of Selections and Regular Expressions B-1

| | |
|-----------------------|-----|
| Selections | B-1 |
| Regular expressions | B-2 |
| Option-key characters | B-3 |

Appendix C MPW Character Reference C-1

Appendix D Resource Description Syntax D-1

- Syntax notation D-1
- Structure of a resource description file D-2
 - Include—include resources from another file D-2
 - Read—read data as a resource D-2
 - Data—specify raw data D-3
 - Type—declare resource type D-3
 - Data-type D-3
 - Fill-type D-4
 - Alignment D-4
 - Switch-type D-4
 - Array-type D-4
 - Resource—specify resource data D-4
- Preprocessor directives D-5
- Syntax D-5
 - Identifiers D-5
 - Token delimiters D-5
 - Compound types D-5
 - Expressions D-5
 - Numbers D-6
 - Variables D-7
 - Strings D-7

Appendix E File Types, Creators, and Suffixes E-1

- File types and creators E-1
- File suffixes E-1
 - Text files E-1
 - Object files E-2

Appendix F Object File Format F-1

- Object file format F-1
- Notation used in this appendix F-2
- Object file records F-3
 - Pad record F-3
 - First record F-3
 - Last record F-3
 - Comment record F-4
 - Dictionary record F-4
 - Module record F-4
 - Entry-point record F-5
 - Size record F-6
 - Contents record F-6
 - Reference record F-7
 - Computed-reference record F-8

Appendix G In Case of Emergency G-1

- Crashes G-1
- Stack space G-1

- Glossary GL-1
- Index IN-1

Part II Command Reference II-1

- Command prototype II-2
- AddMenu—add menu item II-4
- Adjust—adjust lines II-7
- Alert—display an alert box II-8
- Alias—define or write command aliases II-9
- Align—align text to left margin II-10
- Asm—MC68xxx Macro Assembler II-11

- Backup—folder file backup II-16
- Beep—generate tones II-22
- Begin...End—group commands II-23
- Break—break from For or Loop II-25
- BuildMenu—create the Build menu II-26
- BuildProgram—build the specified program II-27

- C—C Compiler II-28
- Canon—canonical spelling tool II-32
- Catenate—concatenate files II-35
- Clear—clear the selection II-36
- Close—close specified windows II-37
- Commando—display dialog box for a command II-38
- Compare—compare text files II-39
- Confirm—display confirmation dialog box II-44
- Continue—continue with next iteration of For or Loop II-46
- Copy—copy selection to Clipboard II-47
- Count—count lines and characters II-48
- CreateMake—create a simple makefile II-49
- Cut—copy selection to Clipboard and delete it II-51
- CvtObj—convert Lisa Workshop object files
to MPW object files II-52

- Date—write the date and time II-54
- Delete—delete files and directories II-55
- DeleteMenu—delete user-defined menus and items II-57
- DeRez—Resource Decompiler II-58
- Directory—set or write the default directory II-61
- DirectoryMenu—create the Directory menu II-62
- DumpCode—write formatted resources II-63
- DumpObj—write formatted object file II-65
- Duplicate—duplicate files and directories II-68

- Echo—echo parameters II-70
- Eject—eject volumes II-71
- Entab—convert runs of spaces to tabs II-72
- Equal—compare files and directories II-74
- Erase—initialize volumes II-76
- Evaluate—evaluate an expression II-77
- Execute—execute a script in the current scope II-80
- Exists—confirm the existence of a file or directory II-81
- Exit—exit from a script II-82
- Export—make variables available to programs II-83

FileDiv—divide a file into several smaller files II-84
 Files—list files and directories II-85
 Find—find and select a text pattern II-87
 Font—set font characteristics II-88
 For...—repeat commands once per parameter II-89
 GetErrorText—display text for system error numbers II-91
 GetFileName—display a standard file dialog box II-93
 GetListItem—display items for selection in a dialog box II-95
 Help—display summary information II-97
 If...—conditional command execution II-99
 Lib—combine object files into a library file II-101
 Line—find a line number II-104
 Link—link an application, tool, or resource II-105
 Loop...End—repeat command list until Break II-110
 Make—build up-to-date version of a program II-111
 MakeErrorFile—create error message textfile II-114
 Mark—assign a marker to a selection II-115
 Markers—list markers II-117
 MDSCvt—convert MDS Assembler source II-118
 Mount—mount volumes II-121
 Move—move files and directories II-122
 MoveWindow—move window to h,v location II-124
 New—open a new window II-125
 Newer—compare modification dates between files II-126
 NewFolder—create a directory II-128
 Open—open a window II-129
 Parameters—write parameters II-130
 Pascal—Pascal Compiler II-131
 PasMat—Pascal program formatter II-134
 PasRef—Pascal cross-referencer II-140
 Paste—replace selection with Clipboard contents II-146
 PerformReport—generate a performance report II-147
 Print—print text files II-148
 ProcNames—display Pascal procedure and function names II-151
 Quit—quit MPW II-154
 Quote—quote parameters II-155
 Rename—rename files and directories II-157
 Replace—replace the selection II-159
 Request—request text from a dialog box II-161
 ResEqual—compare resources in files II-162
 Revert—revert to saved files II-164
 Rez—Resource Compiler II-165
 RezDet—detect inconsistencies in resources II-168

Save—save windows II-170
Search—search files for a pattern II-171
Set—define or write Shell variable II-173
SetDirectory—set the default directory II-175
SetFile—set file attributes II-176
SetPrivilege—set access privileges for folders on file server II-178
SetVersion—maintain version and revision number II-180
Shift—renumber script parameters II-185
Shutdown—shutdown or software reboot II-186
SizeWindow—set a window's size II-187
StackWindows—arrange windows diagonally II-188

Tab—set a window's tab value II-189
Target—make a window the target window II-190
TileWindows—arrange windows in tile pattern II-191
TLACvt—convert Lisa TLA Assembler source II-192
Translate—convert selected characters II-194

Unalias—remove aliases II-196
Undo—undo last edit II-197
Unexport—remove a variable definition from export II-198
Unmark—remove a marker from a file II-200
Unmount—unmount volumes II-201
Unset—remove Shell variables II-202

Volumes—list mounted volumes II-203

Which—determine which file the Shell will execute II-204
Windows—list windows II-206

ZoomWindow—enlarge or reduce a window II-207

Figures and tables

Chapter 1 System Overview 1-1

Figure 1-1 Setup of MPW folders and files 1-11

Chapter 2 Getting Started 2-1

Figure 2-1 Worksheet window 2-3

Figure 2-2 MPW menu bar 2-4

Figure 2-3 Directory menu 2-5

Figure 2-4 Show Directory alert 2-5

Figure 2-5 Build menu 2-6

Figure 2-6 Program Name? dialog box 2-7

Figure 2-7 Finished Sample build 2-7

Figure 2-8 Set Directory standard file dialog box 2-9

Figure 2-9 CreateMake dialog box 2-9

Chapter 3 Using the Shell Menus 3-1

Figure 3-1 File menu 3-3

Figure 3-2 New dialog box 3-4

Figure 3-3 Edit menu 3-6

Figure 3-4 Dialog box of the Format menu item 3-7

Figure 3-5 Find menu 3-8

Figure 3-6 Dialog box of the Replace menu item 3-8

Figure 3-7 Selection by line number 3-10

Figure 3-8 Example of a regular expression 3-10

Figure 3-9 Text selected with the Find command 3-11

Figure 3-10 Window menu 3-11

Figure 3-11 Mark menu 3-12

Figure 3-12 Mark dialog box 3-12

Figure 3-13 Unmark dialog box 3-13

Figure 3-14 Directory menu 3-13

Figure 3-15 Dialog box of the Set Directory menu item 3-14

Figure 3-16 Build menu 3-15

Figure 3-17 CreateMake dialog box 3-15

Figure 3-18 Program Name? dialog box 3-16

Chapter 4 Using MPW: The Basics 4-1

Figure 4-1 Pressing Enter to execute selected text 4-4

Figure 4-2 Help summaries 4-6

Figure 4-3 Hierarchical directory structure 4-10

Figure 4-4 Date dialog box 4-14

Figure 4-5 Rez: the first dialog box 4-20

Figure 4-6 Rez: nested Preprocessor dialog box 4-21

Figure 4-7 Rez: nested Redirection dialog box 4-21

Table 4-1 Basic file management commands 4-7

Chapter 5 Using the Command Language 5-1

| | |
|------------|--|
| Figure 5-1 | Trafficking in variables 5-14 |
| Figure 5-2 | Standard input and output 5-26 |
| Figure 5-3 | Redirecting diagnostic output 5-29 |
| Figure 5-4 | Text highlighted in the active window and target window 5-31 |
| Table 5-1 | Command terminators 5-5 |
| Table 5-2 | Variables defined by the Shell 5-10 |
| Table 5-3 | Variables defined in the Startup file 5-11 |
| Table 5-4 | Parameters to scripts 5-13 |
| Table 5-5 | Special characters and words 5-16 |
| Table 5-6 | Quotes 5-17 |
| Table 5-7 | Special escape conventions 5-17 |
| Table 5-8 | Structured commands 5-19 |
| Table 5-9 | Expression operators in order of decreasing precedence 5-23 |
| Table 5-10 | Filename generation operators 5-25 |
| Table 5-11 | I/O redirection 5-26 |
| Table 5-12 | Pseudo-filenames 5-30 |

Chapter 6 Advanced Editing 6-1

| | |
|------------|-----------------------------------|
| Figure 6-1 | A selection specification 6-5 |
| Figure 6-2 | Selections in two windows 6-5 |
| Table 6-1 | Editing commands 6-2 |
| Table 6-2 | Selection operators 6-4 |
| Table 6-3 | Regular expression operators 6-10 |

Chapter 7 Editing Resources With ResEdit 7-1

| | |
|------------|----------------------------------|
| Figure 7-1 | Disk volume windows 7-3 |
| Figure 7-2 | A File Info window 7-4 |
| Figure 7-3 | A ResEdit file window 7-5 |
| Figure 7-4 | A resource type window 7-6 |
| Figure 7-5 | A Get Info window for ICN#'s 7-6 |
| Figure 7-6 | Editing 'CURS' resources 7-8 |
| Figure 7-7 | 'FONT' editor window 7-9 |
| Figure 7-8 | 'ICN#' window 7-11 |
| Figure 7-9 | Window template data 7-12 |

Chapter 8 Resource Compiler and Decompiler 8-1

| | |
|------------|---|
| Figure 8-1 | Rez and DeRez 8-2 |
| Figure 8-2 | Creating a resource file 8-3 |
| Figure 8-3 | Padding of literals 8-21 |
| Figure 8-4 | Internal representation of a Pascal string 8-24 |
| Table 8-1 | Resource description expression operators 8-22 |
| Table 8-2 | Resource Compiler escape sequences 8-25 |

| | |
|-------------------|--|
| Chapter 9 | Building an Application, a Desk Accessory, or an MPW Tool 9-1 |
| Figure 9-1 | Building a program 9-3 |
| Figure 9-2 | Linking 9-5 |
| Figure 9-3 | Building a desk accessory with DRVRRuntime 9-8 |
| Table 9-1 | Files to link with 9-6 |
| Table 9-2 | File types and creators 9-7 |
| | |
| Chapter 10 | Advanced Programming Tools 10-1 |
| Table 10-1 | Makefile summary 10-2 |
| | |
| Chapter 13 | Writing an MPW Tool 13-1 |
| Figure 13-1 | Parameters in C and Pascal 13-3 |
| Figure 13-2 | I/O buffering 13-5 |
| Figure 13-3 | Memory map 13-9 |
| | |
| Chapter 14 | Creating a Commando Interface for Tools 14-1 |
| Figure 14-1 | Basic template for a Commando dialog box 14-3 |
| Figure 14-2 | Multiregular entry 14-6 |
| Figure 14-3 | Setting the CheckOption default state 14-7 |
| Figure 14-4 | Radio buttons with default setting 14-8 |
| Figure 14-5 | Clicking a button other than the default 14-8 |
| Figure 14-6 | No button specified as set 14-9 |
| Figure 14-7 | TextBox example 14-10 |
| Figure 14-8 | Pop-up menu with default value 14-11 |
| Figure 14-9 | Pop-up menu without default value 14-12 |
| Figure 14-10 | How Font Size dependency is handled 14-13 |
| Figure 14-11 | Font Size pop-up menu with font selected 14-13 |
| Figure 14-12 | List control 14-14 |
| Figure 14-13 | Three-state buttons 14-15 |
| Figure 14-14 | Icon in a Commando window 14-16 |
| Figure 14-15 | Direct dependency 14-17 |
| Figure 14-16 | Inverse dependencies 14-18 |
| Figure 14-17 | Dependency on the Do It button 14-19 |
| Figure 14-18 | Dependencies on radio buttons 14-20 |
| Figure 14-19 | Setting up nested dialog boxes 14-21 |
| Figure 14-20 | Placement of nested dialog buttons 14-22 |
| Figure 14-21 | How to obtain input and output redirection 14-23 |
| Figure 14-22 | Resource description for "individual files and directories" controls 14-24 |
| Figure 14-23 | Examples of "individual files and directories" controls 14-26 |
| Figure 14-24 | Example of multiple input files 14-28 |
| Figure 14-25 | Multiple directories for input 14-29 |
| Figure 14-26 | Example of a "directories" control for multiple input files 14-29 |
| Figure 14-27 | Using the MultiOutputFiles subcase of the case MultiFiles 14-30 |
| Figure 14-28 | A Commando example: frontmost ResEqual dialog box 14-32 |
| Table 14-1 | Summary of recommended sizes for Commando screen elements 14-4 |

Appendix B Summary of Selections and Regular Expressions B-1

| | |
|-----------|-------------------------|
| Table B-1 | Selections B-1 |
| Table B-2 | Regular expressions B-2 |

Appendix C MPW Character Reference C-1

| | |
|-----------|-------------------|
| Table C-1 | MPW operators C-1 |
|-----------|-------------------|

Appendix E File Types, Creators, and Suffixes E-1

| | |
|-----------|-----------------------------|
| Table E-1 | File types and creators E-1 |
|-----------|-----------------------------|

Appendix F Object File Format F-1

| | |
|-------------|-------------------------------|
| Figure F-1 | Record type prototype F-2 |
| Figure F-2 | Pad record F-3 |
| Figure F-3 | First record F-3 |
| Figure F-4 | Last record F-3 |
| Figure F-5 | Comment record F-4 |
| Figure F-6 | Dictionary record F-4 |
| Figure F-7 | Module record F-4 |
| Figure F-8 | Entry-point record F-5 |
| Figure F-9 | Size record F-6 |
| Figure F-10 | Contents record F-6 |
| Figure F-11 | Reference record F-7 |
| Figure F-12 | Computed-reference record F-8 |



Preface

Welcome to Macintosh® Programmer's Workshop 2.0. This preface previews the many new features in MPW 2.0, as well as the new hardware and software requirements. It also gives the syntax conventions used in the rest of this reference.

Power tools for Macintosh programmers

The Macintosh Programmer's Workshop provides professional software development tools for the Apple® Macintosh computer. Briefly, the Macintosh Programmer's Workshop 2.0 consists of the following parts:

- MPW Shell (the programming environment)
- MC68xxx Assembler
- Linker
- Make (for tracking file dependencies)
- Resource editor
- Resource Compiler and Decompiler
- Debugger
- Performance measurement tools
- Commando dialog interface

The system also includes a comprehensive array of additional tools for creating and manipulating text and resource files. The following MPW products are separately available:

- **Macintosh Programmer's Workshop Pascal** provides the additional tools, interfaces, and libraries you need to develop applications, tools, and desk accessories in Pascal.
- **Macintosh Programmer's Workshop C** provides a C Compiler (copyright Green Hills Software) along with the interfaces and libraries needed to develop applications, tools, and desk accessories in C.
- **MacApp™**, the Expandable Macintosh Application. MacApp provides a set of object-oriented libraries that automatically implement the standard Macintosh user interface, thus simplifying and speeding up the process of software development. MPW Pascal is required for use of MacApp.

The entire MPW system is outlined in detail in Chapter 1, "System Overview."

The Macintosh Programmer's Workshop 2.0 provides numerous advantages over previous development systems:

- **Integration:** The various components of the MPW system all run within the MPW Shell environment, which fully adheres to the standard Macintosh interface. The integrated environment enables separately developed applications, called MPW tools, to run in the MPW Shell environment.
- **Automated build process:** A pull-down menu provides several ways to quickly, automatically build or rebuild your programs. You can also automate complex builds by using the Make facility and command-language scripts.
- **Command scripting:** In addition to menu commands, MPW provides a full command language, including Shell variables, command aliases, pipes, and the ability to redirect input and output. You can combine any series of commands into a command file (or "script") for fast, accurate, automatic results.
- **Regular expression processing:** The editor component of the Shell provides powerful search and replace capabilities with regular expressions, which form a language for describing complex text patterns. Regular expressions allow you, for instance, to restructure complex tables with a single command.
- **Extensibility:** You can create your own integrated tools to run within the Shell environment. You can also add your own menu items to the Shell.
- **Ease of use:** The Commando dialog interface gives you immediate on-screen access to all of MPW's versatile options and functions in specialized dialog boxes. This interface makes learning easier and faster. You can compose complex command lines without referring to the manual. You can create a Commando interface for your own tools and scripts as well.

Taken together, these features add up to a level of integration, power, and ease of use not found in any previous microcomputer-based development system.

What's new in MPW 2.0

MPW 2.0 is much faster and easier to use than its predecessor. Many new tools and options to existing tools have been added.

You can quickly and easily build a simple program or desk accessory by using the Build menu. The Commando dialog interface makes the numerous command options available in dialog boxes, with help text on every option instantly available.

The four lists below summarize the changes that have been made since the release of MPW 1.0 and where you can find more information about these changes. The first list describes the changes that are immediately noticeable—the menus. The second list mentions changes and additions to the MPW Shell's command repertoire. The third list introduces the new tools and significant changes to existing tools. The final list itemizes miscellaneous improvements.

New menus

The first thing you'll notice about MPW 2.0 is that the menus have undergone a substantial revision since the release of MPW 1.0:

- **Build menu** lets you build and execute simple programs entirely by using menus, without prior knowledge of many details, such as the command language, compiler, linker and make options. You can rebuild complex programs by selecting a single menu item, after doing some one-time-only setup work. See Chapters 2 and 3.
- **Clipboard control** has been moved from the Window menu to the Edit menu. The Edit menu contains a Show Clipboard item that toggles to Hide Clipboard. See Chapter 3.
- **Directory menu** lets you set the current directory. Use the Build menu to build programs that reside in the current directory. See Chapters 2 and 3.
- **Enter command** has been removed from the Edit menu. It is quicker to click the Status panel in the lower-left corner of the active window or simply to press the Enter key. See Chapter 4.
- **Find options** (searching backwards, selection expressions, and so on) are now contained in the dialog boxes for Find and Replace, as radio buttons and check boxes. A new option, Wrap-Around Search, has been added. There is now more room in the Find menu for adding your own customized menu items. See Chapter 3.
- **Format menu** has been replaced by a Format... command in the Edit menu. Choose Format to display a dialog box listing all of the available fonts in their actual sizes. You can set tabs and auto-indentation, and show invisible settings. Use Command-Y to display the Format dialog box. See Chapter 3.
- **Mark menu** makes it possible to select large portions of a textfile by embedding markers. This menu contains the scriptable commands Mark, Unmark, and Markers. See Chapter 3 and "Selections" in Chapter 6.
- **Menu item styles** (for the windows listed in the Window menu) are new. They supply more information: A standard check mark indicates the active window; a bullet indicates the target window; dirty windows are underlined.
- **Window menu** (formerly Windows) now contains the new commands TileWindows and StackWindows. See Chapter 3 and respective command descriptions in Part II.

New Shell commands

All of these built-in commands are fully described in Part II:

- **Exists:** A new built-in Shell command, Exists determines the existence of a file or directory. See Part II.
- **Mark:** This item in the new Mark menu displays a dialog box asking you to supply a name for a selected text. See Chapters 3 and 6.
- **Markers:** This command displays a list of currently marked texts. See Part II.
- **MoveWindow:** Use this new command to move a window to a specified h,v location. See Part II.
- **Newer:** This built-in Shell command compares two files to see which was modified most recently; the newest filename is displayed. See Part II.

- **Quit:** Use this command (the equivalent of the menu item) to exit MPW at the conclusion of a script. See Part II.
- **Quote:** Like Echo, Quote writes its parameters to standard output, except that Quote writes parameters that contain special characters, enclosing these special characters in single quotation marks. See Part II.
- **Revert:** This command changes your current window back to its condition at the last save. This command is now scriptable. See Part II.
- **Shutdown:** You can now do a scripted shutdown or reboot of your machine. See Part II.
- **SizeWindow:** This command sets a window's size. See Part II.
- **StackWindows:** Use this command to arrange open windows in a diagonal pattern across the screen. See Part II.
- **TileWindows:** You can arrange open windows onscreen in a tile pattern so that each is fully visible. Then select the window you want and use the zoom box control (or its scriptable equivalent, ZoomWindow) to enlarge the window or put it back. See Part II.
- **Undo:** The familiar menu item to undo your last edit is now scriptable. See Part II.
- **Unexport:** This command removes variable definitions from the export list. See Part II.
- **Unmark:** This item in the new Mark menu displays a dialog box asking which marked texts you want to unmark. See Chapter 3.
- **Which:** This new Shell command displays the full pathname for a given file/command/alias. See Part II.
- **ZoomWindow:** You can instantly enlarge or reduce a window by using this scriptable command. It is handy when used in conjunction with TileWindows. See Part II.

New tools

The tools included with the original MPW 1.0 have been improved in many ways for faster performance and increased versatility. In addition to these improvements, the following new tools have been developed:

- **Assembler:** CYCLE and LEAVE directives have been added to the macro processor for use with WHILE statements. Performance enhancements are expected to increase the speed of macro-intensive assemblies by 40 to 50 percent. This improvement has necessitated a change in the format of the LOAD and DUMP files. If you have been using this feature, you will need to rebuild all LOAD files. See Part II.
- **Backup:** This versatile new Shell tool is designed primarily to back up project directories rather than whole disks. See Part II.
- **Commando:** This new user interface displays in dialog boxes all the functions, parameters, and options for MPW commands. Help for each option is available in a special window. See Chapter 4 for an introduction to Commando dialogs. You can also use Commando to create a dialog interface for your own MPW tools. See Chapter 14.

- **Compare:** The new option `-v` displays differences between two files in a format that lets you cut and paste output lines into a source file. See Part II.
- **DeRez:** The new `-i` option lets you specify one or more pathnames to search for `#include` files. An `-s` option lets you search for resource includes. See Part II.
- **GetErrorText:** This command displays the explanatory text for error messages. See Part II.
- **GetFileName:** Use this command to quickly generate a standard file dialog box. See Part II.
- **GetListItem:** This tool makes it easy to display a selection list in a dialog. See Part II.
- **Link:** This tool now contains the method table optimizer for object Pascal, available as the `-opt` option, thus eliminating any need for the Optimize tool distributed with MacApp. See Chapter 10 and Part II.
- **MacsBug:** MacsBug performance has been enhanced and upgraded. MacsBug runs on the Macintosh 512K, Macintosh 512K enhanced, and Macintosh Plus, as well as the Macintosh SE and Macintosh II. The MC68881 floating-point coprocessor is supported. See Chapter 11.
- **MakeErrorFile:** Create error message text with this new command. See Part II.
- **PerformReport:** This new utility reports the processor time spent in your source code or the ROM on a per-procedure basis. Included in MPW 2.0 is a set of performance measurement tools for evaluating the execution speed of your programs. See Chapter 12 and Part II.
- **Print:** This Shell command has been modified to run on all Macintosh computers, including the Macintosh SE and Macintosh II, where it uses the new ROMs. Three options have also been added. See Part II.
- **ProcNames:** This new Pascal utility lists a program's procedure and function names, including their nesting level and line numbers. See Part II.
- **ResEqual:** This command compares two files to determine whether they have identical resources. See Part II.
- **Resource Tools:** Rez and DeRez have some new syntax rules, plus new functions and options. See Chapter 8 and Appendix D.
- **Search:** A new set of five options has been added to increase the versatility of this Shell command to search for both matching and non-matching patterns. See Part II.
- **SetPrivileges:** You can now set access privileges to folders shared on an AppleShare™ file server. See Part II.
- **SetVersion:** This is a versatile tool for maintaining version and revision numbers. See Part II.
- **Translate:** This new command lets you change all occurrences of selected characters in a file. It is useful for changing capitalization and for remapping ASCII control characters. See Part II.

Other new features

- **Editor capabilities:** Several new capabilities have been added:
 - Command-arrow key combinations move the cursor to the extremes of the document.
 - Command-Backspace deletes to EOF.
 - Command-D opens the selection.
 - Command-B clears the selection.
 - Command-W closes the window.
 - Command-period cancels.
- **-c option:** Some commands display a dialog box for a user decision when a conflict occurs. The **-c** option is equivalent to answering “cancel” to any dialog box that might otherwise be displayed. In cases where Cancel is clicked in a confirmation dialog box, the status return, in MPW 2.0, is nonzero. This feature is useful in Shell scripts. For example, the **-c** option used in a duplicate command will abort the script if a conflict should somehow occur. You could then investigate the cause of the conflict.
- **Locked files:** Locked files can be duplicated, but cannot be renamed or moved.
- **Printing:** You can now print from the Finder.
- **Rotate Cursor:** Be advised that a change in Rotate Cursor since MPW 1.0 will cause old tools to crash when recompiled.
- **Shell scripts:** The six command scripts listed here are included to support the items in the new menus Directory and Build. You can easily customize these. See “Modifying the Build Menu and Makefiles” in Chapter 9. Scripts are also individually documented in Part II.
 - Build
 - BuildCommands
 - BuildMenu
 - CreateMake
 - DirectoryMenu
 - SetDirectory
- **Standard Shell variables:** The three new standard Shell variables listed here are included in MPW 2.0 to support Commando. See “Variables Defined in the Startup File” in Chapter 5.
 - {Aliases}
 - {Commando}
 - {Windows}
- **Status panel:** Placing the mouse arrow in the Status panel at the lower-left corner of the MPW worksheet changes the panel into a Do It control. Clicking the control is equivalent to pressing the Enter key.

What you'll need

This section describes the hardware and documentation you'll need in order to develop software with the Macintosh Programmer's Workshop 2.0.

Hardware and system requirements

The Macintosh Programmer's Workshop 2.0 can generate applications that will run on any Macintosh, including the Macintosh II, Macintosh SE, Macintosh Plus, Macintosh 128K, Macintosh 512K and 512K enhanced, and Macintosh XL.

However, the MPW 2.0 system requires, at the minimum, a Macintosh Plus with one megabyte of RAM and a hard disk drive, such as the Apple Hard Disk 20. The new Workshop will not run on the Macintosh XL, the Macintosh 512K, or Macintosh 512K enhanced or on systems without hard disks. MPW 2.0 requires the 128K or 256K ROMs; it cannot execute on the older 64K ROMs. A Macintosh II and a SCSI hard disk drive is an ideal developmental system for use with MPW 2.0.

In general, a small RAM cache of about 32K is useful. However, some functions in MPW 2.0 may run slower with a RAM cache. Use of MPW with the Switcher™ is *not* recommended.

MPW software is shipped on 800K disks. Although MPW 2.0 can still read from and write to disks that use non-hierarchical filing system disks, MPW's files must be kept on disks that use the hierarchical filing system. Hard disks, when used as boot disks, must be hierarchical (HFS) volumes.

Apple's Macintosh peripherals, including the LaserWriter® and LaserWriter Plus printers and the AppleShare™ file server, are supported.

System Folder requirements

Please make sure that you are using the latest versions of the System file and the Finder. This software is frequently upgraded by Apple Computer. *MPW 2.0 requires these minimum system file versions:*

- System file 4.1
- Finder 5.5
- Laser Prep 4.0
- ImageWriter® 2.6
- AppleTalk® ImageWriter 3.1
- LaserWriter 4.0

These files are available on version 2.0 or later of the *System Tools* disk, and on the latest version of the *Printer Installation* disk.

Documentation

All programmers will need Volumes I–IV of *Inside Macintosh* (published by Addison-Wesley, 1985), the definitive guide to the Macintosh Operating System and user-interface toolbox. Additional features of the Macintosh SE and Macintosh II computers are documented in Volume V. If you need to understand and control the numeric environment, you'll need the *Apple Numerics Manual*, a guide to the Standard Apple Numeric Environment (SANE™). Finally, you'll need the appropriate documentation for the programming language you'll be using:

- **Assembly language:** *Macintosh Programmer's Workshop Assembler 2.0 Reference*. This reference is included in your Macintosh Programmer's Workshop package. You may also need the appropriate microprocessor documentation from Motorola.
- **Pascal:** *Macintosh Programmer's Workshop Pascal Reference*. This reference is available as part of a separate MPW product.
- **C:** *Macintosh Programmer's Workshop C Reference*. This reference is available as part of a separate MPW product. For a guide to the C language itself, you'll need *The C Programming Language* by B. Kernighan & D. Ritchie, or a similar C manual.
- **MacApp:** *MacApp Programmer's Reference*. This reference is part of a separate product, MacApp, the Expandable Macintosh Application, available from Apple. The MacApp product also requires MPW Pascal.

About this manual

Part I of this book describes the MPW development system, including the Shell and tools. This reference is written for programmers who are already familiar with the Macintosh. It outlines the process of building a program, but does not deal with the particulars of writing it. Language-specific information is covered in the appropriate language references.

If you are new to MPW, see the brief section "Building a Program: An Introduction" in Chapter 2. This introduction will take you through MPW's build process in minutes. Chapter 3 introduces the commands available from the menus and Chapter 4 covers the basics of using MPW, including features of the Commando dialog interface.

Part II of this manual is a complete alphabetical reference to MPW commands. As you become familiar with MPW and no longer need to refer often to the indexed chapters of Part I, you may find it convenient to remove Part II and place it in a smaller binder for handy reference.

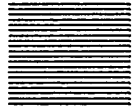
Syntax notation

The following syntax notation is used to describe Macintosh Workshop commands:

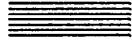
| | |
|--------------------|---|
| terminal | Plain text indicates a word that must appear in the command exactly as shown. Special symbols (-, \$, &, and so on) must also be entered exactly as shown. |
| <i>nonterminal</i> | Items in italics can be replaced by anything that matches their definition. |
| [optional] | Square brackets mean that the enclosed elements are optional. |
| repeated... | An ellipsis (...), when it appears <i>in the text of this reference only</i> , indicates that the preceding item can be repeated one or more times. Do not confuse this reference convention with the ellipsis command-line character (Option-semicolon), used to invoke the Commando dialog interface. |
| a b | A vertical bar indicates an either/or choice. |
| (grouping) | Parentheses indicate grouping (useful with the and ... notation). |

This notation is also used in the output of the Help command. (See “The Help Command” in Chapter 4.)

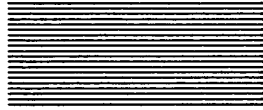
Filenames and command names are not sensitive to case. By convention, they are shown with initial capital letters. Terms printed in **boldface** appear in the glossary.



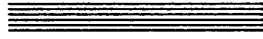
Part I



Shell Reference



Chapter 1



System Overview

The MPW Shell

The **MPW Shell** is an application that provides an integrated, window-based environment for program editing, file manipulation, compiling, linking, and program execution. The other parts of the Macintosh Workshop—the Assembler, compilers, and other tools described below (excepting ResEdit)—operate within the Shell environment. These tools can perform input and output to files and to Shell windows.

The Shell combines a command language, a text editor, and the Commando user interface. You can enter commands in any window, or execute them by using menus and dialogs. A **dialog** may include one or more **dialog boxes**, which may in turn contain text boxes, check boxes, radio buttons, and so on. The command language provides text editing and program execution functions, including parameters to programs, scripting (command file) capabilities, input/output redirection, and structured commands. All of the parameters, functions, and options of the command language are also available in dialog boxes.

The MPW Shell integrates the following functional components:

- **Editor** for creating and modifying text files. The editor implements normal Macintosh-style editing together with scriptable editing commands so that you can program the Shell to perform editing functions. (See Chapters 3, 4, and 6.)
- **Command interpreter** that interprets and executes commands that you enter in a window or read from a file. (See Chapter 5 and Part II.)
- **User interface** that displays dialog boxes providing immediate, graphic access to all of MPW's many functions and options.
- **Built-in commands** that, besides editing commands, include commands for managing files without returning to the Finder, for manipulating windows, processing variables, program control flow, and more. (See Chapter 5.)

Window commands

All work in MPW is done within windows. The following commands are available for manipulating windows:

| | |
|--------------|--|
| Close | Close a window |
| MoveWindow | Move window to h,v location on screen |
| New | Open a new window |
| Open | Open a window |
| SizeWindow | Set a window's dimensions |
| StackWindows | Arrange open windows in a staggered diagonal array |
| Target | Make a window the target window |
| TileWindows | Arrange open windows in a tile pattern |
| Windows | List windows |
| ZoomWindow | Enlarge or reduce a selected window |

File management commands

The MPW Shell provides the following commands for manipulating files and directories without having to exit to the Finder:

| | |
|---------------|---|
| Backup | Back up folder files |
| Catenate | Concatenate files |
| Delete | Delete files and directories |
| Directory | Set the default directory |
| Duplicate | Duplicate files and directories |
| Eject | Eject volumes |
| Equal | Compare files and directories |
| Erase | Initialize volumes |
| Exists | Find out if a file or directory exists |
| Files | List files and directories |
| Mount | Mount volumes |
| Move | Move files and directories |
| Newer | Compare two files to see which was modified most recently |
| NewFolder | Create a directory |
| Print | Print text files |
| Rename | Rename files and directories |
| Save | Save files in all windows |
| SetFile | Set file attributes |
| SetPrivileges | Set access privileges to folders on file server |
| Unmount | Unmount volumes |
| Volumes | List mounted volumes |
| Which | Determine which file (pathname) the Shell will execute |

Editing commands

Besides the Macintosh's usual mice-and-menus editing capabilities, a number of built-in editing commands are provided. You can use these commands both interactively and in command files. Editing commands feature the use of **regular expressions**, a set of special operators that forms a powerful language for defining text patterns. See "Pattern Matching" in Chapter 6 for a discussion of regular expressions.

| | |
|-----------|--|
| Adjust | Adjust lines |
| Align | Align text to left margin |
| Clear | Delete the selection |
| Copy | Copy selection to the Clipboard |
| Cut | Copy selection to the Clipboard and delete the selection |
| Find | Find and select a text pattern |
| Font | Set a window's font characteristics |
| Mark | Mark and name a text selection |
| Markers | List marked selections |
| Paste | Replace selection with contents of the Clipboard |
| Replace | Replace the selection |
| Revert | Revert to saved file |
| Tab | Set a window's tab value |
| Translate | Convert one or more characters |
| Undo | Undo last edit |
| Unmark | Remove a marker from its text selection |

Structured commands

The Shell also provides a number of built-in structured commands. Used mainly in command files, these commands provide conditional execution and looping capabilities:

| | |
|-------------|---|
| Begin...End | Group commands |
| Break | Break from For or Loop |
| Continue | Continue with next iteration of For or Loop |
| Exit | Exit from a script |
| For... | Repeat commands once per parameter |
| If... | Conditional command execution |
| Loop...End | Repeat commands until Break |

Other built-in commands

The MPW Shell also provides a number of other predefined commands:

| | |
|--------------|---|
| AddMenu | Add menu item |
| Alert | Display alert box |
| Alias | Define alternate command names |
| Beep | Generate tones |
| Confirm | Display confirmation dialog box |
| Date | Write the date and time |
| DeleteMenu | Delete a user-defined menu or item |
| Echo | Echo parameters |
| Evaluate | Evaluate an expression |
| Execute | Execute a script without affecting variable scope |
| Export | Make variables available to programs |
| GetErrorText | Display text for system error numbers |
| GetFileName | Display a standard file dialog box |
| GetListItem | Present file selection list in dialog box |
| Help | Display summary information |
| Parameters | Write parameters |
| Quit | Quit MPW |
| Quote | Echo parameters, quoting if needed |
| Request | Request text from a dialog box |
| Set | Define and write Shell variables |
| Shift | Renumber script positional parameters |
| ShutDown | Shut down or reboot machine |
| Unalias | Remove aliases |
| Unexport | Remove variable definition from export list |
| Unset | Remove Shell variables |

MPW command scripts

The items available in the Directory and Build menus use these scripts:

| | |
|---------------|---|
| BuildCommands | Show build commands |
| BuildMenu | Create the Build menu |
| BuildProgram | Build the specified program |
| CreateMake | Create a simple makefile |
| DirectoryMenu | Create the Directory menu |
| Line | Find line number |
| SetDirectory | Set current directory (from Directory menu) |

MPW tools

MPW tools are programs that run within the Shell environment. The tools listed here are included with the Macintosh Programmer's Workshop; several are described in more detail in the sections that follow.

| | |
|---------------|---|
| Asm | MC68xxx Macro Assembler (available as a separate product) |
| C | C Compiler (available as a separate product) |
| Canon | Canonical spelling tool |
| Compare | Compare text files |
| Count | Count lines and characters |
| CvtObj | Convert Lisa [®] object files to MPW object files |
| DeRez | Resource Decompiler |
| DumpCode | Dump code resources |
| DumpObj | Dump object files |
| Entab | Convert runs of spaces to tabs |
| FileDiv | Divide a file into several smaller files |
| Lib | Combine object files into a library file |
| Link | Link an application, tool, or resource |
| Make | Program maintenance tool |
| MDSCvt | Convert MDS Assembler source (part of the MPW Assembler product) |
| Pascal | Pascal Compiler (available as a separate product) |
| PasMat | Pascal program formatter (part of the MPW Pascal product) |
| PasRef | Pascal cross-referencer (part of the MPW Pascal product) |
| PerformReport | Generate a report analyzing program performance |
| ProcNames | Display Pascal procedure and functions names |
| ResEqual | Compare files on a resource-by-resource basis |
| Rez | Resource Compiler |
| RezDet | Detect inconsistencies in resources |
| Search | Search files for a pattern |
| SetVersion | Maintain version and revision numbers |
| TLACvt | Convert Lisa TLA Assembler source (part of the MPW Assembler product) |

Assembler

The Assembler translates MC68000, MC68010, MC68020, and MC68030 assembly-language programs into object code. MC68881 floating-point instructions and MC68851 memory-management instructions are also supported. The Assembler provides powerful macro facilities, code and data modules and entry points, local labels, and (optional) optimized instruction selection. Assembly-language interfaces are provided to the "Inside Macintosh" libraries. Other libraries and example files are also provided.

The Assembler is provided as a separate product, MPW Assembler, which includes the following:

- translation of MC68000, MC68010, MC68020, and MC68030 assembly-language programs into object code
- support for MC6881 floating-point instructions and MC68851 memory management instructions
- powerful macro facilities, code and data modules, and entry points, local labels, and (optional) optimized instruction selection
- assembly-language interfaces to "Inside Macintosh" routines
- conversion tools
 - TLACvt to convert Lisa Workshop Assembler (TLA) source files to MPW Assembler source files
 - MDSCvt to convert Macintosh 68000 Development System (MDS) Assembler source files to MPW Assembler source files
- sample programs

Pascal tools

The Pascal system is provided as a separate product, MPW Pascal, which includes the following:

- Pascal Compiler
- Pascal cross reference program (PasRef)
- Pascal source file format program (PasMat)
- Pascal procedure and name program (ProcNames)
- Pascal runtime library
- Pascal interfaces to the "Inside Macintosh" routines
- sample programs

Macintosh Workshop Pascal 2.0 is an improved version of MPW 1.0 Pascal. The improvements include: access to C functions and global data, arbitrary-length identifiers, object Pascal extensions, support for dynamic array types greater than 32K, generation of MC68881 code, optional long word data and stack alignment, and peephole optimization.

C Compiler

The C Compiler is provided as part of a separate product, MPW C, which includes the following:

- C Compiler
- Standard C Library
- C interfaces to the "Inside Macintosh" libraries
- sample programs

The C Compiler implements the full C language as defined in *The C Programming Language*, by Brian Kernighan and Dennis Ritchie. The usual extensions to this definition provide enumerated types and structure assignment, parameters, and function results. In addition, Apple extensions provide SANE numerics and interfaces to Pascal functions and Macintosh traps. Most Standard C Library functions, including character and string processing, memory allocation, and formatted input/output, are also provided. Macintosh Workshop C 2.0 supports generation of MC68881 code, as well as optional long word data and stack alignment.

Linker

The Linker combines object code files into executable programs or driver resources. The Linker optionally includes only the code and data modules that are referenced. The Linker replaces the code segments in an existing resource file, without disturbing other resources in the file. An option directs the Linker to produce a link map as a text file. Other options allow the creation of an object module cross reference file or a file containing a list of all the unreferenced modules.

A separate tool, Lib, provides library manipulation. Linking is performed automatically for simple programs constructed by using the Build menu. See Chapter 10 for details on the use of the Linker with more elaborate programs.

Make

The Make tool simplifies software construction and maintenance. Its input is a list of dependencies between files, and instructions for building each of the files. Make generates commands to build specified target files, rebuilding only those components that are out-of-date with respect to their dependencies. Makefiles are generated automatically when you build simple programs from the Build menu. To use Make with more elaborate programs, see Chapter 10.

Resource Compiler and Decompiler

The Resource Compiler (Rez) reads a textual description of a resource and converts it into a standard Macintosh resource file. The Resource Decompiler (DeRez) converts resources into a textual representation that can be edited in the Shell, and recompiled with Rez. You can use DeRez to create Resource Compiler input from any existing resource files. Rez and DeRez need templates (type declarations) to define resource types. Definitions of the standard Macintosh resource types ('MENU', 'STR#', 'ICON', and so on) are provided in two commented text files, Types.r and SysTypes.r. Another tool, RezDet, checks resource files for consistency. These resource tools are documented in Chapter 8.

Rez has a new feature in MPW 2.0—an option that permits you to add resources without disturbing other resources in the file.

Commando

The Commando tool implements the Commando dialog user interface for all other MPW tools and commands. Obviously, this is a great convenience for dealing with tools offering many interdependent options. Newcomers to MPW will appreciate Commando's instant assistance in building complex command lines. You can also use Commando to create specialized dialogs for your own MPW tools and scripts.

Commando looks in a tool's or script's resource fork for a resource of the type cmdo. Commando then loads the resource, builds a dialog, handles events, and passes the resulting command line back to the Shell for execution. Commando relies on three Shell variables to do its work: {Commando}, {Aliases}, and {Windows}. The basics of using Commando dialogs are described in Chapter 4. Dialogs utilizing specialized types of dialog boxes are presented with the tools they support in Part II. Chapter 14 tells you how to create a Commando interface for your own tools and scripts.

Conversion tools

Canon is a tool for regularizing the spelling and capitalization of identifiers in source files moved from other systems. (In the Macintosh Workshop languages, all characters are significant rather than just the first eight as in the Lisa Workshop. In C, case also matters.)

The file Canon.dict contains the correct spelling and capitalization for "Inside Macintosh" ROM routines. C programmers, in particular, will find Canon and Canon.dict useful.

Entab is a useful tool for converting space characters and tabs to conform to MPW editor or other editor conventions.

You can look up these conversion tools in Part II.

Performance measurement tools

The performance measurement tools enable you to pinpoint where your code is spending time. These libraries allow you to sample the program counter, produce a file of output data, and analyze that data with a report generator. Advanced programmers will find these tools useful for streamlining the execution of their code. Chapter 12 is devoted to this subject.

Applications

Applications are stand-alone programs that can execute outside the Shell environment. A single application, ResEdit, is provided with MPW. It is assumed that you already have the Font/DA Mover, which is distributed on the *System Tools* and *System Installation* disks. Any application can be executed from the MPW Shell.

ResEdit

ResEdit is an interactive, graphically-based resource editor for creating, editing, and copying resources. An interface like that in the MacDraw[®] application is provided to help you design your own fonts. MPW Pascal includes a set of extended Resource Manager routines that make it possible to write your own add-on resource editors for ResEdit. See Chapter 7 for a thorough explanation of ResEdit.

Debugger

An improved MacsBug debugger is provided with MPW 2.0, fully supporting the MC68000 and MC68020 processors, as well as the MC68881 floating-point coprocessor. MacsBug resides in RAM together with your program. MacsBug allows you to examine memory, trace through a program, or set up break conditions and execute a program until they occur. MacsBug runs on all Macintosh computers, including the Macintosh SE and Macintosh II. See Chapter 11 for instructions on using MacsBug and Appendix F for the object file format.

Special command scripts

Several special command scripts are provided. These text files contain commands that are read by the MPW Shell at startup and shutdown:

- The Startup file is a command script containing another script, UserStartup, that is run each time you start the MPW Shell. You can use UserStartup to customize MPW. The Startup file is discussed in detail in Chapter 5.
- The Suspend and Resume files are scripts that preserve the state of the Shell environment while a stand-alone application is executing. The Quit file saves the state of the Shell environment when you exit to the Finder.

Examples

In addition to the examples excerpted in this reference work, you'll find numerous complete examples in folders included on the MPW distribution disks.

Sample program files

Source files are provided for the sample application from *Inside Macintosh*, as well as for a sample MPW tool and a sample desk accessory. Assembly-language versions of these programs are contained in the folder AExamples. MPW Pascal and MPW C also include Pascal and C versions of the sample files, in the folders PExamples and CExamples. The Examples folders also contain instruction files and makefiles for building the sample programs. Some of these examples are referred to in Chapter 2, in the introduction to building a program.

Note that these are part of the respective C, Pascal, and Assembler products.

Command-language examples

Examples of the use of the MPW command language are provided in the folder Examples. Among these are

- Addmenu commands for creating user-defined menu items
- A list of UNIX-oriented aliases
- Suggestions for modifying the Startup script

To learn more about these examples, open the file Instructions in the Examples folder. Additional examples are included with each of the MPW commands in Part II. The command language is documented in Chapter 5.

Overview of MPW files and directories

Appendix A contains a complete list of all of the Macintosh Workshop 2.0 files. It also describes the recommended setup of files on a hard disk. Figure 1-1 shows the initial setup of your MPW folders and files on a hard disk. (The Pascal and C systems are included.)

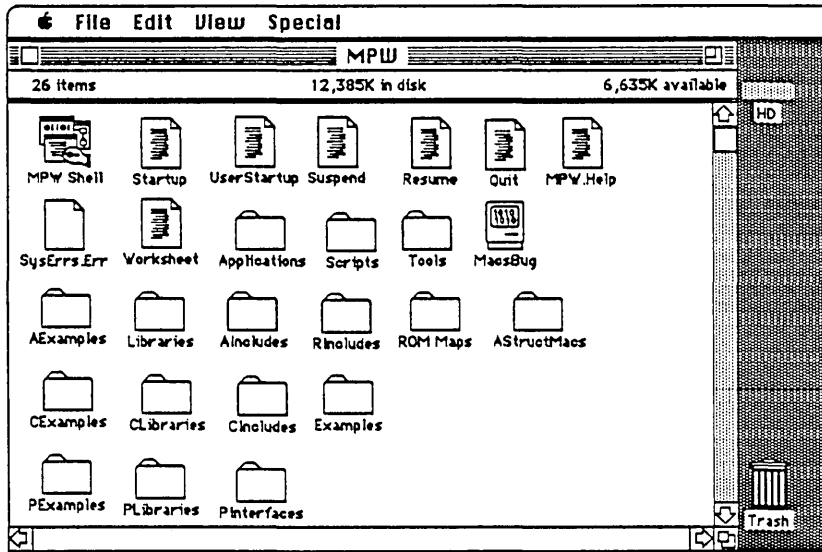
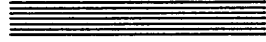


Figure 1-1
Setup of MPW folders and files

For important information about setting up your MPW system, see "Installing the System" in Chapter 2.



Chapter 2



Getting Started

This chapter explains how to start using MPW. If you are new to MPW, the tutorial “Building a Program: An Introduction,” later in this chapter, will introduce you to the extraordinary power and ease of using this environment.

Installing the system

Macintosh Programmer’s Workshop 2.0 is shipped on five 800K disks: MPW1, MPW2, MPW3, MPW Assembler1, and MPW Assembler2. The Pascal and C systems occupy another disk each. This section describes how to install the Macintosh Workshop files onto any hard disk configuration. MPW 2.0 cannot be installed in its entirety on an 800K disk system.

Appendix A, “Macintosh Workshop Files,” shows the recommended arrangement of files on a hard disk. HFS pathname rules are explained in this chapter.

- ❖ *Note:* A command script named Startup is executed by the MPW Shell during initialization. This file defines several Shell variables, including the variables that indicate the location of MPW tools, applications, include files, and libraries. Startup must be in the same folder as the MPW Shell.

Use the Finder to follow these steps:

1. Create a folder named MPW on the hard disk.
2. Copy the contents of all five disks to folder MPW. If you have MPW Pascal or C, also copy those disks into the MPW folder.
3. Move the entire contents of the folder More Tools to the Tools folder; then throw away the (now empty) More Tools folder.
4. If you have MPW Assembler, move the files Asm, MDSCvt, MDSCvt.Directives, TLACvt, and TLACvt.Directives to the Tools folder. If you have Pascal or C, move the Pascal and C Compilers (named Pascal and C) into the Tools folder. MPW Pascal includes the additional Pascal tools PasMat and PasRef—these should also be moved into the Tools folder.
5. *Make sure that the following files are in the same folder as the MPW Shell:*
 - Startup
 - UserStartup
 - Suspend
 - Resume
 - Quit
 - MPW.Help
 - SysErrs.Err

By default, the Macintosh Programmer’s Workshop assumes installation in the MPW folder as described above. Other configurations are possible but would require some changes to the pathnames defined in the Startup file so that the Shell and tools could find various files.

Starting up

- ❖ *Note:* A small RAM cache (perhaps 32K) is useful when running MPW 2.0. You may use larger caches on the Macintosh Plus, Macintosh SE, or Macintosh II, with the Assembler and Pascal. MPW C may be used with larger RAM caches on systems with more than 1 megabyte of memory. However, some functions in MPW 2.0 may run more slowly with large RAM caches. Use of MPW with the Switcher is not recommended.

From the Finder, select and open the MPW Shell icon. The **Worksheet window** (shown in Figure 2-1) will appear with its full pathname in the title bar (for example, "HD:MPW:Worksheet"). This window has no close box, and is always present on the screen; otherwise it's just like any other window.

You can also start the Workshop by double-clicking on any Macintosh Workshop text document or tool.

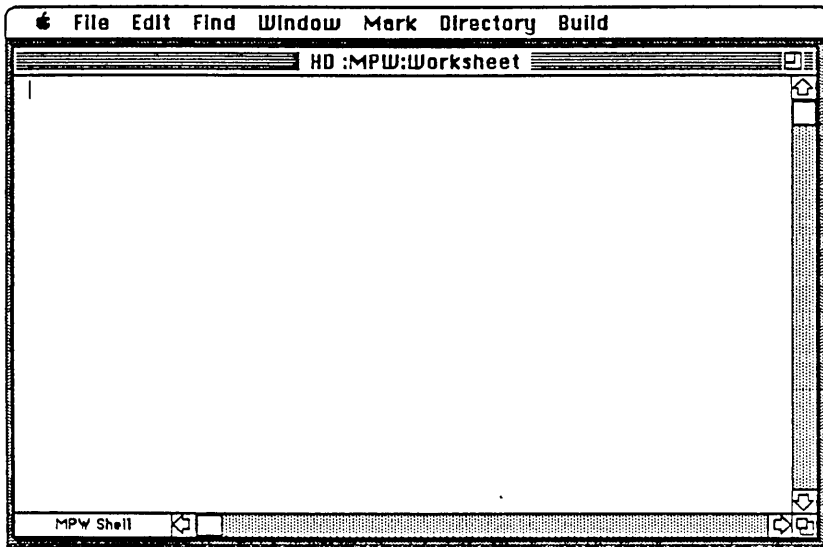


Figure 2-1
Worksheet window

The menus available from the Shell appear in the menu bar at the top of the screen. An explanation of each menu is provided in Chapter 3. You can easily add your own menu names. (See Chapter 9.)

A **status panel** at the window's lower-left corner shows the name of the command that's currently executing, or simply "MPW Shell" when you're not executing a command. A mouse click on the status panel is equivalent to pressing the Enter key.

When you first start the Macintosh Programmer's Workshop, a script called Startup executes. The Startup file defines several variables and command aliases (alternate command names); this file is further described in Chapter 5.

Important

The Startup file must be in the same directory as the MPW Shell.

Selecting commands from menus

In MPW, commands may be built-in commands, scripts (command files), tools, or applications, as explained in Chapter 1.

Several of the built-in commands can be executed by using the File, Edit, Mark, and Window menus. The Directory and Build menus are optional, and are normally installed by UserStartup scripts. The items in these menus execute scripts (see Chapter 3 for details about menus).

You can add your own menu items to the File, Edit, Find, Directory, and Build menus. By using the AddMenu command you can even add your own menus. Each user-defined menu item specifies a list of MPW commands that are executed when the menu item is selected. See the file AddMenu in the Examples folder for a number of ideas for user-defined menus.



Figure 2-2
MPW menu bar

Building a program: an introduction

This section shows how easy it really is to use MPW 2.0 by taking you step by step through the process of building an example program. You'll find that the Build menu and the Commando dialog boxes make the learning process intuitive and comfortable. Even if you've never used MPW before, you can immediately use the Build menus to build programs.

MPW's automated Build menu lets you assemble, compile, and link simple programs without studying the command language, the numerous Compiler and Linker options, or countless other details. You can use the Build menu to build applications, tools, and desk accessories written in assembly language, C, Pascal, and Rez or a combination of these languages. You may include resource specifications when building programs with these menus.

The example programs

In this introduction, three assembly-language programs included with MPW are suggested as examples:

- **Sample:** the "Inside Macintosh" sample application
- **Count:** an MPW tool that counts characters and lines in a file (see Part II)
- **Memory:** a sample desk accessory that displays the memory available in the application and system heaps, and on the boot disk

Similar program examples are included with MPW C and MPW Pascal. If you are primarily interested in programming in one of these languages, be sure to read, in the corresponding language reference, the section on the example programs.

You can routinely rebuild more complex programs by selecting a single menu item. There is a smooth transition from the simple builds to the more complex ones. (See Chapter 9 for information on how to modify the Build menu and the makefile that it creates.)

The source files for each of these three assembly-language examples are in the "AExamples" folder. For example, the source for Count consists of the files Count.a and Stubs.a. A makefile that contains the commands for building all of the examples is also provided in the same folder. Instruction files are also provided on the MPW disks for each language. If you are new to MPW, we recommend that you start with the tutorial that follows rather than with the Instructions file on the disks. At the conclusion of this tutorial you will be referred back to the disk instructions.

Two easy steps

You can build each of the example programs in two steps, using the Directory and Build menus:

1. Set the current directory.
2. Build the program.

Both of these steps are explained below. You can use this section to take MPW on a test drive.

1. Set the current directory.

Open the Directory menu. The upper half of the menu contains the commands to show the current directory and to change it to an arbitrary directory. The lower half of the menu lists frequently used directories.

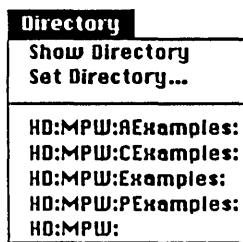


Figure 2-3
Directory menu

Select Show Directory to find out what your current directory is. You'll see the alert shown in Figure 2-4.

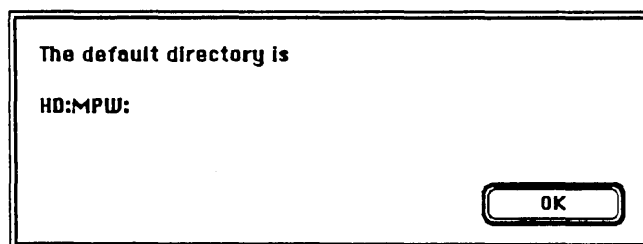


Figure 2-4
Show Directory alert

Click OK to remove the alert. You're going to build the assembly-language program Sample, so you'll need to set the current directory to the directory that contains the assembly-language examples. Now open the Directory menu again and select the menu item that ends in "AExamples." (See Figure 2-3.) Selecting "AExamples" from the Directory menu runs commands that set the current directory. You can check to see if the current directory has been correctly reset by selecting the Show Directory menu item again. (The Set Directory... menu item is used to add other directories to the list at the bottom of the Directory menu. This menu item is explained in "Building a New Program" later in this chapter.)

2. Build the program.

Now open the Build menu and select any one of the four Build menu items.

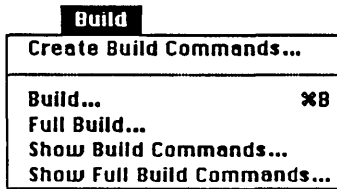


Figure 2-5
Build menu

Each Build item builds your specified program in a slightly different way:

| | |
|-----------------------------|--|
| Build... | The program is built automatically, but only files that have been modified since you last built the program will be processed. Use this item to save time. The Command-key equivalent is Command-P. |
| Full Build... | The program is completely built, ignoring any object files or intermediate files that may exist from a previous build. |
| Show Build Commands... | The commands needed to build the program (using just those files affected by modifications since the last build) are displayed on the worksheet, but not executed. You can then select any or all of the commands—or modify them—and then press Enter to execute them. |
| Show Full Build Commands... | All the commands needed to completely rebuild the program (whether modified since the last build or not) are displayed on the worksheet, but not executed. This is a convenient way to see all of the commands used in building the program you've selected. |

See “Build Menu” in Chapter 3 for more information on Build menu items. When selected, each Build item first displays a dialog box like that in Figure 2-6, requesting the name of your program.

For this tutorial, select Full Build.

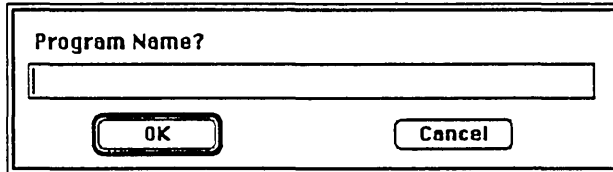


Figure 2-6
Program Name? dialog box

When the Program Name? dialog box appears, type the name of the program you want to build (in this case, type “Sample”) and then click the OK button. (Be sure that you type the name Sample and *not* Sample.a. Since you have already set the directory to AExamples, you don’t need to indicate that you want to build the assembly-language version of Sample. If you give Sample.a as the program name, the Build script will attempt to build the source file.)

The Worksheet window now becomes the frontmost window. The status panel in the lower-left corner flashes the name of each operation as it is performed by MPW. Each of the MPW commands used by the Full Build script appears on the worksheet as it is executed. When the build has finished, your worksheet should look like Figure 2-7.

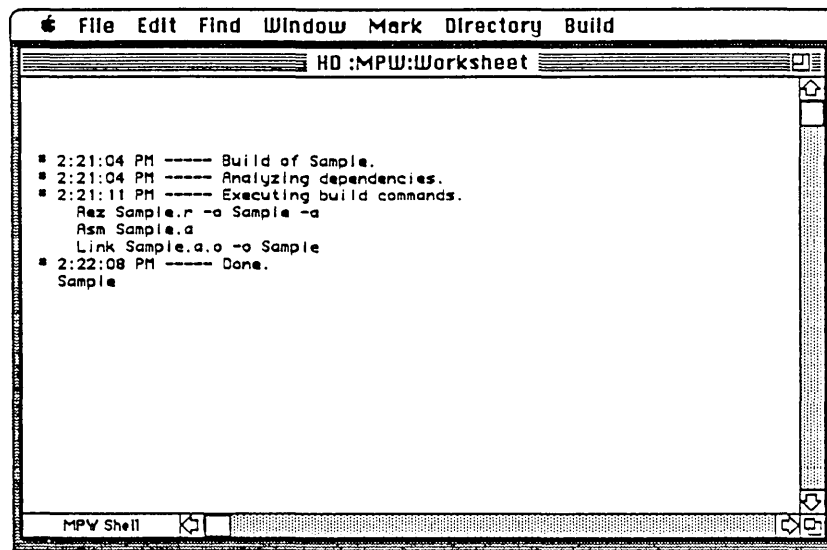


Figure 2-7
Finished Sample build

To check your work, press Enter. The Shell then executes the newly built program, displaying the text-edit window that Sample creates, as described at the beginning of *Inside Macintosh*. When you quit the Sample program, you are returned to the Shell.

Use the same procedure to build the two other examples in the AExamples folder: the tool Count and the desk accessory Memory. For guidance in using these examples, consult the file Instructions in the folder AExamples.

In general, to run a newly built program, select its name (and, in the case of a tool, any parameters), and press Enter. If the program you have built is an application, your open windows, user-defined menus, and other status information will be saved before the program is run. When you quit the application you are returned directly to MPW with your previously open windows and menus still displayed. If the program is an MPW tool it is run without leaving MPW (be sure to specify any required parameters and options).

When you build a desk accessory by using Build or Full Build, the last line of the Build transcript is a command that will run the Font/DA Mover to install the desk accessory in the System file. After this installation is complete, the desk accessory will appear in the Apple menu. If your Font/DA Mover isn't in the Applications folder or in another directory specified by the {Commands} shell variable, then you should use either the Finder or the MPW Duplicate command to move it there.

If you're curious about the functioning of any of the Build commands, see Chapter 9 for more background on the Build process.

Building a new program

The Directory and Build menus are convenient to use when writing programs of your own. You use slightly different steps for creating a program of your own:

1. Set the current directory by using the Directory menu.
2. Type your program.
3. Select Create Build Commands... from the Build menu.
4. Select a build item from the Build menu.

Each of these steps is explained below.

1. Set the directory.

The first step in creating a new program is to set the directory where you want your new program to reside. You can select one of the directories that appears in the Directory menu, or you can select another directory by using the Set Directory menu item. When you select Set Directory from the Directory menu, a standard file dialog box, like that in Figure 2-8, appears.

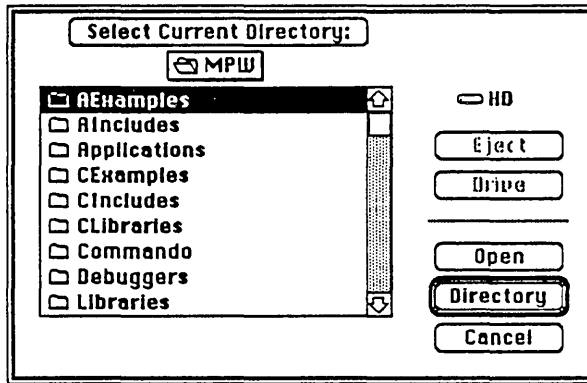


Figure 2-8
Set Directory standard file dialog box

Select the directory you need. After highlighting the directory you want, click "Directory" or "Select Current Directory:" at the top of the dialog box. The new directory will then be added to the list of directories on the Directory menu.

2. Type your program.

The next step is to create the source files for your program. Select New in the File menu. (Remember that assembly-language source filenames should end with ".a", C filenames with ".c", Pascal filenames with ".p", and Rez filenames with ".r".) An empty window now appears and you are ready to type your program. Enjoy!

3. Select Create Build Commands... from the Build menu.

When you've finished typing in your program, select Create Build Commands from the Build menu. You'll see the dialog box shown in Figure 2-9.

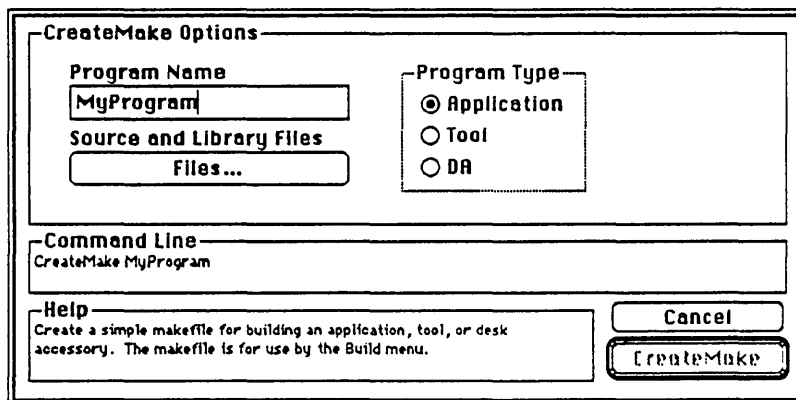


Figure 2-9
CreateMake dialog box

Type in the program's name (without ".a", ".c", or ".p" suffixes) and click a radio button to indicate whether you want to create an application, tool, or desk accessory. When you click the Files button, another dialog box appears, permitting you to select the needed source and library (ending with the ".o" suffix) files. Your program will be linked with these files.

- ❖ *Note:* It isn't necessary to indicate the standard library files supplied with MPW. Your program will be automatically linked with the appropriate libraries. The reference for CreateMake in Part II explains which standard library files will be used.

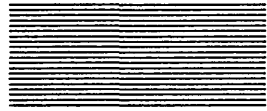
The Create Build Commands item in the Build menu runs a script that creates a makefile with the necessary commands for building programs written in assembly-language, C, Pascal, Rez, or a combination of languages. This file is given your program's name with the suffix ".make".

- ❖ *Note:* The Build script uses Make to determine the minimum operations necessary to bring the program up to date. The Build script looks for its build instructions first in *program.make* (for example, *Sample.make*). If no such file is found, the Build script looks for its instructions in *MakeFile*.

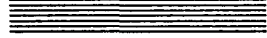
4. Select a build item from the Build menu.

The four build items on the Build menus are variations on a theme. (See Chapter 3 for an explanation of each item. A brief explanation appears earlier in this chapter under step 2 of "Two Easy Steps.") For now, select Full Build. The rotating beach ball cursor appears, indicating that processing has begun. Each step of the build process is displayed on the worksheet as it occurs. Any errors will be displayed also, making it easy to track down a bit of misplaced syntax. When you have fixed the problem, just select Build from the Build menu to quickly rebuild the program. The record of previous builds is left on the worksheet.

See Part II for detailed information on each of the Build menu items.



Chapter 3



Using the Shell Menus

This chapter describes MPW's menus and associated dialog boxes. You can build simple programs by using the Directory, File, and Build menus. (See Chapter 2.) The other menus are used for general editing. Advanced editing capabilities are discussed in Chapter 6.

Features

The MPW Shell provides the following editing features:

- Both menu and command-language editing. The menu commands provide the usual Macintosh interface.
- Selecting text by program syntax. You can double-click any of the characters () [] { } " ' ` to select everything between the character and its mate. (To select text between quotes, click the *left* quote.)
- Selection of large sections of text by embedding markers. Marked selections are scriptable; your command files can refer to one or more marked selections. The marker commands, Mark and Unmark, are available from the Mark menu. Basic interactive use of markers is covered later in this chapter. See Chapter 6 for more detailed information on scripting marked selections.
- Complete integration of editing functions with the command interpreter. In the MPW Shell, there is no separation of "command" and "editor" modes. To the Shell, text is text—it is only when you try to directly execute a string of text that the Shell decides whether it is a legitimate command or not.
- Scriptable commands. Because editing commands are part of the command language, you can use them with structured commands and variables to put together scripts that define new editing commands. (See Chapter 6.)
- Regular expressions for matching text patterns. These make possible powerful search and replace functions that eliminate the need to make repetitive changes by hand. (See Chapter 6.)

File format

Shell text is saved as a text-only (TEXT) file. The file contains tab and return characters, but no other formatting information. This format is compatible with other applications that create text-only files—for example, the Shell can process MacWrite® files saved with the Text Only option. When you select the Open command, the Shell displays all text-only files in its standard file dialog box, regardless of the file creator.

- ❖ *Note:* From the Finder, you can open a text file created by another application by selecting both the MPW Shell and the text file icons, and then choosing the Open command.

You can display the invisible characters (spaces, tabs, returns, and all other "control" characters) with the Show Invisibles menu item.

A file's tab setting, font setting, selection, window settings, auto-indent state, invisibles state, and markers are saved with the file, in its resource fork.

Menu commands

In general, the menu interface is the familiar Macintosh implementation. There are a few differences and extensions, which are detailed in the following sections. (It's assumed that you are already familiar with standard Macintosh editing techniques.)

All menu commands act on the active (that is, the frontmost) window.

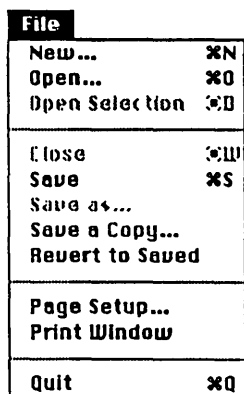
- ❖ *Note:* Many menu items (including several items in the File menu and all user-defined items) are disabled when you are running tools, scripts, applications and other commands. This convention prevents you from closing windows that the command may be reading, and from trying to run another command at the same time.

Apple menu

About the MPW Shell Displays version and copyright information.

File menu

Each of the items in the MPW File menu is described below.



| File | |
|-----------------|-----|
| New... | ⌘N |
| Open... | ⌘O |
| Open Selection | ⇧⌘O |
| Close | ⇧⌘W |
| Save | ⌘S |
| Save as... | |
| Save a Copy... | |
| Revert to Saved | |
| Page Setup... | |
| Print Window | |
| Quit | ⌘Q |

Figure 3-1
File menu

New... Displays the New dialog box, shown in Figure 3-2. The MPW New dialog box allows you to enter a name and select a directory location for the document. The Command-key equivalent is Command-N.

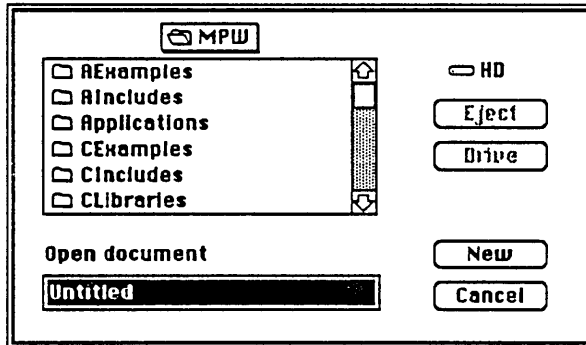


Figure 3-2
New dialog box

Open... Displays an Open dialog box (similar to that in Figure 3-2) that allows you to open any TEXT file on the disk. When you open a file for the first time, the selection point is at the top of the file. When you open the file again, it reappears in the same state in which it was saved; that is, the previous selection or insertion point is preserved unless the file has been modified outside the editor. The Command-key equivalent is Command-O.

Note: If you try to open a document that's already open in another window, that window will be brought to the front.

Open Selection If you select a document name within a window, the Open Selection command automatically displays the selected name. This is a useful shortcut when you have already displayed filenames on the screen, with the Files command for example—you can then select a filename and open a file directly, bypassing the usual Open dialog box. Variable and command substitution occur on the selection. The Command-key equivalent is Command-D.

Close Closes the active (frontmost) window. The Command-key equivalent is Command-W.

Save Saves the active window under its current name, without closing it. This menu item is dimmed if the contents of the window haven't been modified since it was last saved. The Command-key equivalent is Command-S.

Save as... Displays a Save As dialog box, allowing you to change the name and directory location of the active window. Saves the current contents of the window as the "Save As" file, and allows you to continue editing the new file. The old file is closed without saving, under its original name.

- Save a Copy... Saves the current state of the active window to a new file on the disk. You can then continue editing the *old* file.
- Revert to Saved Throws away any changes you have made since you last saved the active window.
- Page Setup... Displays the standard Page Setup dialog box.
- Print Window Prints either the entire active window or the selection in the active window. If any text is selected in the active window, that text is printed. If no text is selected, the entire contents of the window (that is, the entire file) are printed.
- The Print menu item doesn't display the usual Print dialog box. Instead, you can specify printing parameters by setting the Shell variable {PrintOptions}, described in Chapter 5. Printing options include the number of copies to print, which pages to print, print quality, font and font size, headings and title, borders, and printing the pages in reverse order (for use with the LaserWriter). See the description of the Print command in Part II for a complete specification of these options, or enter the command Help Print to see a summary.
- Technical note:* The Print Window menu item executes the Shell command
- ```
Print {PrintOptions} "{Active}" d
 >> "{Worksheet}"
```
- Print Selection executes the same command, with `.$` added after the name of the active window.
- Important:* For the Print command to work properly, you must install the printer drivers available on version 1.0 or later of the *Printer Installation* disk. Use the Chooser Desk Accessory from the Apple menu to specify which printer to use. Use the Page Setup dialog box to specify paper size, orientation, and reductions or enlargements.
- Quit Returns to the Finder, first allowing you to save the current state of all open files. The Command-key equivalent is Command-Q.

---

## Edit menu

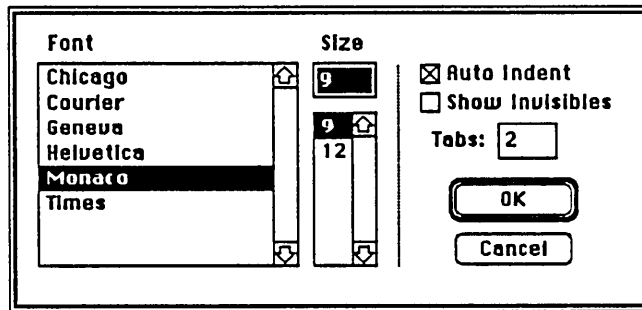
See “Editing With the Command Language” in Chapter 5 for more information on using the scriptable forms of the commands on this menu.

| Edit           |    |
|----------------|----|
| Undo           | ⌘Z |
| Cut            | ⌘K |
| Copy           | ⌘C |
| Paste          | ⌘U |
| Clear          |    |
| Select All     | ⌘A |
| Show Clipboard |    |
| Format...      | ⌘Y |
| Align          |    |
| Shift Left     | ⌘[ |
| Shift Right    | ⌘] |

**Figure 3-3**  
Edit menu

|                |                                                                                                                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Undo           | Undoes the most recent changes to <i>text</i> in the active window (but <i>not</i> changes to resources such as font or tab settings). You can select Undo again to redo changes. The Command-key equivalent is Command-Z. |
| Cut            | Copies the current selection in the active window to the Clipboard, and then deletes it from its original location. The Command-key equivalent is Command-X.                                                               |
| Copy           | Copies the current selection in the active window to the Clipboard. The Command-key equivalent is Command-C.                                                                                                               |
| Paste          | Replaces the contents of the current selection in the active window with the contents of the Clipboard. The Command-key equivalent is Command-V.                                                                           |
| Clear          | Deletes the current selection in the active window.                                                                                                                                                                        |
| Select All     | Selects the entire contents of the active window. The Command-key equivalent is Command-A.                                                                                                                                 |
| Show Clipboard | Opens a window displaying the contents of the Clipboard, if any.                                                                                                                                                           |

Format... Displays the Format dialog box offering a selection of fonts and sizes. The Command-key equivalent is Command-Y. This dialog box is shown in Figure 3-4.



**Figure 3-4**  
Dialog box of the Format menu item

- Tabs** Sets the number of spaces that a tab character will signify for the active window. (The default tab setting is set by the Shell variable {Tab}, described in Chapter 5.)
- Auto Indent** Toggles Auto Indent on and off. When Auto Indent is on, pressing Return lines up text with the previous line. (A check mark indicates that Auto Indent is on.)
- Show Invisibles** Displays the invisible characters as follows:
- |                              |   |
|------------------------------|---|
| Tab                          | Δ |
| Space                        | ◇ |
| Return                       | ↵ |
| All other control characters | ; |

The rest of the dialog box consists of a selection of the fonts installed in your System file. Available font sizes are displayed in the dialog window.

❖ *Note:* Selecting a font and font size affects the *entire* active window, not just the current selection in that window.

- Align** Aligns the currently selected text with the top line of the selection.
- Shift Left, Shift Right** These commands move the selected text left or right by one tab stop. You can thus move a block of text while maintaining indentation. Shift Left adds a tab at the beginning of each line. The Command-key equivalent is Command-[. Shift Right removes a tab, or the equivalent number of spaces, from the beginning of each line. The Command-key equivalent is Command-]. If you hold down the Shift key while using these menu items, the selection will be shifted by one space, rather than by one tab.

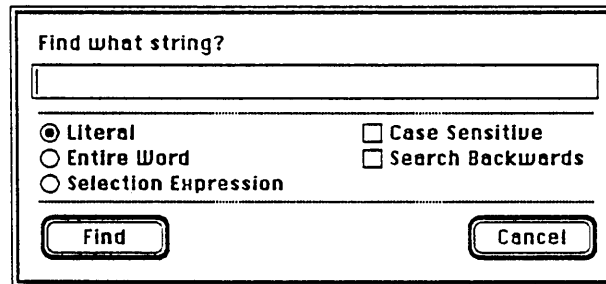
## Find menu

Each of the items in the Find menu is described below.

| Find              |    |
|-------------------|----|
| Find...           | ⌘F |
| Find Same         | ⌘G |
| Find Selection    | ⌘H |
| Display Selection |    |
|                   |    |
| Replace...        | ⌘R |
| Replace Same      | ⌘T |

**Figure 3-5**  
Find menu

- Find...** Displays a Find dialog box and finds the string you specify. By default, the Editor searches forward from the current selection in the active window (and does not wrap around). The Command-key equivalent is Command-F. This dialog box is very similar to the Find-and-Replace dialog box described below; the explanation of the radio controls and check boxes applies to both dialog boxes.
- Find Same** Repeats the last Find operation, on the active window. The Command-key equivalent is Command-G.
- Find Selection** Finds the next occurrence of the current selection in the active window. The Command-key equivalent is Command-H.
- Display Selection** Scrolls the current selection in the active window into view.
- Replace...** Displays the Find-and-Replace dialog box shown in Figure 3-6 and explained below. The Command-key equivalent is Command-R.
- Replace Same** Repeats the last Replace operation. The Command-key equivalent is Command-T.



**Figure 3-6**  
Dialog box of the Replace menu item

The operation of this dialog box is very similar to that of the Find dialog box, except that selected strings can be located and replaced with a different string throughout a file. Both dialog boxes have three radio buttons offering you one of three mutually exclusive options:

- Literal Finds the exact string (without regard for case) that you specify, wherever it may appear, even if part of other words or expressions.
- Entire Word Finds the specified string only when it occurs as a single word. To the Editor, a word is composed of the characters a–z, A–Z, 0–9, and the underscore character ( \_ ). (You can change these default values by redefining the Shell variable {WordSet}—see “Predefined Variables” in Chapter 5.)
- Selection Expression Enables the full selection and regular expression syntax, as used with the command language and described in Chapter 6. These expressions allow powerful selection and pattern matching capabilities that use a special set of metacharacters introduced below.

Any combination of the three check boxes may be selected:

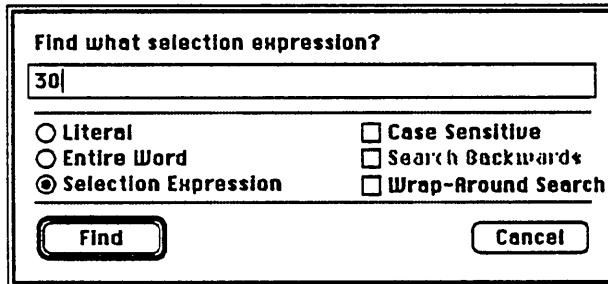
- Case Sensitive Searching is normally case insensitive; selecting this menu item specifies case-sensitive searching. (It does this by setting the Shell variable {CaseSensitive}—see “Variables Defined in the Startup File” in Chapter 5.)
- Search Backward Search backward, from the current selection to the beginning of the file. (Normally, searching is forward, and stops at the end of the file.)
- Wrap-Around Search Searches forward to the end of file, then wraps around and searches from the beginning of the file to the location of the cursor when the search was initiated. (Direction of search is reversed if Search Backward is also checked.)

❖ *Note:* For Find and Find-and-Replace operations, a beep indicates that the selection was not found.

## Selection expressions

When the Find-and-Replace dialog's "Selection Expression" switch is selected, you can use a special set of expression operators to specify selections and text patterns. This section introduces a commonly used subset of these selection operators. Many more capabilities are available, and a full discussion can be found in Chapter 6.

**Selection by line number:** A number given by itself specifies a line number. In the figure below, for example, the command selects line number 30 in the active window.

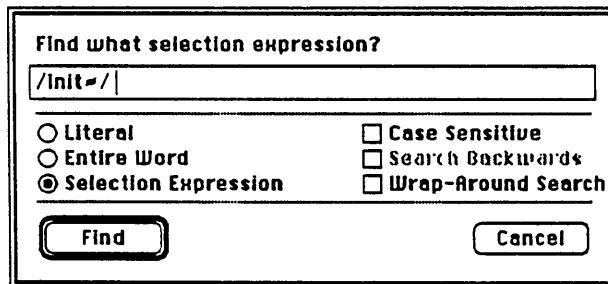


**Figure 3-7**  
Selection by line number

**Wildcard operators:** The same wildcard operators used in filename generation can also be used to specify text patterns for Find commands:

- ? Any single character (other than Return).
- = Any string of 0 or more characters, not containing a Return. (To get the = character, press Option-X.)
- [*characterList*] Any character in the list.  
*Note:* The brackets must be typed; they don't indicate an optional syntax element.
- [~*characterList*] Any character *not* in the list. (To get the ~ character, press Option-L.)

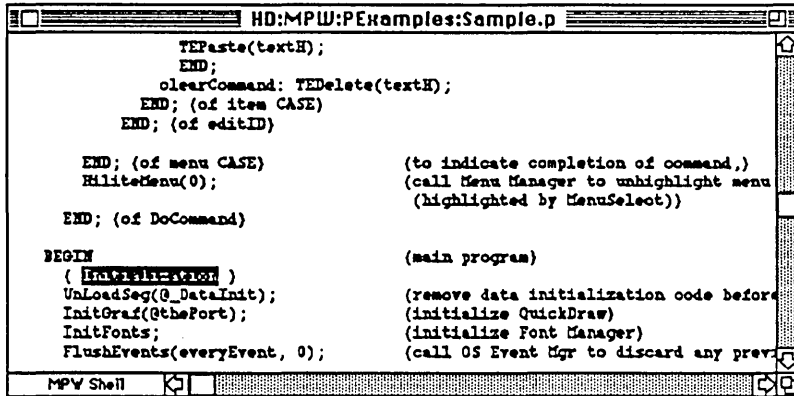
These pattern matching operators are part of a larger set called **regular expression operators**. A regular expression consists of literal characters and/or regular expression operators, and must be enclosed in slashes (/.../). The figure below shows an example.



**Figure 3-8**  
Example of a regular expression



This command finds and selects any string that begins with "init" and is followed by any characters other than a return. Figure 3-9 shows the result of this command.

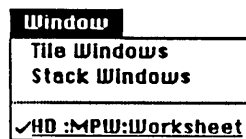


**Figure 3-9**  
Text selected with the Find command

As mentioned, many additional Find and Replace capabilities are available. (See Chapter 6.) In the command language, the Find and Replace functions are performed by the Find and Replace commands. There's also a tool named Search (described in Part II) that can search through a list of files for the occurrence of any text pattern.

## Window menu

The upper half of the Window menu contains the two commands Tile Windows and Stack Windows; the lower half lists all open windows, as shown in Figure 3-10. Selecting a window from the menu brings that window to the front, that is, superimposes it over anything else on your display. A check indicates that the window is currently the "active" window, that is, the frontmost. A bullet (•) indicates that the window is the "target" window, that is, the second to the front. Underlining indicates that a window contains changes that have not yet been saved.



**Figure 3-10**  
Window menu

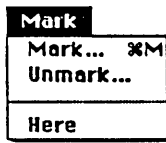
- Tile Windows**     Use this command to arrange windows in a tile pattern on the screen so that each window's contents are visible. Then choose a window and click its zoom box to enlarge it to full screen size. See Part II for information on using TileWindows in scripts.
- Stack Windows**    Use this command to arrange windows in a diagonally staggered pattern on your screen. This is the "open file folder" way to see several windows at once. See Part II for a description of the scriptable version, StackWindows.
- Worksheet**         The Worksheet window always appears first in the Window menu. The menu item lists the full pathname of the worksheet.

---

## Mark menu

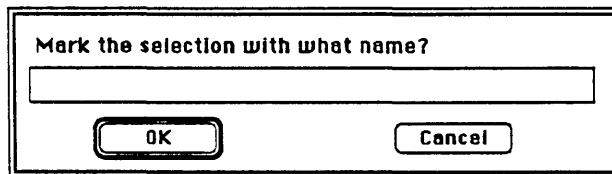
A marker is a text selection that has been given a name. Markers are useful for navigating within a window, and they can simplify many selection expressions. The upper half of the Mark menu contains the commands Mark and Unmark and the lower half lists all existing markers. To jump to the location of a marker you simply choose the name of the marker you want from the Mark menu, shown in Figure 3-11 (only the marker "Here" has been created in this example).

Markers can be created and used both interactively, via the Mark menu, and programmatically, via the Shell commands Mark, Unmark, and Markers. For a detailed discussion of the syntax, characteristics, and programmatic use of markers, see Chapter 6 and Part II.



**Figure 3-11**  
Mark menu

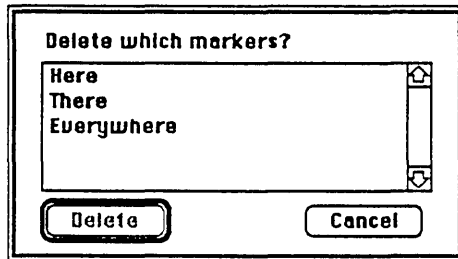
**Mark...** To create a new marker interactively, first select the text you want to mark, then choose "Mark" from the Mark menu. A dialog box like that in Figure 3-12 appears, asking for the name you want the marker to have. The editable text field in the dialog box is initialized to the first word (that is, whatever you would select by a double-click) in the selection. If you click Cancel in the dialog box, the selection is unchanged and no new marker is created. If you click OK a new marker is created with the specified name and the new marker's name is added to the list of marker names displayed by the Mark menu.



**Figure 3-12**  
Mark dialog box

If you try to create a new marker using the name of an already existing marker, a dialog box will appear, giving you the chance either to delete the old marker and add the new (OK), or to forget about adding the new marker (Cancel).

Unmark... If you choose the Unmark menu item from the Mark menu, you'll see a dialog box, like that in Figure 3-13, that contains a list of currently defined markers and the two buttons Delete and Cancel. If a marker is currently selected, its name is highlighted in the marker list. You can select any number of marker names from the list. If you click Delete, every marker selected in the list is deleted. If you click Cancel, the selection remains unchanged and no markers are deleted.

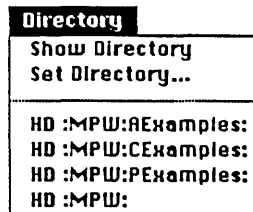


**Figure 3-13**  
Unmark dialog box

---

## Directory menu

The Directory menu, shown in Figure 3-14, lets you display and easily change the name of the default (current) directory. The Directory menu is implemented by the scripts DirectoryMenu and SetDirectory, that you can modify to suit your own needs.



**Figure 3-14**  
Directory menu

Show Directory

An alert box displays the name of the current default directory. Click OK to make the alert go away.

Set Directory...

Sets the default directory. When you select this menu item the Set Directory... dialog box, shown in Figure 3-15, appears, providing interactive selection of the default directory. Your selection is then added to the Directory menu.

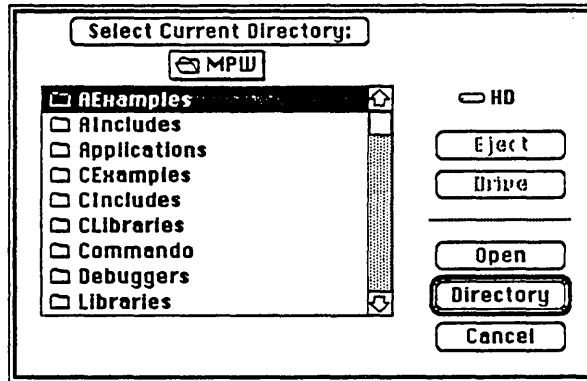


Figure 3-15  
Dialog box of the Set Directory menu item

<DirectoryName>

Selecting a directory name makes this directory the new default directory.

As you select various default directories, using either the Set Directory... menu item or the SetDirectory command, each is added as a separate menu item to make it easy to return to that directory in the future. The UserStartup script creates menu items for each of the Examples folders in the MPW directory, and for the default directory at the time the UserStartup script is run. You can add your own favorite directories by modifying UserStartup.

---

### Warning

Directory names should not contain any of these special characters:

- ; ^ | < / (

These characters have special meanings when they appear as menu items.

---

## Build menu

The Build menu, shown in Figure 3-16, has two primary purposes. First, it is used to create a makefile containing the commands needed to build a program. Second, it is used either to build a specified program or to display the commands needed to do the build. Use of the Build menu is demonstrated in Chapter 2, "Building a Program: An Introduction."

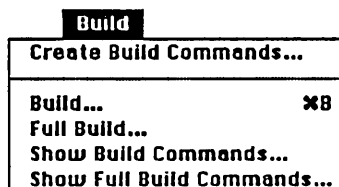


Figure 3-16  
Build menu

Create Build  
Commands...

Creates a makefile containing the build commands for a specified program. When you click Create Build Commands..., the CreateMake dialog box appears. (See Figure 3-17.) You can then enter the program name and select its type (that is, Application, Tool, or Desk Accessory—make sure that you do *not* include a ".a", "-c", or ".p" suffix to the program name).

Click the Files button to select the program's source and library files. (MPW libraries are automatically linked; certain special libraries you may require may not be automatically linked. See CreateMake in Part II.) If the program's name is *program*, a new makefile, called "*program.make*", is created. The makefile will contain simple build commands from the program.

Be sure to run Create Build Commands whenever you create additional source or library files for a program. Answering the CreateMake dialog box generates a new set of rules in *program.make* that includes the new source files.

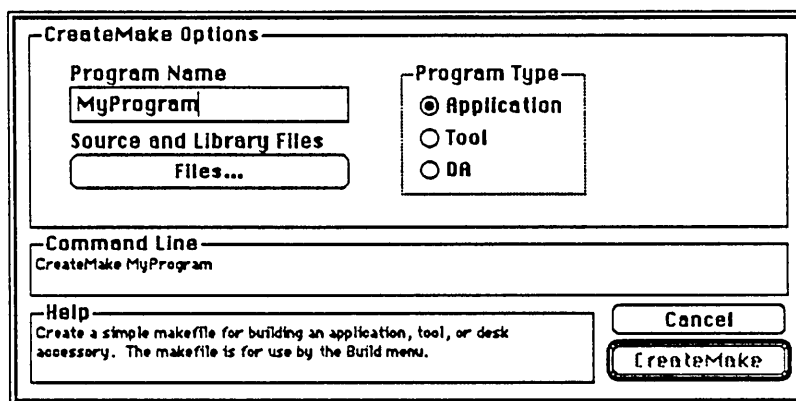
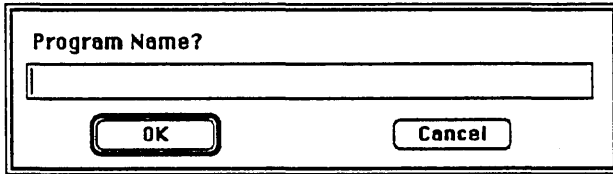


Figure 3-17  
CreateMake dialog box

When you select from the Build menu, one of the following four Build items, a dialog box appears, asking for the name of the program you want to build.



**Figure 3-18**  
Program Name? dialog box

Type the name and click the OK button. The commands used to build the program, plus any error messages, will then be displayed on the worksheet.

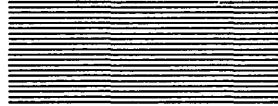
- |                          |                                                                                                                                                                                                                                                                 |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Build...                 | The program is built automatically, but only files that have been modified since you last built the program will be compiled. Use this item to save time. The Command-key equivalent is Command-P.                                                              |
| Full Build...            | The program is completely built, ignoring any object files or intermediate files that may exist from a previous build.                                                                                                                                          |
| Show Build Commands      | The commands needed to build the program (for just those files affected by modifications since the last build) are displayed on the worksheet, but not executed. You can then select any or all of the commands—or modify them—and press Enter to execute them. |
| Show Full Build Commands | All the commands needed to completely rebuild the program (whether modified since the last build or not) are displayed on the worksheet, but not executed. This is a convenient way to see all of the commands used in building the program you've selected.    |

Each of the four Build menu items uses the MPW tool Make to determine which operations are necessary to build the program. The Makefile "program.make" is created by the Create Build Commands... menu item (described above). If you have not used this item—that is, if *program.make* doesn't exist—MPW will use the file MakeFile. (See Chapter 10.)

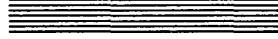
---

## User-defined menus

You can define your own menu commands with the AddMenu command, described at the end of Chapter 5. These commands can be appended to existing menus, or you can create new menus.



## **Chapter 4**



### **Using MPW: The Basics**

This chapter introduces the basic conventions for manipulating files, editing text, executing commands, and responding to dialogs in the Macintosh Programmer's Workshop. You can easily enter all commands, command options, and parameters by using the menus and dialog boxes. The basics for directly typing commands in a window are also introduced. A full discussion of command scripting can be found in Chapter 5. See Chapter 2 for an introduction to building a simple program, using examples contained in the "Examples" folders.

---

---

## Editing

Basic editing functions are available as menu commands. You can open a file with the Open command, or by selecting its name on the screen and choosing the Open Selection command (Command-D) from the File menu. You can select and edit text with the usual Macintosh editing techniques, using menu commands to cut, copy, and paste selected text. The menu commands are described in Chapter 3.

Editing with MPW is unique in that most menu functions are duplicated in the Shell command language. Editing and other command-language functions are fully integrated—you enter and execute editing commands just like any other commands. Editing commands are entered in the **active window** (the frontmost window), but they act on text in the **target window** (the second window from the front), or another window that you explicitly name. The command language lets you produce scripts of editing commands: You can save any series of commands as a normal text file, and execute the file by simply entering the filename. Command-language editing is discussed further in "Editing With the Command Language" in Chapter 5.

---

---

## Entering commands

All MPW commands and their options can be selected from menus and dialog boxes. Generally, this interactive method of command selection is the easiest. You can immediately execute commands selected from menus and dialog boxes, or you can use the dialog boxes to compose complex command lines that can then be copied to a command script.

The dialog boxes for MPW commands are generated by the Commando user interface (described in the last section of this chapter). Besides the usual Macintosh dialog boxes, Commando provides several new forms and controls to handle the special requirements of MPW tools. For example, dialogs for commands with many options may have several nested dialog boxes. Which dialog boxes are actually displayed may vary according to dependency relations between the particular options you may have selected. Some of the specialized dialog controls are introduced at the end of this chapter. Other unique dialog boxes are shown in Part II of this reference with their respective commands. A detailed discussion of all the elements of Commando dialogs can be found in Chapter 14, which explains how to create a Commando interface for your own tools and scripts.

Of course, you can also type commands directly in any window as a series of words separated by spaces or tabs.



---

## Typing commands in a window

By default, command output and any error messages appear in the window immediately below the executed command line. Commands are not case sensitive. You can have multiple open files, and you can enter commands in any window.

The simplest commands consist of the command name only. For example, type the command

```
Date
```

and press the Enter key (without pressing Return first—that is, the insertion point must be on the same line as the command when you press Enter). This command outputs the date and time:

```
Tuesday, July 14, 1987 7:12:00 AM
```

Commands can have options. For example,

```
Date -d
```

The **-d** option tells the Date command to list the date only,

```
Tuesday, July 14, 1987
```

Commands typed into a window are referred to as **standard input**. When the results of the command(s) are then displayed in the same window (the normal, default setting) they are called **standard output**. Any window that is used to enter standard input and display standard output is referred to as the **console**.

Most commands read from standard input, write their output to standard output, and write error messages to diagnostic output. By default, standard input refers to text that is selected and entered while the tool is running. Standard output and diagnostic output appear following the commands. (These input and output defaults can be changed using I/O redirection. See Chapter 5 for details.)

---

## The Enter key

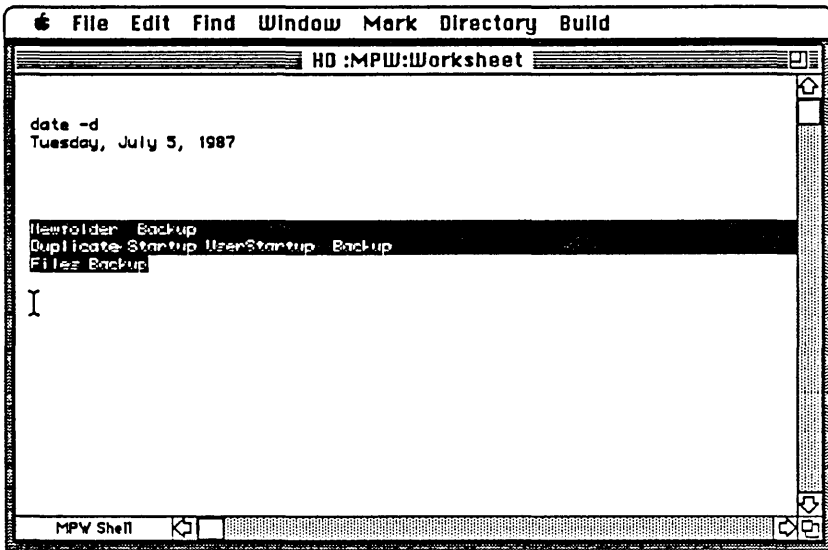
The Enter key serves as a “do it” button, causing commands to be executed. You can type commands in any window and press the Enter key to execute the command line. (When no text is selected, the entire line is executed, regardless of where the insertion point is on the line.) You can also select command text that is already on the screen and press the Enter key to execute the selected text.

Pressing Command-Return is equivalent to pressing the Enter key. In addition, using the mouse to click the status panel in the lower-left corner of the active window is a convenient way to get the same result.

---

## Executing several commands at once

By selecting several lines of command text and then pressing Enter, you can execute any number of commands with one stroke. An example is shown in Figure 4-1.



**Figure 4-1**  
Pressing Enter to execute selected text

In Figure 4-1, executing the selected text would first make a new folder (directory) named Backup, then copy the files Startup and UserStartup into Backup, and then list all of the files in Backup. (Each of these commands, and the pathname syntax, is described in the sections that follow.)

You can also directly execute text files that contain other commands, simply by entering the filename of the script. Executing a script has the same effect as selecting the commands in an open window and pressing Enter—the only difference is the scope of variable and alias definitions (discussed in Chapter 5).

---

### Important

Commands that don't produce any output run silently; this facilitates their use in scripts. Other commands are equipped with a "run silently" option.

---

---

## Terminating a command

To terminate a command while it's executing, press Command-period, the standard Macintosh command for this purpose.

---

### Important

Many commands (including Asm, C, and Pascal) normally take their input from a file; however, if no file is specified, they will begin reading from the console (that is, from the window where the command was entered: "standard input"). If the Shell appears not to be listening to the commands you are entering, it probably isn't: The currently executing command (shown in the active window's status panel) may be reading the text that you enter. If a program is reading from standard input, you can press Command-Enter (or Command-Shift-Return) to indicate end-of-file and terminate input. (See "Terminating Input With Command-Enter" in Chapter 5.)

---

---

## The Help command

The Help command displays summary information for commands. For example, to display a description of the Files (list files) command and its options, type the command

Help Files

and press the Enter key. You'll see the following syntax description:

```
Files [option...] [name...] > fileList
 -c creator # list only files with this creator
 -d # list only directories
 -f # list full pathnames
 -i # treat all arguments as files
 -l # long format (type, creator, size, dates, etc.)
 -m columns # n column format, where n = columns
 -n # don't print header in long or extended format
 -q # don't quote filenames with special characters
 -r # recursively list subdirectories
 -s # suppress the listing of directories
 -t type # list only files of this type
 -x format # extended format--fields specified by format
```

Note: The following characters can specify the format

```
a Flag attributes
b Logical size, in bytes, of the datafork
r Logical size, in bytes, of the resource fork
c Creator of File ("Fldr" for folders)
d Creation date
k Physical size, in kilobytes, of both forks
m Modification date
t Type
o Owner (only for folders on a file server)
g Group (only for folders on a file server)
p Privileges (only for folders on a file server)
```

- ❖ *Note:* In Help texts, the square brackets are a syntax element indicating that a parameter is optional. An ellipsis (...) indicates that the preceding item may be repeated. (Note that this use of the ellipsis is a syntax convention only for Help text and documentation; an ellipsis in an actual command line invokes the command's Commando dialog.) Syntax notation is described in the preface to this manual. The number sign (#) is the MPW comment character.

You can directly edit and execute the text on the screen. For example, assuming that your current directory is [MPW], you can edit the above text as follows:

1. Use the mouse to select [option...] and [name...]; replace them with the option -l and the directory name AExamples.
2. Remove the output specification > fileList.

The result is a command that will list the files in directory AExamples, in long format:

```
Files -l AExamples
```

(AExamples is the directory containing sample assembly-language programs; -l is an option that generates “long” output.) Press Enter to execute the command—directory information will appear immediately following the command.

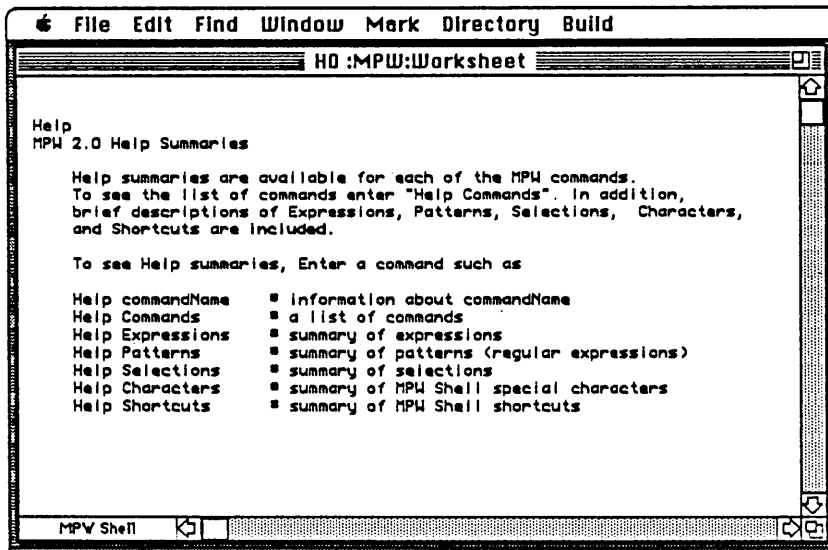
You can also use the Help command to display additional summary information, including

- an annotated list of all MPW commands
- an annotated list of the characters that have special meanings to the MPW Shell
- descriptions of the syntax of expressions, selections, and text patterns
- a summary of MPW Shell shortcuts

For general information about Help, execute the Help command with no parameters:

```
Help
```

This command displays the information shown in Figure 4-2.



**Figure 4-2**  
Help summaries

You can directly execute the Help commands given in the “Help Summaries” list.

❖ *Note:* The MPW Help file should be in the same directory as the MPW Shell or in the System folder.

---

---

## File management commands

The MPW Shell lets you manipulate files without returning to the Finder. Table 4-1 introduces the most commonly used file management commands.

❖ *Note:* The descriptions in the table omit some of the command options that are available. For complete descriptions, see Part II.

**Table 4-1**  
Basic file management commands

---

|                                                            |                                                                                                                                                               |
|------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Backup [option] -from folder -to folder [ <i>file...</i> ] | Copy files in source folder to destination folder based on modification date. This is useful when you maintain an identical backup folder on a separate disk. |
| Catenate [ <i>file...</i> ]                                | Read the data fork of each file and write it to standard output. (By default, standard output is to the active window, immediately after the command.)        |
| Delete <i>name...</i>                                      | Delete file or directory <i>name</i> . If <i>name</i> is a directory, all of its contents are deleted.                                                        |
| Directory <i>directory</i>                                 | Set the default directory to <i>directory</i> .                                                                                                               |
| Directory                                                  | Directory with no parameters writes the pathname of the current directory.                                                                                    |
| Duplicate <i>name...</i> <i>targetName</i>                 | Duplicate file or directory <i>name</i> to file or directory <i>targetName</i> .                                                                              |
| Exists <i>name...</i>                                      | Determine the existence of file or directory <i>name</i> .                                                                                                    |
| Files [ <i>name...</i> ]                                   | List names of directories and files. Options allow you to include various attributes in the listing.                                                          |
| GetFileName [ <i>options...</i> ] [ <i>pathname</i> ]      | Display a standard file dialog box.                                                                                                                           |
| Mount <i>drive...</i>                                      | Mount volumes.                                                                                                                                                |
| Move <i>name...</i> <i>targetName</i>                      | Move file or directory <i>name</i> to <i>targetName</i> .                                                                                                     |
| New [ <i>name...</i> ]                                     | Open a new window.                                                                                                                                            |
| Newer [ <i>option</i> ] <i>name...</i> <i>target</i>       | Compare modification dates between files <i>name</i> and <i>target</i> . List files newer than <i>target</i> .                                                |
| NewFolder <i>name...</i>                                   | Create the new directory <i>name</i> .                                                                                                                        |
| Open [ <i>option</i> ] [ <i>names...</i> ]                 | Open a window.                                                                                                                                                |
| Rename <i>name1 name2</i>                                  | Rename File or Directory <i>name1</i> to <i>name2</i> .                                                                                                       |
| Revert                                                     | Revert window to previous saved state.                                                                                                                        |
| Save [-a   <i>windows...</i> ]                             | Save a window.                                                                                                                                                |
| SetFile [ <i>option...</i> ] <i>file...</i>                | Set file attributes.                                                                                                                                          |

(continued)

**Table 4-1** (continued)  
Basic file management commands

---

|                                                  |                                                                                                                                                                                                                  |
|--------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SetDirectory <i>directory</i>                    | Set the default directory.                                                                                                                                                                                       |
| SetPrivileges [ <i>option</i> ] <i>folder...</i> | Set access privileges for folders on the file server.                                                                                                                                                            |
| SetVersion [ <i>option ...</i> ] <i>file</i>     | Independently maintain the version and revision numbers as a resource in the application or tool. Optionally, update a version and revision string in a source file.                                             |
| Target <i>name</i>                               | Make a window the target window.                                                                                                                                                                                 |
| Volumes [ <i>name... </i> ]                      | List mounted volumes.                                                                                                                                                                                            |
| Which [ <i>command</i> ]                         | Determine, for the specified <i>command</i> , which existing aliases, Shell built-in commands, and commands accessed via the Shell variable { <i>Commands</i> } will be executed when <i>command</i> is entered. |
| Windows                                          | List open windows.                                                                                                                                                                                               |

---

---

## File and window names

In the MPW, files and windows are specified in the same way. When a name is passed as a parameter to a command, the system looks first for an open window with that name; if no window is found, it looks for a file on the disk.

The following rules apply to naming:

- Names are not case sensitive.
- A single component (file or directory name) of an HFS pathname is limited to 31 characters.
- Any character except a colon (:) may be used in a file or directory name. (Colons separate elements in a pathname.)

It's wisest to avoid using spaces and special characters in filenames. When using filenames that contain spaces, you'll need to **quote** them so that they won't be interpreted as individual words in the command language—for example, you would need to specify the name "System Folder" as follows:

```
Files "HD:System Folder"
```

For the rules concerning quoting, see "Quoting Special Characters" in Chapter 5.

---

## Selection specifications

Commands that take filenames for parameters can also act on the **current selection** in a window. The current selection character, § (Option-6), represents the currently selected text in a window. There are two ways to use this character:

§           Currently selected text in the target window. (The target window is the second window from the front, as explained in Chapter 1.)

*name*.§    Currently selected text in window *name*.

For example, the Count command counts lines and/or characters in a file. The command

```
Count -l Sample.a.§
```

counts the lines within the current selection in the window Sample.a.

The current selection is explained more fully in “Editing With the Command Language” in Chapter 5.

❖ *Note:* The MPW Shell uses a number of special characters (like §) from the extended character set. These characters are fully listed in Appendix C.

---

## Directories and pathnames

With the hierarchical file system (HFS), specifying a filename alone is often not enough to identify a file—you frequently need to specify a pathname. A **full pathname** is specified as follows:

*volume* :[*directory* :...] *filename*

A full pathname contains at least one colon (:), but cannot begin with a colon. An example of a full pathname is

```
"HD:MPW:MPW Shell"
```

(The quotation marks are required because the filename “MPW Shell” contains a space.)

A partial pathname is usually all you’ll need to specify. When HFS encounters a partial pathname, it begins the path at the current default directory. *Any name that contains no colons or begins with a colon is considered a partial pathname. A partial pathname that contains no colons is a leafname.* For example, the name

```
:AExamples
```

is taken as a partial pathname. However, the name

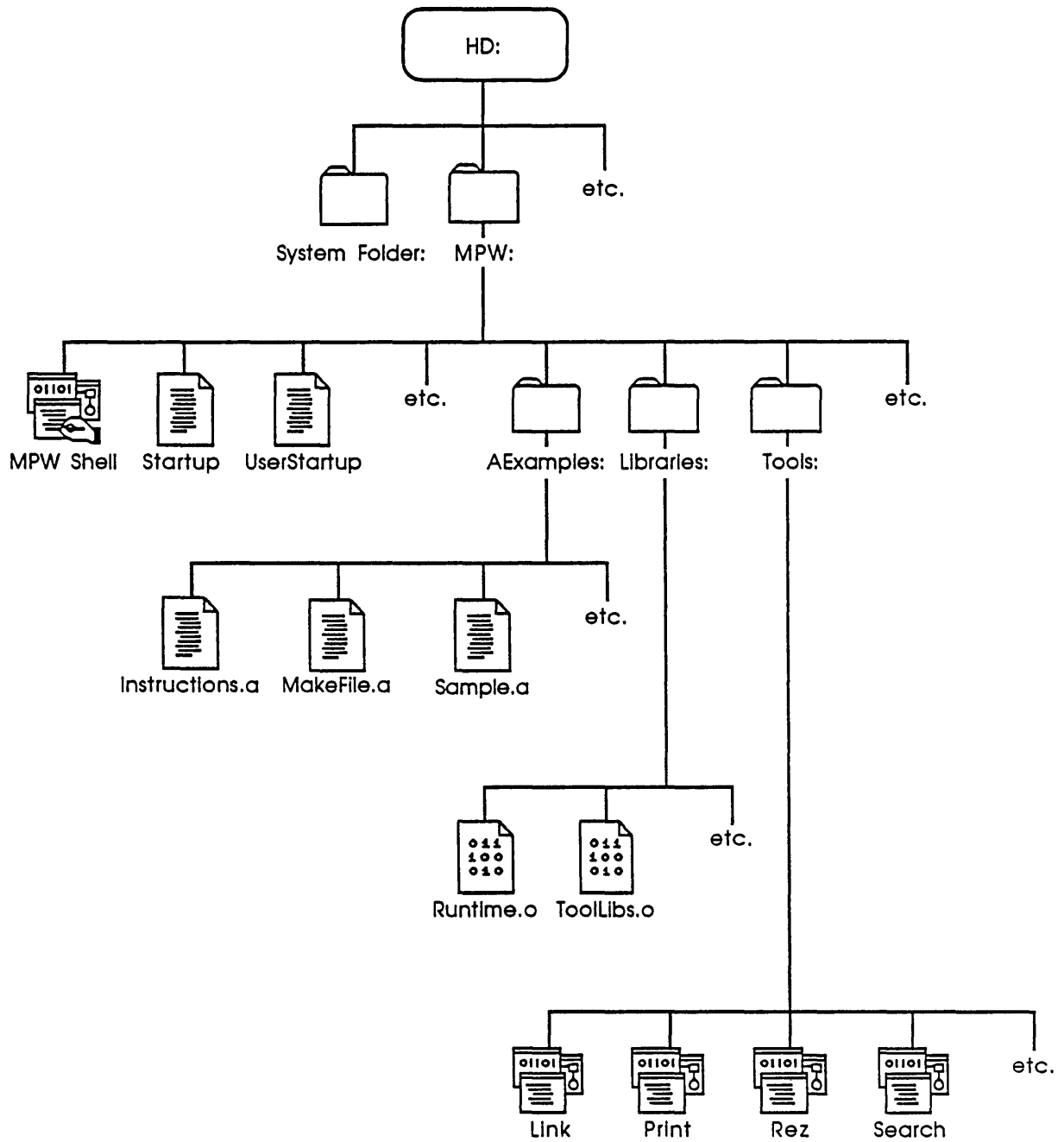
```
MPW:
```

is taken to be a *full* pathname (that is, a volume name only), rather than meaning the directory HD:MPW. (When in doubt, you can always specify the full pathname for a file or command.)

Double colons (::) in a pathname specify the current directory’s parent directory; triple colons specify the “grandparent” directory (two levels up), and so on. See the File Manager chapter in Volume IV of *Inside Macintosh* for more information on HFS conventions.

❖ *Note:* Notice that there’s no single “root” directory—each volume name (that is, disk name) is a separate starting point for a directory tree.

Figure 4-3 is a directory tree showing the arrangement of MPW files and folders.



**Figure 4-3**  
Hierarchical directory structure



You can use the Files command to list the names of files and directories. For example, the command

```
Files HD:MPW:
```

might display the following:

```
:AExamples:
:AIncludes:
:Applications:
:Libraries:
:RIncludes:
:Tools:
'MPW Shell'
MPW.Help
Quit
Resume
Startup
Suspend
SysErrs.Err
UserStartup
Worksheet
etc.
```

In the output of the Files command, the names that begin and end with colons are directory names, and the other names are filenames. All of these names are partial pathnames—in this case, “HD:MPW” forms the beginning of each pathname. Also note that filenames containing special characters are quoted.

---

## Command search path

When you enter a command name (that is, a leafname), the Shell searches for the command in the directories listed in the Shell variable {Commands}. As described in Chapter 5, this search path is initially set to

```
:(the current directory)
HD:MPW:Tools:,
HD:MPW:Applications:,
HD:MPW:Scripts:
```

This means that when you type any command the Shell first assumes that you want to execute a tool; if it can't find the tool, it then assumes that you want an application; if it can't find the application, it then assumes that you want a command script. If your frequency of use is different, you can change the search path to improve the Shell's performance. (See Chapter 5.)

---

## Changing directories

You can change the default directory with the Directory command. Assuming you have a hard disk named HD, you could change the default directory to the directory AExamples in the MPW folder with the command

```
Directory HD:MPW:AExamples
```

Like most commands, Directory runs silently—it generates output only if an error occurs. To verify that you have set the appropriate directory, enter the Directory command with no parameters:

```
Directory
```

This command displays the default directory.

Remember that to specify a pathname containing spaces or other special characters, you'll need to quote it by surrounding it with single or double quotes. (See Chapter 5.)

### An aside: the Alias command

For frequently used commands such as Directory, you may get tired of typing the entire command name. You can easily define your own alternate names with the Alias command. For example,

```
Alias dir Directory
```

After executing this command, you can execute the Directory command by entering the new command name:

```
dir
```

To make an alias definition part of the Shell's standard startup procedure, place the definition in the file UserStartup. See Chapter 5, "The Startup and UserStartup Files."

---

## Pathname variables

One way of specifying a pathname is by using Shell variables. For example, the Shell variable {MPW}, defined in the Startup file, expands to form the full pathname for the MPW folder, in this case "HD:MPW:" (assuming that the MPW folder is at the top level). Thus, the previous Directory command could be entered as

```
Directory "{MPW}AExamples"
```

In this particular case, the quotes aren't necessary. If you adopt the practice of never using spaces or other special characters in a pathname, then you don't need to bother with quotes. On the other hand, if a pathname may contain spaces or other special characters, then it would be a good idea to use quotes whenever variables are included in a pathname.

You can use the Set command to define and redefine variables, as described in Chapter 5. To see the values of all currently defined variables, enter the Set command with no parameters:

```
Set
```

---

## Wildcards (filename generation)

You can specify a number of files at once by using the wildcard characters ? and ≈ (Option-X). The ? character matches any single character (except a colon or Return); ≈ matches any string of zero or more characters (other than colon or Return). For example, the command

```
Files ≈.a
```

lists all filenames in the current directory that end with the suffix “.a”. (Several other wildcard characters can also be used to generate filenames—see “Filename Generation” in Chapter 5.)

---

---

## Commando dialogs

The Commando user interface lets you operate any properly configured MPW command by means of a special Macintosh dialog, rather than the traditional command line interface. Commando dialogs may consist of several dialog boxes containing a variety of controls. You can choose options, select filenames, pick directories, and read help information for each option. Commando lets you operate MPW commands in a more intuitive format. All options are visible, and help text for each option can be instantly displayed in the frontmost dialog box.

Because of the complexity of many MPW commands, several specialized controls and nested dialog boxes have been implemented for them. The various types of controls and dialog boxes are introduced below. Other unique dialog boxes, specific to a particular command, are described with the command in Part II.

---

## Invoking Commando dialogs

To invoke a Commando dialog for tools and other commands from the worksheet: type the command name followed by either an ellipsis (...) or the word Commando in front of the command line.

- ❖ *Note:* To get the ellipsis character, hold down the Option key while simultaneously typing the semicolon (;) character. Although three periods closely resemble an ellipsis character, Commando won't be fooled; you *must* use Option-semicolon to get the true ellipsis character that invokes Commando.

The ellipsis may appear anywhere in a command line (except with quotes or after `␣`) and it is considered a word-break character. The ellipsis invokes the Commando user interface after the Shell has carried out all alias and variable substitutions. The entire command line is passed to Commando and the output of Commando is then executed by the Shell.

Alternatively, if you don't want the resulting command line to be immediately executed, you can type

```
commando toolname
```

In this case, Commando will not find a command if the command has been aliased to a different name. The tool's frontmost Commando dialog box is displayed. Clicking the Do It button writes the command line to standard output (that is, the window in which you typed the command) instead of executing it immediately. This second method of using dialog boxes is useful for building command lines that are to be cut and pasted into scripts.

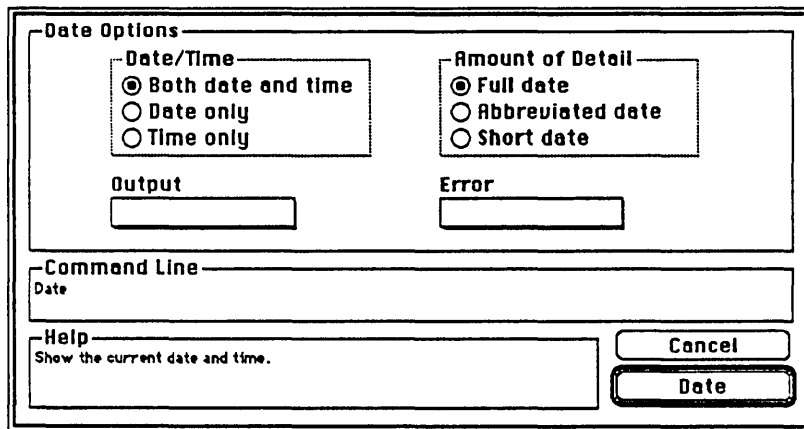
---

## Using Commando dialogs

The function and appearance of Commando dialog boxes may vary widely according to the syntax and semantics of the particular command or tool selected. The basic dialog box is typical of a simple command such as Date, the first example used above. Type

Date ...

Be sure to use Option-semicolon to get the ellipsis. Then press Enter. Figure 4-4 shows the resulting Commando dialog box for Date.



**Figure 4-4**  
Date dialog box

Most dialog boxes share the basic structure shown in Figure 4-4. Various controls for options and parameters appear in the largest, upper area of the box. Date has two parameters, “Date/Time” and “Amount of Detail.” Each of these parameters is defined by a radio button control. The default settings for Date appear preselected as the topmost radio button for each parameter.

Clicking and holding down the mouse button on any control or option displays help information in the standard help window at the bottom of every Commando dialog box.

Use the output box to redirect output. See the section “Redirecting Output” later in this chapter.

The Command Line window displays the command line resulting from the options you select from the dialog box. As you select options or change parameters, this Command Line box is continuously updated.

Clicking the lower-right Do It button passes the completed command line back to the Shell for execution. Alternatively you can press the Enter key. If you change your mind, you can click the Cancel button, which is the same as pressing Command-period.

---

## Standard dialog box controls

This section describes the most frequently encountered Commando dialog box controls.

### Generic text parameters

Not only do tools have options, they also have parameters. Nonspecific parameters, where the parameter can be just about any string, are simply entered in a editable text field. For items where text is required, the text is quoted by Commando before being passed to the Shell. You can scroll the line right and left (by dragging) if the text in the box is longer than the text box. Here's an example of an editable text field:

Mark the selection with what name?

### Repeatable options

Various text field options, such as the `-d[define]` option in Rez and Asm, may be specified more than once. The control below shows an option of this type. The number of lines displayed is controllable by the tool's builder. The small window is basically an area where text can be entered, very much like the Notepad desk accessory. This window does not automatically wrap around lines larger than the window area. Instead, it scrolls left and right. You create a new line by pressing the Return key. Scroll the window horizontally by dragging. The window is scrolled vertically either by dragging or by the scroll bar control.

Preprocessor defines:

|                  |   |
|------------------|---|
| Language=english | ↑ |
| size=height*200  | ↓ |

### Radio buttons

Some options are mutually exclusive and are therefore available as a set of radio buttons. The default setting of the button corresponds to the default state of the option. Groups of mutually exclusive items are often surrounded with a labeled perimeter:

|                                                                                                                                            |
|--------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Print Quality</p> <p><input type="radio"/> High</p> <p><input checked="" type="radio"/> Standard</p> <p><input type="radio"/> Draft</p> |
|--------------------------------------------------------------------------------------------------------------------------------------------|

## Check boxes

An option, such as the Assembler's **-print** option, may have many simultaneous settings. Options like this are implemented with check boxes (versus on/off radio buttons). Most of the MPW tool's options are Boolean flags. Check boxes are used for these types of options also and are usually surrounded by a labeled perimeter.

**Listing Control**

- Show macro expansions
- Allow automatic page ejects
- Show warning messages
- Show macro call statements
- Show generated object code
- Show up to 255 bytes of data
- Show macro directive lines
- Show header lines
- Show generated literals
- Show assembly status

## Shadow pop-up menus

Some options require the name of a window, alias, font, or Shell variable. Commando will display a field of this type with a shadow:

Window

When you click inside the shadowed box, a pop-up menu displays all the choices for that particular field (that is, windows, aliases, fonts, or Shell variables). The menu box is aligned around the current selection. Also, the current selection is checked in the menu box. As long as the mouse button is held down, the menu behaves like the standard pull-down menu. If necessary, the pop-up menu will scroll. When the mouse is released within a menu item, that item then appears in the shadowed box.

Window

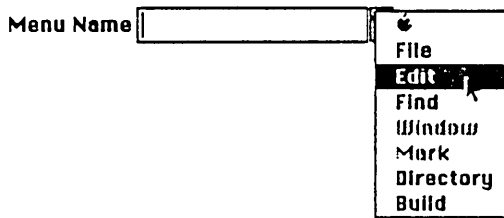
- ✓ HD:0S:Worksheet
- HD:MPW:MyCroft:test.c
- HD:MPW:MyCroft:getopt.c

## Other pop-up variations

Some options are similar to the pop-up menus above but also allow a little more flexibility. The Menu Name box in AddMenu allows you to type in the name of a new menu or select an existing menu name from a list of names.

Menu Name

Click on the menu icon at the right of the box to display a pop-up menu containing the existing choices.



Drag down the pop-up menu until the item you want is highlighted and then release the mouse button. The selected item will appear in the text-edit box.

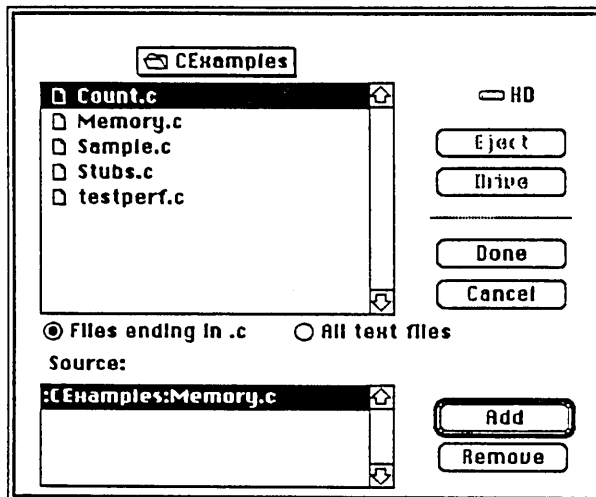
### Multiple input files

When a tool can handle multiple input files of the same type (C, ASM, Rez, and so on), only a single button is displayed.

#### Source Files

Clicking on the button displays a modified standard file dialog box. Commando adds some functionality to the standard file package (SFGetFile) to let you select multiple files in different directories. Another scrollable list appears under the file list. Use the standard file controls to select files. After you've selected a file with the Add button, the dialog box does not go away. Instead, the file is added to the lower list. You can delete a file from the list by selecting it (in the lower list) and clicking Remove. When all desired files have been selected, click Done or Cancel to return to Commando's dialog box.

A tool may tell Commando that the tool requires files with a particular extension. A radio button lets you display and select any text file (or whatever type the tool wants). When you select a folder, the Open button reads "Open." When a file is selected, this same button is labeled "Add." If you select a file that has already been added to the lower list, then the file in the lower list is selected (and scrolled into view if necessary), and the Remove button undimmed.

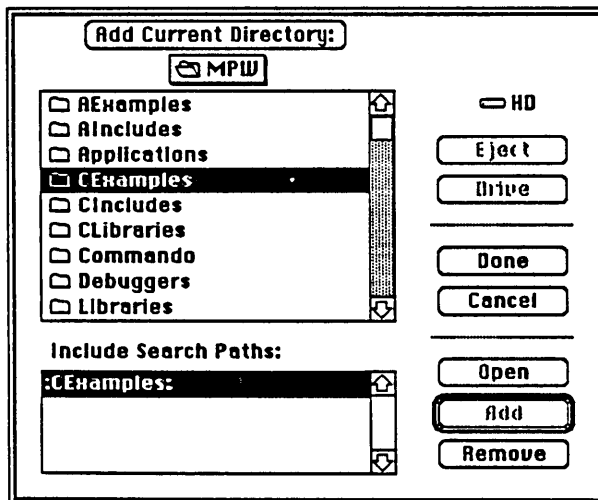


## Multiple directories

Some tools, such as C and Asm, have options that let you specify directories to search when looking for various files. Clicking a single button, like this one, will display a modified standard file dialog box:

### Include Directories

The selection of multiple directories works the same way as the selection of multiple files. In this example, however, only folders are visible. Because a selected directory has the potential for being both *opened* and *added* to the lower list, there must be two controls for both operations. Clicking the Add button adds the directory selected in the upper list to the lower list. The Open button operates in its normal manner: Clicking it opens the selected folder. You can delete a directory from the lower list by selecting it (in the lower list) and clicking Remove. Finally, clicking Continue or Cancel returns control to Commando.



## Multiple files and/or directories

For MPW tools or built-in commands that can deal with both multiple files and directories, this dialog box, almost the same as the one shown above, lets you select files and directories. The model is almost the same as the one above, except that both files and folders are visible. Selecting anything in the upper scroll window highlights the lower Add button. The controls work as shown in the example above.

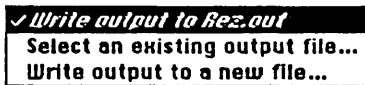


### Single input or output file

You select options or parameters that require a single file (whether it be for input or output) with a control similar to the example below. Clicking in the dotted rectangle displays a pop-up menu with choices depending upon the tool. The first choice can be either Default Output or No Output (or, if the file is an input file, Default Input or No Input). The Default Output is used for tools that write to a default output file if one is not specified. Link and Rez, for example, write to link.out and rez.out if no explicit output file is specified. If Input File... or Output File... is selected, SFGGetFile (for input files) or SFPutFile (for output files) is displayed so that a file can be chosen. If the filename selected is too long to fit in the space provided, the middle of the path is annotated with "...".

Resource Output File

Here's an example of an output file pop-up menu:

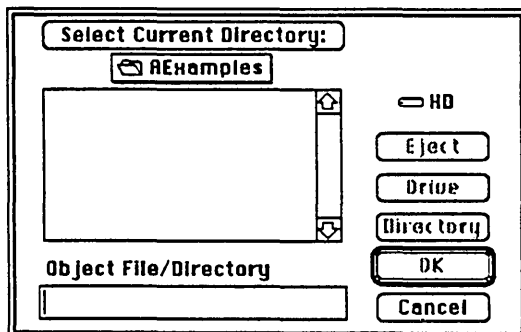


### Output file where a file or directory may be specified

The various compilers have options to specify the object filename or the object file directory. Commando displays a pop-up menu similar to the one above, except that the standard dialog box that appears when you select the Output File or Directory... item looks like the one below. When you select a directory in the file's window, the File button changes to Directory. The File button is dimmed when the text-edit field is empty.

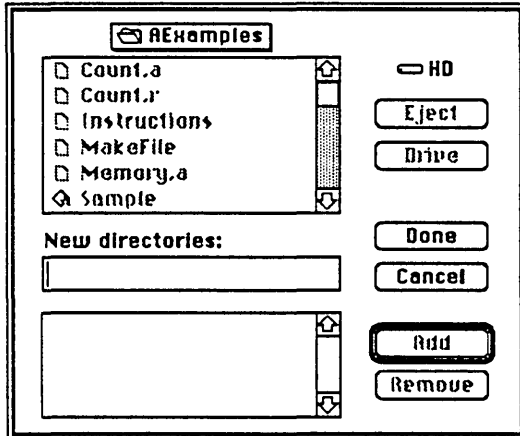
File/Directory  *Use default naming convention*

This dialog box appears when the Output File or Directory... item is selected:



## New directories

The NewFolder command lets you specify the creation of multiple directories. The example below (based upon SFPutFile) is used to create multiple directories. When you type a directory name in the middle text-edit area and click the Add button (or press Return), a pathname is added to the lower list. The root of the new directory is the same as what is displayed in the upper scroll list. You can continue to add more directories. Click the Done button to close the dialog box.



---

## Special dialog box controls

Commando uses standard Macintosh text-edit boxes, radio buttons, and check boxes. In addition to these, you'll encounter some specialized controls, because of the variety of options and parameters and certain dependencies between them. These various types of specialized controls are introduced below.

### Nested dialog boxes

Some tools, such as Rez and PasMat, have more options and parameters than can fit into one dialog box. The additional options are grouped into nested dialog boxes that are available from the first dialog box. Figure 4-5 below shows, as an example, the first dialog box of Rez.

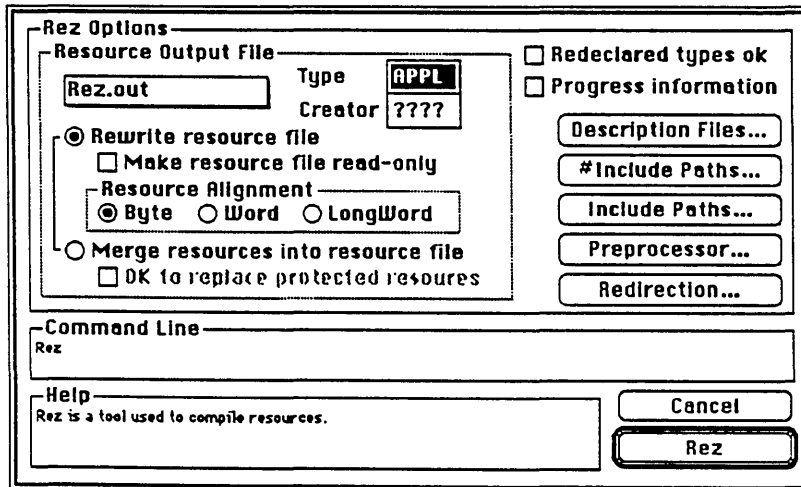
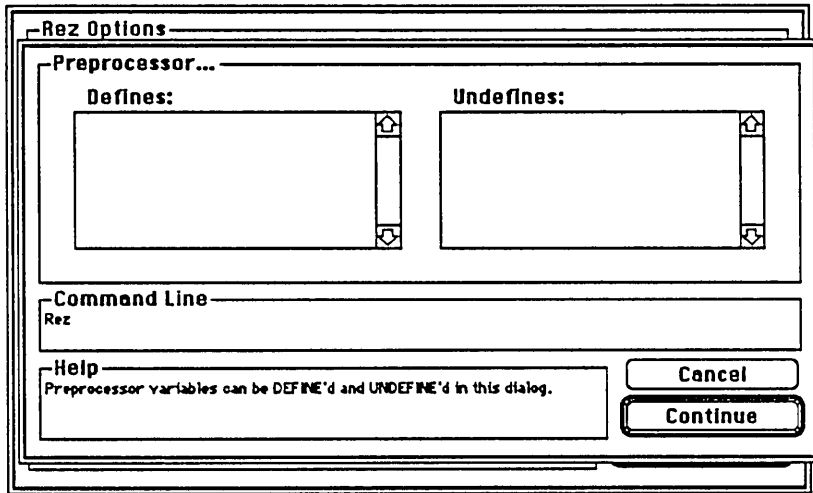


Figure 4-5  
Rez: the first dialog box

Note the five control buttons at the right side of the “Rez Options/Parameters” window. When you click one of these buttons, a nested dialog box appears with the title of the selected button. For example, selecting the button labeled “Preprocessor...” displays the nested dialog box shown in Figure 4-6.

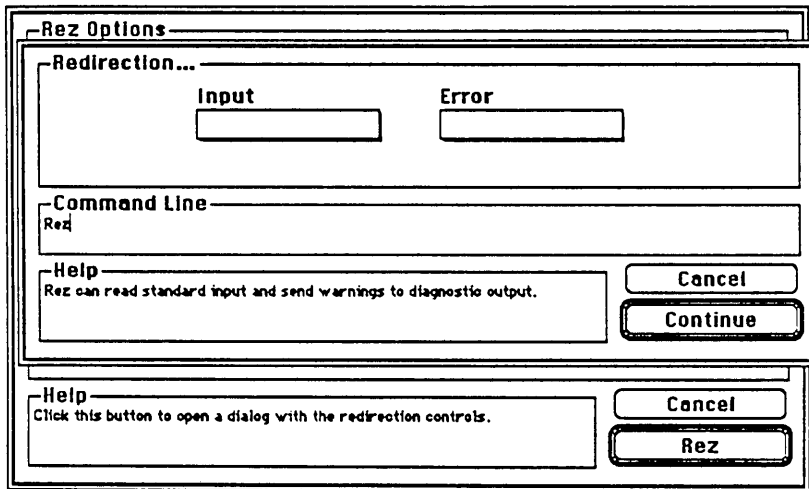


**Figure 4-6**  
Rez: nested Preprocessor dialog box

As you type in the preprocessor defines and undefines, the command line you began in the first dialog box is further updated in the Command Line window of the nested dialog box. The lower-right Do It button in a nested dialog is always labeled “Continue.” Clicking Continue closes the nested dialog box, and again displays the first dialog box with the command line updated to show the options and parameters selected in the nested dialog box. If you click Cancel, changes from nested dialog boxes are not recorded and you return to the first dialog box. From there you can then select another nested dialog box.

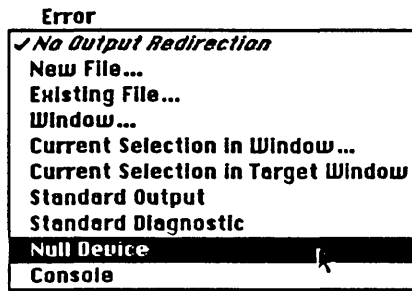
### Redirecting output

Every tool that can write information to standard output or to standard error has controls to assign destinations for this output. Consider the Error Output window in the Redirection nested dialog box of Rez, shown in Figure 4-7.



**Figure 4-7**  
Rez: nested Redirection dialog box

Clicking inside the Error window displays the pop-up menu shown below.



Here "Null Device" has been selected. When the mouse is released, the filename dev:null appears in the Error window. Whenever you select an output redirection, the two radio buttons directly beneath the Error window are activated.

Selecting Existing File in the pop-up menu displays the standard file dialog box. Selecting New File brings up the standard output file dialog box and lets you create a new file. Selecting Window brings up a list of the windows to choose from. Because a window is a file, a window could also be chosen with the Existing File command. When you select Current Selection in Target Window, output is redirected to `§`. Selecting Current Selection in Window also brings up a list of windows to choose from. When you choose a window, output is redirected to `window.§`. When you choose any file other than a new file, the Overwrite and Append buttons are activated. These buttons correspond to the functions of the `>`, `≥` and `>>`, `≥≥` redirection operators, respectively. Selecting No Output Redirection clears the dimmed box so that no redirection occurs.

After you release the mouse over Null Device, the command window looks like this:



The Diagnostic Output windows and Standard Input windows (in the case of tools that read standard input) work in a similar fashion.

### Options dependent on other options

Some options may be dependent on another option. For example, the `-hf` (header font) and `-hs` (header size) options of the print tool don't mean anything unless the `-h` (header) option is specified. Commando implements this model by disabling all controls dependent upon some other control. When you check (or otherwise activate) the main control, the dependent controls are enabled. Another example is the AddMenu command. The syntax of this command is

```
AddMenu [menuName [itemName [command...]]]
```

An itemName cannot be entered until a menuName is entered. Likewise, a command cannot be entered until an itemName is entered.

Menu Name

Item Name

Commands

Here is the same set of options after "Find" has been typed in the first text-edit entry field. Notice that as soon as something is entered in the field, the "Item Name" entry is enabled, but the Commands field remains dimmed.

Menu Name

Item Name

Commands

When an item is selected for the Item text-edit box, the Commands field is enabled.

Menu Name

Item Name

Commands

There may be several text-edit boxes that are disabled (dimmed) until you have entered something in the adjacent enabled text-edit box.

### Three-state controls

Some options, like the **-a** option of Setfile, need the support of a three-state control. For example, Setfile can set, clear, or do nothing to the bundle bit. Clicking this control cycles through its three states. The color of the diamond determines its state:

- Gray—Don't touch the flag
- White—Clear the flag
- Black—Set the flag

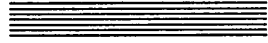
**Attributes**

- Locked
- Invisible
- Bundle
- System
- Protected
- Open
- Changed
- Init'd
- on Desktop





## **Chapter 5**



### **Using the Command Language**

So far, we've introduced only isolated groups of commands without treating the Shell's command language as a whole. This chapter describes the complete syntax of the MPW command language and explains its use. Each command is defined in detail in Part II.

---

---

## Overview

The command language provides the following features:

- built-in and user-definable variables of the form *{variableName}*
- command aliases, used to create alternate names for commands
- command substitution, by which commands enclosed in back quotes ( `...` ) are replaced by their output
- a quoting mechanism for disabling special characters or inserting invisible characters in text: `∂` literalizes a single character; '...' and "..." quote strings
- an extensive set of structured commands for controlling the order of command execution, including `Begin...End`, `If...Else...End`, and `For...In...End`
- filename generation with "wildcard" operators such as `*` and `?`
- redirection of input and output with the `<`, `>`, `>>`, `≥`, and `≥≥` operators

When you enter command text, the Shell first interprets and processes all special symbols, before actually running the command. The order of interpretation is explained later in this chapter under "How Commands Are Interpreted." For the most part, the order of presentation in this chapter follows the order of interpretation by the Shell.

In order to begin using MPW, you should read the following sections of this chapter as a minimum:

- the opening sections of the chapter, which describe the basic form of all commands: "Types of Commands," "Entering and Executing Commands," and "Structure of a Command"
- "Command Scripts" and "Special Scripts"
- "Variables"
- "Quoting Special Characters"

The operators and syntax of the command language are summarized in Appendix D.

---

---

## Types of commands

In all, four kinds of commands are provided:

- **Built-in commands**, such as `Files` or `Duplicate`, are part of the MPW Shell.
- **Command scripts**, such as `Startup`, are text files that contain commands. You can combine any series of MPW commands in a text file, and execute the file by entering its filename, just like any other command. You can also pass parameters to a script and use them in commands within the file.
- **Tools**, such as `Link` or `Asm`, are executable programs (that is, separate files on the disk) that are fully integrated with the Shell environment.
- **Applications**, such as `ResEdit` or `MacPaint`<sup>®</sup>, are stand-alone programs that can be launched from the Shell, but run outside the Shell environment.



To execute a tool, application, or script, you need to have the proper program file on your disk.

- ❖ *Note:* A built-in command overrides a script or executable program with the same name. You should therefore use either full pathnames or quotes to specify a command file or program with the same name as a built-in command. (Quotes work for this purpose because the names of built-in commands must appear unquoted—see “Quoting Special Characters” later in this chapter.)
- ❖ *Note:* The Shell will not execute tools whose modification date is 12:00 AM 1/1/04.

---

---

## Entering and executing commands

Press the Enter key to execute command text. You can select command text on the screen and press Enter to execute the selected text. If no text is selected, pressing Enter executes the entire line that contains the insertion point. Alternatively, you can use the mouse to click the Status Panel in the worksheet’s lower-left corner, or press Command-Return; both of these have the exact same result as pressing the Enter key.

---

### Important

If no text is selected, pressing Enter always passes the *entire line* to the Shell (or to whatever other program happens to be reading from the console—this rule also applies to your own integrated programs that run within the Shell).

---

### Caution

If you enter a line that ends with the Shell escape character, `^_`, the command interpreter will pause, waiting for the rest of the line.

All commands return a **status value**: 0 indicates successful completion; nonzero values usually indicate an error. This value is returned in the `{Status}` variable, described later in this chapter.

---

## Negative status

The command interpreter will return negative status values when it encounters an error. These values are as follows:

- 1 Command not found, script is a directory, script is not executable, or script has a bad date.
- 2 Filename expansion failed or there was an error in the expression syntax.
- 3 Bad syntax. Quotes and braces were not balanced, or were missing end or “)” command. Error in control constructs.
- 4 Missing filename following I/O redirection or the file could not be opened.
- 5 Invalid expression (If, Break If, Continue If, and other such constructs).

- 6 Tool could not be started.
- 7 Runtime error during tool execution. Most likely an out-of-memory error.
- 8 User aborted the tool from the debugger.
- 9 User aborted the tool with Command-Period.

These values can be used to distinguish between errors returned by the commands themselves and errors returned by the Shell.

---

### Important

All negative numbers are reserved for the Shell. Use only positive numbers for errors in tools or scripts.

---

---

---

## Structure of a command

A command is written as a list of words separated by blanks. (Blanks may be either space or tab characters.) The first word is the name of the command, and each word that follows is passed as a parameter to the command. The general form of a simple command is

*commandName* [ *parameters...* ] *commandTerminator*

Each of these elements is described below.

---

### Command name

The **command name** is either the name of a built-in command or the filename of the program or script to execute. Command names are not case sensitive. Alternate names can be defined for a command—see “Command Aliases” in this chapter for information.

The command name is passed to tools and scripts as parameter 0, and can be referenced by scripts in the variable {0}, explained later in this chapter under “Variables.”

---

### Parameters

Each of the subsequent words in a command is a **parameter** to the command or to the command interpreter. Note that certain parameters, such as I/O redirection, are interpreted by the Shell, and never seen by the command itself. Variables are also interpreted before being passed to the program.

By convention, there are two distinct types of parameters to commands: **options** and **files**. See the “Command Prototype” section at the beginning of Part II for more details on these conventions.

You can reference parameters within scripts by using the variables {1}, {2},...{n}. (See Table 5-4.)

---

## Command terminators

Each command is normally terminated by a return character. Commands can also be terminated by the pipe symbol (`|`), the conditional execution operators (`&&` and `||`), or the simple command terminator (`;`). Each of these symbols may be followed by a return. Table 5-1 describes the command terminators in order of decreasing precedence.

Except as modified by structured commands, commands are read sequentially and executed as they are read.

**Table 5-1**  
Command terminators

---

|                                   |                                                                                                                                                                                                                                                                    |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>cmd1   cmd2</code>          | Saves the standard output of <code>cmd1</code> in a temporary file and uses it as the standard input of <code>cmd2</code> . (Standard I/O is explained later in this chapter.)<br><br><i>Note:</i> In MPW, unlike UNIX systems, the commands execute sequentially. |
| <code>cmd1 &amp;&amp; cmd2</code> | Executes <code>cmd2</code> only if <code>cmd1</code> succeeds (that is, returns a status value of 0).                                                                                                                                                              |
| <code>cmd1    cmd2</code>         | Executes <code>cmd2</code> only if <code>cmd1</code> fails (returns a nonzero status value).                                                                                                                                                                       |
| <code>cmd1 ; cmd2</code>          | Executes <code>cmd1</code> followed by <code>cmd2</code> ; this terminator allows more than one command to appear on a single line.                                                                                                                                |

These command terminators may be applied to both simple and structured commands. They all group from left to right. Parentheses can be used to group commands for conditional execution and pipe specifications. Some examples follow.

```
Files | Count -l
```

This command pipes the output of the Files command (a list of files and directories) to the Count command, which counts the lines in the list.

```
Asm Sample.a && Link Sample.a.o -o Sample.code ||
(Echo Failed; Beep)
```

This example begins by assembling Sample.a. If that operation succeeds, it links the object file; but if the assemble-and-link operation fails, it echoes the message "Failed," and beeps.

## Command continuation

You can continue a command onto the next line by typing `\` (Option-D) followed by a return. Both characters are discarded when the line is interpreted. The return must come *immediately* after the `\`, with no blanks or comments between them. (For more information about the `\` escape character, see "Quoting Special Characters" in this chapter.)

```
Echo This is all \
one command
```

Notice that the output appears on one line.

---

## Comments

The number sign (#) indicates a comment. Everything from the # to the end of the line is ignored. (Comments *always* end at the next return, even if the return is preceded by a  $\partial$ .)

```
Echo This is echoed. # This is not.
Echo parameters # comment ∂
 more parameters # another comment
```

---

## Simple versus structured commands

All of the commands introduced so far have been **simple commands**. Simple commands consist of a single keyword, followed by zero or more parameters. Simple commands are distinguished from **structured commands**—commands such as For and If, which let you control the order in which other commands are executed. For example,

```
For file In =.c; Count {file}; End
```

All of the structured commands are built-in, and usually have more than one keyword. The entire structured command is read before its execution begins.

Also see “Structured Commands” in this chapter.

---

---

## Running an application outside the Shell environment

You can run an application outside the MPW Shell environment by executing the program name just like any other command. For example,

```
ResEdit
```

The application is loaded and launched as if it had been started from the Finder. Any files specified as parameters are passed to the program via the application parameter handle, in Finder fashion. (See “Finder Information” in the Segment Loader chapter of *Inside Macintosh*.) The following option is available on the command line:

**-p file...** Tell the program to print the specified files.

For example,

```
MacPaint -p "HD:Screen 1" "HD:Screen 2"
```

This command tells the Shell to run MacPaint (assuming MacPaint is in a directory listed in the Shell variable {Commands}, and to print the files Screen 1 and Screen 2.

The Shell environment is saved when the application is launched and restored when the application terminates. (These actions are performed by the Suspend and Resume command files, described below.)

---

### Caution

Running an application from a command script terminates the script.

---

---

---

## Command scripts

You can create your own commands by writing text files of previously defined commands, called **scripts** or **command files**. You can execute such a file just like any other command within the Shell environment—the name of the file you created is the name of the new command.

For example,

```
Date
Echo Volumes.....
Volumes
Echo Current Directory.....
Directory
Echo Files.....
Files
```

If this text is on the screen, you can execute it by selecting it and pressing Enter. You could also save this text as a script so that it's always available. To save it under the name "Info", for example, you could first select the command text, and type the following command in another window:

```
Duplicate -d $ Info
```

You can now execute this series of commands by entering the command name Info. (Recall that the \$ character indicates the selection in the target window.)

You can pass parameters to a script just as you would to a predefined command, using the normal Shell syntax:

```
filename [parameters...]
```

Parameters can be referred to within the scripts by using the built-in variables {1}, {2},... {n}, explained below under "Parameters to Scripts."

- ❖ *Note:* As a matter of convenience, scripts (as well as applications and tools) are usually kept in directories that the Shell automatically searches when a leaf name is given for a command name. This convention allows you to invoke the command by using its leaf name instead of its full pathname. The Shell variable {Commands} contains a comma-separated list of directories to be searched; you can easily modify it to include additional directories.

---

---

## Special scripts

The scripts described in this section are provided with MPW. You can modify commands in each of these files to suit your needs.

---

### Important

Each of these scripts must be in the same directory as the MPW Shell, or in the System Folder.

---

---

## The Startup and UserStartup files

When you start up the Shell, commands are initially read from a file named Startup. The Shell executes the commands in Startup as if you had entered them interactively. The Startup file provided with MPW contains several default variable and alias definitions. You can modify the commands in Startup to suit your own needs; for instance, you can change the default pathnames to suit a special directory configuration.

Startup executes another script called UserStartup. It's recommended that you use this file for your own changes and additions to the startup sequence. You can redefine the variables defined in Startup, set and export any number of additional command-language variables, and define aliases and create menu items. Aliases and variables are fully described in the sections that follow.

---

## Suspend, Resume, and Quit

When you run an application from the Shell, commands are read from the file Suspend. When you quit the application and return to the Shell, commands are read from the file Resume. The Suspend and Resume files save state information about variable definitions, exports, aliases, and windows before running an application, and restore the state after returning to the Shell.

When you quit from the Shell, commands are read from the file Quit. The Shell executes these commands before closing any windows.

❖ *Note:* If you cancel from the Quit command, the Quit file will already have been executed.

Like Startup and UserStartup, these scripts run as if you had entered the commands interactively. You can modify them to suit any special requirements you might have.

---

---

## Command aliases

An **alias** is an alternate name for a command (and possibly some parameters). The Alias command is used to define aliases, and to display the list of aliases. If an alias has been defined, it will be recognized by the command interpreter and the corresponding definition will be substituted.

❖ *Note:* Variable substitution and alias substitution occur on the alias definition itself, after it has been substituted.

The following commands are used to define and undefine aliases:

|                           |                                                             |
|---------------------------|-------------------------------------------------------------|
| Alias <i>name word...</i> | <i>Name</i> becomes an alias for the list of words.         |
| Alias <i>name</i>         | Displays any alias definition associated with <i>name</i> . |
| Alias                     | Displays all alias definitions.                             |
| Unalias <i>name</i>       | Removes any alias definition associated with <i>name</i> .  |
| Unalias                   | Removes all alias definitions.                              |

Aliases are local to the script in which they are defined (and are globally available if they are defined in the Startup file or entered interactively). Aliases are automatically inherited from enclosing scripts, and may be redefined locally. However, aliases redefined locally will revert to their previous value when the script terminates.

See the Alias and Unalias commands in Part II for a complete specification of aliases and several examples.

---

## Executable error messages

The following alias is defined in the Startup file:

```
Alias File Target
```

That is, the word “File” is defined as an alias for the Target command, which opens a file as the target window. (See Chapter 6.) This alias is useful when a compiler returns an error message such as

```
Not a parameter name: counts
 File "Count.c" ; line 73
```

By placing the insertion point anywhere on the line or by selecting the entire line and pressing the Enter key, you’ll automatically open the specified file as the target window, find and select the offending line, and bring the window to the top. The command that the Shell actually executes is

```
Target "Count.c" ; Line 73
```

“Line” is a script, which automatically finds and selects a line by number and then brings the target window to the top.

---

---

## Variables

The Shell provides several predefined variables and allows you to declare any number of additional variables. Variables are used for

- shorthand notation
- providing status information
- local variables in scripts
- parameters to scripts and tools
- setting certain defaults for the MPW Shell

You can define or redefine variables with the Set command, and remove variable definitions with the Unset command. For example,

```
Set PFiles HD:MPW:PFiles:
```

This command defines a variable {PFiles} with the value “HD:MPW:PFiles:”.

Variables have strings as their values. You can reference them by using the notation {*name*}, where *name* is the name of the variable. When a command containing a variable {*name*} is executed, {*name*} is replaced with the current value of the variable. For example,

```
Files {PFiles}Src.p
```

In this example, {PFiles} is replaced with its definition before the command is executed.

A variable may form one or more words, or part of a word. If a variable is undefined, {*name*} is removed (that is, replaced with the null string).

Variable names are case insensitive, and can’t include the right brace character (}), for obvious reasons. It’s wise to avoid using any special characters in variable names—future extensions to the command language may assign special meanings to some of these characters.

❖ *Note:* For variables such as {Exit} and {CaseSensitive} that can be either “true” or “false,” the variable is considered “true” if it is set to anything other than zero or the null string (a string of length zero). The variable is considered “false” if it is set to zero, null, or undefined. The best way to set one of these variables is as follows:

```
Set Exit 1 # turn {exit} on
Set Exit 0 # turn {exit} off
```

(These values also apply to expressions that return a Boolean value, defined later in this chapter under “Structured Commands.”)

---

## Predefined variables

Table 5-2 lists the variables defined by the MPW Shell. These variables provide the status value returned by the last command, and the pathnames of several files and directories.

**Table 5-2**  
Variables defined by the Shell

---

|                  |                                                                                                                                                                                                                                                                                                                         |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| {Active}         | Full pathname of the current active window.                                                                                                                                                                                                                                                                             |
| {Aliases}        | Contains a list of all defined aliases with each name separated by a comma. The list contains only the names, not the definitions. Commando uses this variable with the built-in commands Alias and Unalias. Commando needs this variable in order to know the names of existing variables. {Aliases} must be exported. |
| {Boot}           | Volume name of the boot disk.                                                                                                                                                                                                                                                                                           |
| {Command}        | Full pathname of the last command executed. (For built-in commands, this is the name of the command.)                                                                                                                                                                                                                   |
| {ShellDirectory} | Full pathname of the directory that contains the MPW Shell.                                                                                                                                                                                                                                                             |
| {Status}         | Result of the last command executed. (A value of 0 means successful completion. Any other value is an error code: Typically, 1 means an error in parameters, and 2 means that the command failed.)                                                                                                                      |
| {SystemFolder}   | Full pathname of the directory that contains the System and Finder files.                                                                                                                                                                                                                                               |
| {Target}         | Full pathname of the target window (that is, the second window from the top—by default, this is the window where editing commands take effect).                                                                                                                                                                         |
| {Windows}        | This variable contains a list of the current windows with each name separated by a comma. Commando uses this list to allow redirection of output or input to or from existing windows. Commando needs this variable in order to know the names of the current windows. {Windows} must be exported.                      |
| {Worksheet}      | Full pathname of the Worksheet window.                                                                                                                                                                                                                                                                                  |



---

## Variables defined in the Startup file

Table 5-3 lists the variables that are defined in the Startup file (described in the “Special Scripts” section in this chapter). These variables define pathnames and default settings to the Shell, and are referenced by the Shell and by some of the MPW tools. You can change any of these definitions to suit your own needs.

❖ *Note:* Hierarchical file system (HFS) pathname conventions are described in Chapter 4.

**Table 5-3**  
Variables defined in the Startup file

---

**Variables referenced by the command interpreter**

{MPW} The volume or folder containing the Macintosh Programmer's Workshop. Initially set to "{Boot}MPW:". If your MPW directory is somewhere other than the root of the boot volume, modify the value of {MPW} in the Startup file.

{Commands} A list of the directories that the Shell searches when looking for a command to execute. Directories in the list are separated by commas. A single colon indicates the default directory. {Commands} is initially set to

```
., {MPW}Tools:, {MPW}Scripts:, {MPW}Applications:
```

—that is, the current directory, then HD:MPW:Tools, then HD:MPW:Scripts, and then HD:MPW:Applications.

**Variables referenced by the command interpreter**

{Exit} When {Exit} is set to a nonzero value, scripts terminate whenever a command returns a nonzero status. This nonzero status is returned as the status value of the script. (See the {Status} variable in Table 5-2.) {Exit} is initially set to 1.

{Echo} When {Echo} is set to a nonzero value, commands are written to diagnostic output after aliasing, variable substitution, command substitution, and filename generation, and just prior to execution. This capability is useful for watching the progress of a script and for debugging scripts—as the first line of your file, you would include the line

```
Set Echo 1
```

{Echo} is initially set to 0.

{Test} When {Test} is set to a nonzero value, the command interpreter executes built-in commands and scripts, but not tools or applications. {Test} is useful for checking the control flow in command files. (It's most useful if {Echo} is also nonzero.) {Test} is initially set to 0.

{Commando} This variable tells the Shell which command to execute when the ellipsis character (Option-semicolon) is present anywhere in a command line. The Startup file sets this variable to "Commando". This variable allows the development of similar tools whose output is to be executed by the Shell. If the variable is not set, then the ellipsis character is removed from the command line and normal execution proceeds. {Commando} must be exported if scripts are to use Commando.

(continued)

**Table 5-3 (continued)**  
Variables defined in the Startup file

---

**Variables referenced by the editor**

|                 |                                                                                                                                                                                                                                                                                                                                          |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| {CaseSensitive} | Any nonzero value specifies case-sensitive pattern matching. {CaseSensitive} is initially set to 0 (that is, false). You can also set {CaseSensitive} in the "Find and Replace" dialog boxes.. (See Chapter 3.)                                                                                                                          |
| {Tab}           | Default tab setting for new windows (initially 4). You can also set {Tab} in the "Format..." dialog available from the Edit menu.                                                                                                                                                                                                        |
| {WordSet}       | The set of characters that constitute a word to the editor (for Find and Replace menu commands, and for word selection by double-clicking). By default, {WordSet} is set to the characters a-z, A-Z, 0-9, and _ (underscore). If a character is not in the list, the editing commands regard it, like a blank, as a break between words. |
| {PrintOptions}  | Options used by the Print Window and Print Selection menu commands. Initially set to "-h". (The -h option prints pages with headers. For more information on possible print options, see the Print command in Part II.)                                                                                                                  |

**Pathnames for libraries and include files**

|               |                                                                                                                                 |
|---------------|---------------------------------------------------------------------------------------------------------------------------------|
| {AIncludes}   | The directories to search for assembly-language include files, referenced by the Assembler. Initially set to "{MPW}AIncludes:". |
| {CIncludes}   | The directories to search for C include files, referenced by the C Compiler. Initially set to "{MPW}CIncludes:".                |
| {CLibraries}  | The directory that contains C library files. Initially set to "{MPW}CLibraries:".                                               |
| {Libraries}   | The directory that contains shared library files. Initially set to "{MPW}Libraries:".                                           |
| {PInterfaces} | The directories to search for Pascal interface files, referenced by the Pascal Compiler. Initially set to "{MPW}PInterfaces:".  |
| {PLibraries}  | The directory that contains Pascal library files. Initially set to "{MPW}PLibraries:".                                          |
| {RIncludes}   | The directory that contains Resource Compiler (Rez) include files. Initially set to "{MPW}RIncludes:".                          |

---

## Parameters to scripts

When a script is executed, its parameters automatically set the value of certain Shell variables. These variables are explained in Table 5-4.

**Table 5-4**  
Parameters to scripts

---

|                          |                                                                                                                                           |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| {0}                      | Name of the currently executing script.                                                                                                   |
| {1}, {2},...{ <i>n</i> } | First, second, and <i>n</i> th parameter passed to the current script. (These values are null for commands entered interactively.)        |
| {#}                      | Number of parameters (excluding the command name).                                                                                        |
| {Parameters}             | Equivalent to {1} {2} ...{ <i>n</i> }.                                                                                                    |
| {"Parameters"}           | Equivalent to "{1}" "{2}" ..."{ <i>n</i> }". This form should be used if the parameters could contain blanks or other special characters. |

The {Parameters} variable is especially useful when the number of parameters is unknown. The quoted forms, such as "{1}" or {"Parameters"}, are usually preferable to the unquoted forms because, after variable substitution, {1}, {2}, and so on could contain blanks or other special characters. For example, consider the Line script (which is useful with error messages as explained earlier in this chapter under "Executable Error Messages"):

```
Find "{1}" "{Target}" # Find line n in the target window.
Open "{Target}" # Make the target window the active d
 # (top) window.
```

This script takes one parameter, a line number. Parameter {1} is quoted to handle the case where Line is called without any parameters. In this case the value of {1} is the null string, and without the quotes the {1} would completely disappear, leaving the name of the target window as the only parameter to Find. The quotes ensure that at least a null string is sent to Find as its first parameter—this is essential, because the window name must be the second parameter. Also notice that the {Target} variable is quoted, because it's a filename that might contain blanks or other special characters. (For more information on quoting rules, see "Quoting Special Characters" later in this chapter.)

---

## Defining and redefining variables

The following commands are used to define and modify variables:

|                       |                                                                                                                                             |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| Set <i>name value</i> | Assigns the string <i>value</i> to variable <i>name</i> .                                                                                   |
| Set <i>name</i>       | Writes the value of variable <i>name</i> to standard output.                                                                                |
| Set                   | Writes a list of all variables and their values to standard output.                                                                         |
| Unset <i>name</i>     | Removes the definition of variable <i>name</i> .                                                                                            |
| Unset                 | Removes the definition of <i>all</i> variables in the current scope. (For an explanation of the scope of a variable, see the next section.) |

---

### Caution

Removing all variables in the outermost scope can have serious consequences. For example, the Shell uses the variable {Commands} to locate MPW tools and other commands. The Assembler and Compilers use other variables to help locate include files. Some variables, such as {Boot}, cannot be reinitialized without restarting MPW.

---

Defining a variable and making it available for use by scripts and programs involves two separate steps:

1. You can define a variable with the Set command. Note that variables are local to the script in which they are defined—a variable definition ceases to exist when its command file terminates.
2. You can pass a variable to scripts and tools with the Export command. After you export a variable, nested scripts can reference that variable, and may override its value locally—but any redefinition is strictly local, and terminates when the script terminates. It's impossible to affect the value of a variable in an enclosing script. (See Figure 5-1.)

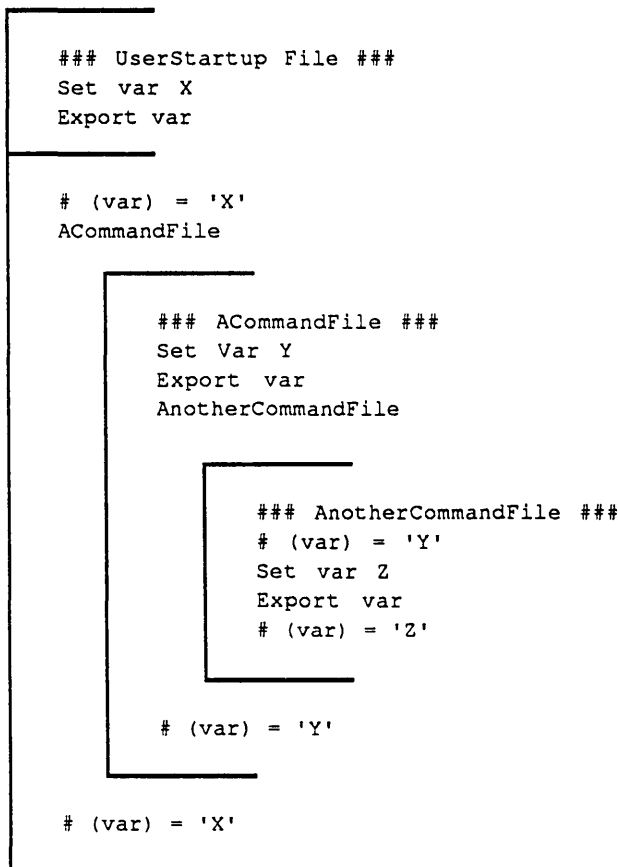
---

## Exporting variables

The Export command makes variables available to scripts and tools:

|                         |                                                                  |
|-------------------------|------------------------------------------------------------------|
| Export <i>name...</i>   | Exports the named variables.                                     |
| Export                  | Writes the list of exported variables to standard output.        |
| Unexport <i>name...</i> | Removes specified variables from the list of exported variables. |
| Unexport                | Writes the list of unexported variables to standard output.      |

You can define a variable globally by setting its value in the Startup file and exporting it. Figure 5-1 illustrates how Export works.



**Figure 5-1**  
Trafficking in variables

- ❖ *Note:* You can use the Execute command to execute a script without creating a new scope for variables, exports, and aliases. The Shell “executes” the Startup, Suspend, Resume, and Quit scripts, and Startup uses Execute to run the UserStartup script. For more details about Execute, see Part II.

---

---

## Command substitution

Command substitution causes a command to be replaced by its output. You can specify command substitution by enclosing one or more commands in back quotes (``...``). The backquote key is located at the upper-left corner of the original Macintosh keyboard; it is located near the space bar of the newer keyboards. When the command is executed, the standard output of the enclosed commands replaces the ``...``. Command substitution can form part of a word, a complete word, or several words. Command substitution is not done within “hard” quotes (that is, the standard single quotes `'...'`).

- ❖ *Note:* If the standard output of the enclosed commands contains return characters, the returns are replaced by blanks. If the output ends with a return, this return is discarded.

For example, the command

```
Echo The date is `Date`
```

echoes the parameters, replacing the `Date` command with its output, as follows:

```
The date is Wednesday, October 22, 1987 10:40:00 PM
```

The following example duplicates the files whose names are output by the `Files` command:

```
Duplicate `Files -t MPST MyDisk:` "{MPW}Tools"
```

``Files -t MPST MyDisk:`` is replaced with a string of filenames of type MPST (that is, MPW tools) before the `Duplicate` command is executed; these files are then copied to the folder `{MPW}Tools`. This command is useful because the `Files` command allows you to specify files with a certain type or creator, which you can't do with wildcard operators.

---

---

## Quoting special characters

There are numerous characters that have special meanings to the MPW Shell. Normally, the Shell performs the action indicated by the special character—but you can disable a character's special meaning (that is, include it as a literal character) by quoting it. You commonly need quotes when specifying filenames that contain blanks or other special characters, or when searching for the literal occurrence of a special character. See also “Pattern Matching” in Chapter 6.

Table 5-5 lists all of the special symbols recognized by the Shell.

**Table 5-5**  
Special characters and words

| Character | Meaning                                                                                                                  | Where described                                |
|-----------|--------------------------------------------------------------------------------------------------------------------------|------------------------------------------------|
| Space     | Separates words                                                                                                          | "Structure of a Command"                       |
| Tab       | Separates words                                                                                                          |                                                |
| Return    | Separates commands.                                                                                                      | "Structure of a Command"<br>(Table 5-1)        |
| ;         | Separates commands.                                                                                                      |                                                |
|           | Separates commands, piping output to input.                                                                              |                                                |
| &&        | Separates commands, executing the second if the first succeeds.                                                          |                                                |
|           | Separates commands, executing the second if the first fails.                                                             |                                                |
| (...)     | Command grouping; grouping in filename generation.                                                                       |                                                |
| ...       | Invokes Commando.<br>The ellipsis must be obtained by using the Option-semicolon key sequence; <i>not</i> three periods. | "Invoking Commando Dialogs" in Chapter 4       |
| #         | Comments.                                                                                                                | "Structure of a Command"                       |
| ∂         | Escape character: quotes the subsequent character.                                                                       | In this section (Table 5-7)                    |
| '...'     | Quotes all special characters, except "..."                                                                              |                                                |
| "..."     | Quotes all special characters, except ∂, {, " and `                                                                      |                                                |
| /.../     | Quotes all special characters, except ∂, {, ` and /                                                                      |                                                |
| \...\     | Quotes all special characters, except ∂, {, ` and \                                                                      |                                                |
| {...}     | Variable substitution.                                                                                                   | "Variables"                                    |
| `...`     | Command substitution.                                                                                                    | "Command Substitution"                         |
| ?         | Matches a single character in filename generation.                                                                       | "Filename Generation" in this chapter and      |
| =         | Matches any string in filename generation.                                                                               | "Pattern Matching" in Chapter 6                |
| [...]     | Character list in filename generation.                                                                                   |                                                |
| *         | Zero or more repetitions in filename generation.                                                                         |                                                |
| +         | One or more repetitions in filename generation.                                                                          |                                                |
| « »       | Specified number of repetitions in filename generation.                                                                  |                                                |
| <         | Input file specification.                                                                                                | "Redirecting Input and Output"<br>(Table 5-11) |
| >         | Output file specification.                                                                                               |                                                |
| >>        | Output file specification (append).                                                                                      |                                                |
| ≥         | Diagnostic file specification.                                                                                           |                                                |
| ≥≥        | Diagnostic file specification (append).                                                                                  |                                                |

You can literalize a character by preceding it with the Shell escape character, `\` (Option-D), or by including it within the quote symbols `'...'`, `"..."`, `/.../`, or `\...\`. The escape character, `\`, quotes a single character only; the other quote symbols may be used to quote part or all of a word. These symbols are described in Table 5-6.

**Table 5-6**  
Quotes

---

|                                          |                                                                                                                                                                                                                                                                                                                                                                                                 |
|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>'...'</code>                       | “Hard quotes”: Take the enclosed string literally—no substitutions occur. The quotes are removed before execution.                                                                                                                                                                                                                                                                              |
| <code>"..."</code>                       | “Soft quotes”: Take the enclosed string literally. <code>\c</code> , variable substitutions, and command substitutions occur. The quotes are removed before execution.                                                                                                                                                                                                                          |
| <code>/.../</code> or <code>\...\</code> | Regular expression quotes: Normally used to enclose regular expressions. Take the entire string literally, including the quote characters—the <code>/</code> or <code>\</code> characters are <i>not</i> removed. Variable substitutions and command substitutions occur. <code>'...'</code> , <code>"..."</code> , and <code>\</code> have their usual meanings—however, they are not removed. |

Single quotes, double quotes, and `\` are removed before parameters are passed to programs (unless they are themselves enclosed in quotes). For example, here is how you could define an AddMenu that compiles a C program in the active window:

```
Wrong: AddMenu Extras "C Compile" C "{Active}"
Right: AddMenu Extras "C Compile" 'C "{Active}"'
```

The first example 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*). The second example is correct—when the AddMenu command is executed, the single quotes defeat variable expansion; they are then stripped off before the item is actually added. The double quotes remain, in case the pathname of the active window happens to contain any special characters.

❖ *Note:* When quoting spaces (as in filenames), you'll usually use double quotes (soft quotes), to permit variable and command substitution.

Slashes (or backslashes) are used to pass regular expressions as parameters to commands, without filename expansion occurring. For example,

```
Search /proc=/ Sample.p
```

This command searches the file `Sample.p` for any string beginning with the characters `"proc"`. (See “Pattern Matching” in Chapter 6 and the description of the Search command in Part II.)

**Table 5-7**  
Special escape conventions

---

|                      |                                                                                                                                            |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <code>\c</code>      | Escape character: Take the single character <code>c</code> literally. The four escape conventions that follow are exceptions to this rule. |
| <code>\Return</code> | <code>\Return</code> is discarded, allowing you to continue a command onto the next line.                                                  |
| <code>\n</code>      | Inserts a return character.                                                                                                                |
| <code>\t</code>      | Inserts a tab character.                                                                                                                   |
| <code>\f</code>      | Inserts a form feed character.                                                                                                             |

---

---

## How commands are interpreted

When you send text to the command interpreter (by pressing the Enter key or the equivalent), the following sequence of steps is performed:

1. *Alias substitution.*
2. *Evaluation of control constructs.* (This means that control constructs can't be produced by command substitution but can have aliases.)
3. *Variable substitution, command substitution.* All variables (unquoted or quoted with "...", /.../, or \...\ ) are replaced with their value. All commands enclosed in `...` (unquoted or quoted with "...", /.../, or \...\ ) are replaced with their output. If the ellipsis is found, Commando is executed and the command is replaced by the output of Commando.
4. *Blank interpretation.* After variables and commands have been substituted, the command text is divided into individual words separated by blanks. A blank is an unquoted space or tab.

*Note:* The following symbols are normally considered separate words, whether or not they are set off by blanks:

; | || && ( ) < > >> ≥ ≥> ...

Within expressions (used with If and Evaluate), all operators are considered separate words, unless they are quoted—see “Structured Commands” in this chapter.

5. *Filename generation.* A word that contains any of the unquoted characters ?, =, [, \*, +, or « after variable substitution is considered a filename pattern. The word is replaced with an alphabetically sorted list of the filenames that match the pattern. (If no filename is found that matches the pattern, an error results.)
6. *Input/output redirection.* Because this step is performed last, variable substitution, command substitution, and filename generation can all be used to form the filenames used in I/O redirection.
7. *Execution.*

Any part of this process can be suppressed by using quotes as described in the previous section. Remaining single and double quotes are removed prior to execution.



---

---

## Structured commands

Structured commands (listed in Table 5-8) override the normal sequential execution of commands. They can be used interactively and within scripts. They may be nested arbitrarily deeply (subject to a limitation on stack space). The entire structured command is read before execution begins. All structured commands are built into the MPW Shell.

---

### Caution

After the Shell "executes" an opening parenthesis or the opening word of a Begin, If, For, or Loop command, it will not execute any subsequent commands until a matching closing parenthesis or End word is encountered. While it is waiting for the end of the command, the status panel of the Worksheet window will contain the left parenthesis character, (, or the command name. You can abort the entire structured command by typing Command-period.

---

The status value for a structured command is the status of the last command executed within the structured command (except for the Exit command, which lets you set your own status value).

❖ *Note:* Expressions (used in If, Break, Continue, and Exit) are defined in the section following the table.

**Table 5-8**  
Structured commands

---

|                       |                                                                                                                                                                                                                                                                                                                                                                                                  |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ( <i>command...</i> ) | Parentheses are used to group commands for conditional execution, pipe specifications, and input/output specifications.                                                                                                                                                                                                                                                                          |
| Begin...End           | Begin<br><i>command...</i><br>End<br><br>Like parentheses, Begin and End group commands for conditional execution, pipe specifications, and input/output specifications.                                                                                                                                                                                                                         |
| If...                 | If <i>expression</i><br><i>command...</i><br>[ Else If <i>expression</i><br><i>command...</i> ] ...<br>[ Else<br><i>command...</i> ]<br>End<br><br>If... executes the commands following the first <i>expression</i> whose value is true (that is, nonzero and non-null). At most one of the lists of commands is executed. If none of the commands is executed, If returns a status value of 0. |
| For...                | For <i>name</i> In <i>word...</i><br><i>command...</i><br>End<br><br>For... executes the enclosed commands once for each word from the "In <i>word...</i> " list. For each iteration, a variable of the form { <i>name</i> } represents the current value from the <i>word...</i> list. (See the examples below.)                                                                                |

(continued)

**Table 5-8 (continued)**  
Structured commands

|            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Loop...End | <p>Loop<br/>    <i>command...</i><br/>End</p> <p>This command repeatedly executes the enclosed commands.<br/>The Break command is used to terminate the loop.</p>                                                                                                                                                                                                                                                                                                                                   |
| Break      | <p>Break [ If <i>expression</i> ]</p> <p>Break terminates execution of the immediately enclosing For or Loop. If the expression is present, the loop is terminated only if the expression evaluates to true (nonzero and non-null).</p>                                                                                                                                                                                                                                                             |
| Continue   | <p>Continue [ If <i>expression</i> ]</p> <p>Continue terminates this iteration of the immediately enclosing For or Loop and continues with the next iteration. If the expression is present, the Continue is executed only if the expression evaluates to true (nonzero and non-null).</p>                                                                                                                                                                                                          |
| Exit       | <p>Exit [ <i>number</i> ] [ If <i>expression</i> ]</p> <p>Exit terminates execution of the script in which it appears. If <i>number</i> is present, it is returned as the status value of the script; otherwise, the status of the last command executed is returned. If the expression is present, the script is terminated only if the expression evaluates to true (nonzero and non-null). (You can also use Exit interactively, to terminate execution of all previously entered commands.)</p> |

The return characters in the command definitions above are significant; a return must appear at the end of each line as shown above, or be replaced by a semicolon (;).

The following keywords are recognized when they appear unquoted as the first word of a command:

```
Begin For If Else Loop End Break Continue Exit
```

The keyword "In" is recognized when it appears unquoted following For; the keyword "If" is recognized when unquoted following Else, Break, Continue, and Exit. These keywords are not considered special in other contexts and need not be quoted.

❖ *Note:* These keywords can't be produced as a result of variable substitution or command substitution.

You can apply conditional execution (&& and ||), pipe specifications (|), and input/output specifications (<, >, >>, ≥, and ≥≥) to entire structured commands (that is, to Begin...End, If...Else...End, For...End, and Loop...End, and to commands within parentheses). The operator should appear following the End or closing parenthesis. For example, you can collect the output of a series of commands and redirect it as follows:

```
Begin
 Echo Good day
 Echo Sunshine
End > OutputFile
```

Input/output specifications are discussed later in this chapter. Each of the structured commands is described in detail in Part II.

---

## Control loops

The For and Loop commands are used for looping.

The For...End command executes the enclosed commands once for each word in the "In *word*..." list. The current *word* is assigned to variable *name*, so you can reference the current word by using the Shell variable notation, *{name}*. For example,

```
For File In *.c
 C "{File}" ; Echo "{File}" compiled.
End
```

The Loop command provides unconditional looping—you'll need to use the Break or Exit commands to terminate the loop. You can use the Continue command to continue with the next iteration.

For example, the script below runs a command several times, once for each parameter:

```
Repeat - Repeat a command for several parameters
#
Repeat command parameter...
#
Repeat command once for each parameter in the parameter
list. Options can be specified by including them in
quotes with the command name.
#
Set cmd "{1}"
Loop
 Shift
 Break If {#} == 0
 {cmd} "{1}"
End
```

In this example, the Shift command (explained in the next section) is used to step through the parameters, and the Break command ends the loop when all the parameters have been used. Using the script Repeat, you could compile several C programs, with progress information, using the command

```
Repeat 'C -p' Sample.c Count.c Memory.c
```

Repeat might also be used to set the font and font size for all the open windows:

```
Repeat 'Font Courier 10' `Windows`
```

---

## Processing command parameters

In addition to the commands introduced in Table 5-8, there are several other commands that are highly useful in scripts. The following commands are used to display or modify parameters:

|                                     |                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Echo [ <i>parameters...</i> ]       | Writes its parameters, separated by blanks and terminated by a return, to standard output.                                                                                                                                                                                                                                                                                             |
| Parameters [ <i>parameters...</i> ] | Writes its parameters, including its name, to standard output. One parameter is written per line, preceded by the parameter number in braces and a space. A return is written following the last parameter.                                                                                                                                                                            |
| Shift [ <i>number</i> ]             | Renames the parameters by subtracting <i>number</i> from the parameter number; that is, parameters <i>number</i> +1, <i>number</i> +2, and so on are renamed 1, 2, and so on. If <i>number</i> is not specified, the default value is 1. The variables {1}, {2}...{n}, {#}, {Parameters}, and {"Parameters"} are all affected. Shift does not affect parameter {0} (the command name). |

Echo and Parameters are useful for checking how your parameters will behave before actually passing them to a command (for example, to check how your quotes are working out). For example,

```
Parameters "-"
```

For an example of how the various structured commands can work together, see "Sample Scripts" at the end of this chapter.

---

## Expressions

Expressions are used in the If, Break, Continue, and Exit commands. They're also used in the Evaluate command, which returns the result of an expression.

Table 5-9 lists the expression operators in order of decreasing precedence. Some operators have more than one representation; these equivalent symbols are listed on a single line. Groupings indicate operators of the same order of precedence.

**Table 5-9**  
Expression operators in order of decreasing precedence

| Operator         | Operation                              |
|------------------|----------------------------------------|
| 1. <i>(expr)</i> | Expression grouping                    |
| 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 pattern (regular expression)     |
| !~               | Not equal pattern (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 decimal number. Relational operators return the value 1 when the relation is true and the value 0 when the relation is false.

**Logical operators:** The logical operators `!`, `NOT`, `¬`, `&&`, `AND`, `||`, and `OR` interpret operands of value 0 or null as false; and nonzero, non-null operands as true.

**Numbers:** Numbers may be either decimal or hexadecimal integers representable by a 32-bit signed value. Hexadecimal numbers begin with either `$` or `0x`. Every expression is computed as a 32-bit signed value. Overflows are ignored.

**String operators:** The operators `==`, `!=`, `==~`, and `!~` compare their operands as strings. All others operate on numbers.

**Comparing text patterns:** The `==~` (equal pattern) and `!~` (not equal pattern) operators are like `==` and `!=` (which compare two strings), except that `==~` and `!~` are used for comparing a string with a text pattern. The right-hand side is a regular expression against which the left-hand operand is matched. For example:

```
If "{1}" !~ /.{acp}/
 Echo Filename must end with .a, c, or .p
End
```

❖ *Note:* The regular expression in the above example must be enclosed in the regular expression quotes, `/.{acp}/`. See Chapter 6 for more information about regular expression syntax.

If the regular expression contains the tagging operator `@`, then, as a side effect of evaluating the expression, Shell variables of the form `{@n}` containing the matched substrings are created for each tag operator in the expression. (For an example, see the implementation of a wildcard rename command, under the description of the `Rename` command in Part II.)

**Use of special characters:** Within expressions in the `If`, `Break`, `Continue`, `Exit`, and `Evaluate` commands, the following Shell operations are disabled:

- filename generation
- conditional execution (`||` and `&&`)
- pipe specifications (`|`)
- input/output specifications (`>`, `>>`, `≥`, `≥≥`, and `<`)

This allows the use of many expression operators that would otherwise have to be quoted. In the case of `If` commands, the conditional execution or I/O specification should come after the `End` word. For other commands that contain expressions, you can specify conditional execution or I/O redirection by enclosing the command in parentheses. For example,

```
(Evaluate {1} + {2}) ≥ Errors
```

---

---

## Filename generation

After variables have been substituted, an unquoted word that contains any of the characters

? ≈ [ \* + «

is considered a filename pattern. The word is replaced with an alphabetically sorted list of filenames that match the pattern. An error is returned if no filename is found that matches the pattern.

You can specify a group of file (or window) names with the “wildcard” notation given in Table 5-10.

**Table 5-10**  
Filename generation operators

---

|                           |                                                                                           |
|---------------------------|-------------------------------------------------------------------------------------------|
| ?                         | Matches any single character (except return or colon).                                    |
| ≈                         | Matches any string of zero or more characters (except return or colon).                   |
| [ <i>characterList</i> ]  | Matches any character in the list.                                                        |
| [¬ <i>characterList</i> ] | Matches any character not in the list.                                                    |
| *                         | 0 or more repetitions of the preceding character or character list (?* is the same as ≈). |
| +                         | 1 or more repetitions of the preceding character or character list.                       |
| «number of repetitions»   | Specified number of repetitions of the preceding character or character list.             |

*Note:* The pattern matching is case insensitive.

*Note also:* The pathname separator (:) must appear explicitly in the pattern—the : character will never be substituted for ?, ≈, or [...].

These special characters are the same **regular expression operators** used in editing commands. For a complete discussion of regular expressions, see Chapter 6.

Naturally, you need to be careful with these wildcard operators. The Parameters and Echo commands are very useful for double-checking which filenames a command will generate. For example, before giving the command

```
Delete ≈.c.o
```

you might want to run the command

```
Parameters ≈.c.o
```

This command lists your “.c.o” files to standard output so that you can make sure you really want to delete them all.

❖ *Note:* Wildcard characters only generate names that match existing filenames; they do not create new files. For example, the following attempt to rename files *will not work*:

```
Rename ≈.obj ≈.o
```

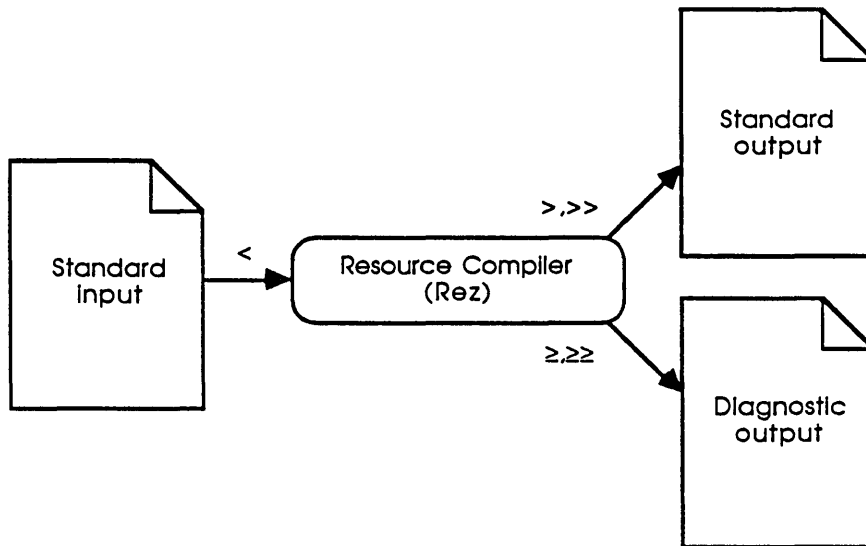
An example of how to perform a wildcard rename can be found under the description of the Rename command in Part II.

---

---

## Redirecting input and output

All built-in commands, scripts, and tools are provided with three open files: standard input, standard output, and diagnostic output (Figure 5-2). By default, standard input comes from the console (the window where the command is executed); standard output and diagnostics are returned to the console, immediately following the command.



**Figure 5-2**  
Standard input and output

You can override these default assignments with the `<`, `>`, `>>`, `≥`, and `≥≥` symbols described in Table 5-11. Note that input and output specifications are interpreted by the Shell; they are not passed to commands as parameters. Parentheses (or the `Begin` and `End` commands) can be used to group commands for input/output specifications.

**Table 5-11**  
I/O redirection

---

|                            |                                                                                                           |
|----------------------------|-----------------------------------------------------------------------------------------------------------|
| <code>&lt; name</code>     | Standard input is taken from <i>name</i> .                                                                |
| <code>&gt; name</code>     | Standard output replaces the contents of <i>name</i> . File <i>name</i> is created if it doesn't exist.   |
| <code>&gt;&gt; name</code> | Standard output is appended to <i>name</i> . File <i>name</i> is created if it doesn't exist.             |
| <code>≥ name</code>        | Diagnostic output replaces the contents of <i>name</i> . File <i>name</i> is created if it doesn't exist. |
| <code>≥≥ name</code>       | Diagnostic output is appended to <i>name</i> . File <i>name</i> is created if it doesn't exist.           |



Files and windows are treated identically—when given a name, the system looks first for an open window. Input and output can also be applied to selections:

- `&` indicates the current selection (in the target window).
- `name.&` indicates the current selection in window *name*.

From the point of view of a command running within the Shell environment, input always comes from the standard input file and output goes to the standard output file. The command doesn't need to know whether standard input happens to be text from a file, from a window, a selection, or typed in from the keyboard. For example, in the statement

```
Program > OutputFile
```

the string "`> OutputFile`" is interpreted by the Shell and is not passed as a parameter to the command—this process is completely invisible to the command.

I/O specifications also apply to scripts. The standard input, standard output, and diagnostic output files provided to a script become the defaults for commands in the file.

In addition to the sections later in this chapter, you'll find more on input and output in "Standard I/O Channels" in Chapter 13.

---

## Standard input

By default, standard input is supplied by typing text and pressing Enter, or by selecting text that is already on the screen and pressing Enter. You can redirect standard input with the `<` operator. Note, however, that most commands that read standard input also accept a filename parameter. For example, the following two commands have the same result:

```
Catenate < Sample.c
Catenate Sample.c
```

The `Alert` command reads from standard input if no message is supplied as a parameter to the command, but `Alert` doesn't accept filenames as parameters. Thus input redirection is the only way to cause `Alert` to read input from a file.

```
Alert Errors # Display Alert box containing the word Errors
Alert < Errors # Display Alert box containing the contents
of the file Errors.
```

Many commands, including the Assembler and Compilers, optionally read standard input to allow input to be read from a pipe (`|`) or entered interactively, as explained in the next section.

## Terminating input with Command-Enter

Many commands read from standard input if no filename is specified. For example, if you execute the command

```
Asm
```

the Assembler will begin reading from standard input—that is, you can enter text to it, and it will process each line as you enter it.

You can repeatedly enter text to a program that reads standard input, by typing or selecting text and pressing Enter. You indicate end-of-file by holding down the Command key and pressing Enter. For example, after you execute the command

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

the Catenate command will be running (its name will appear on the status panel at the bottom of the window). You can now enter data from the keyboard or select and enter text from various windows, and all of it will be concatenated to the Worksheet window. Command-Enter indicates end-of-file and terminates the command.

---

## Standard output

By default, **standard output** appears in the window in which the command was executed (that is, the console), immediately following the command. When commands are executed from menus, standard output appears following the selection in the active window. You can redirect standard output with the > and >> operators. For example,

```
Catenate File1 File2 > CombinedFile
```

The Catenate command concatenates File2 to File1—but instead of appearing in the active window, output is sent to the file named CombinedFile. If window CombinedFile is open on the desktop, its contents are overwritten. Otherwise, file CombinedFile is replaced (or created if it doesn't exist).

The >> operator appends standard output to the end of a selection, window, or file. If the named file doesn't exist, a new file is created. For example,

```
Catenate $ >> AFile
```

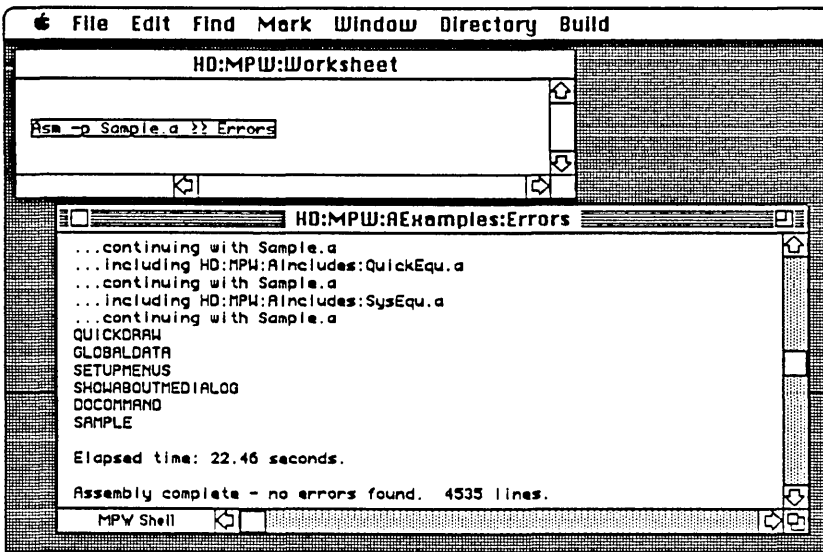
appends the contents of the current selection in the target window to AFile. (If the command was entered in the active window, the current selection is the selection in the target window.) You can also specify a selection in a named window:

```
Catenate Sample.c.$ >> AFile
```

---

## Diagnostic output

By default, a command's diagnostic output also appears immediately after the command, interleaved with standard output. The diagnostic output of commands executed from menus appears following the selection in the active window. You can redirect diagnostic output exactly as you redirect standard output, except that you use the operators  $\geq$  and  $\geq\geq$  in place of  $>$  and  $>>$ . You may find it useful to have all error reporting appear in a separate window set aside for that task. For example, in Figure 5-3, the Assembler has been run, and error and progress information has been appended to a window called "Errors".



**Figure 5-3**  
Redirecting diagnostic output

---

## Pseudo-filenames

**Pseudo-filenames** are a set of device names that you can use in place of filenames, but that have no disk files associated with them. Any command can open a pseudo-filename as a file. These device names are most commonly used for I/O redirection.

Table 5-12 shows the available pseudo-filenames.

**Table 5-12**  
Pseudo-filenames

---

|             |                                                                                                                                                                                                               |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Dev:Console | Always refers to the current console device. The console is the default source of input and the default destination of output—that is, the active window where a command is entered and its output displayed. |
| Dev:Null    | Null device. If you read from Dev:Null, it immediately returns end-of-file. If you write to Dev:Null, output is thrown away.                                                                                  |
| Dev:StdIn   | Standard input.                                                                                                                                                                                               |
| Dev:StdOut  | Standard output.                                                                                                                                                                                              |
| Dev:StdErr  | Diagnostic output.                                                                                                                                                                                            |

The last three names, StdIn, StdOut, and StdErr, are used to explicitly represent input and output. You can use these specifications, for example, to send a command's output and diagnostics to the same file:

```
Search /NULL/ *.c > Found ≥ Dev:StdOut
```

Because the Shell opens standard input, standard output, and diagnostic output in the order they appear, file Found is opened first, then diagnostic output is redirected to the same file. The following command has the same effect:

```
Search /NULL/ *.c ≥ Found > Dev:StdErr
```

However, if the filename and pseudo-filename specifications are simply reversed, the result is quite different:

```
Search /NULL/ *.c ≥ Dev:StdOut > Found
```

This command redirects diagnostic output to the previous standard output (probably the active window), then redirects output to file Found.

Pseudo-filenames are especially useful in a script when you want to do something like sending standard output to the diagnostic output. Here are some examples:

```
Echo "An error message." >> Dev:StdErr
```

```
Echo "HELP !" >> Dev:Console
```

Dev:Null is useful in scripts when you want to throw away diagnostic output. For example:

```
Eject 1 ≥ Dev:Null
```

This command ejects the disk in drive 1; if no disk is in drive 1, the script continues to run silently. (Note that you would also need to set {Exit} to 0—see “Variables” earlier in this chapter.)

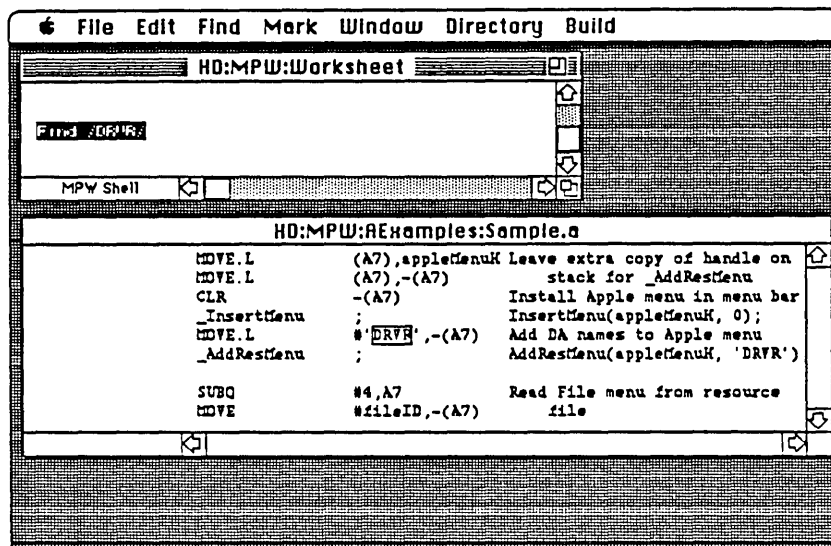
---

---

## Editing with the command language

Almost all menu functions have equivalents in the command language. In most respects, there is no difference between the menu items and their command-language equivalents. The primary difference is that with the command language, you enter commands in the active (frontmost) window, while the editing command acts on a selection in another window. You can explicitly name a window as a parameter to the command. If you don't specify a window, the command acts on the target window.

For example, to use command-language techniques to edit the file Sample.a, you must first open that file, and then click on another window, such as the Worksheet window, to make it the active window. You enter your commands in the active window, as shown in Figure 5-4. When you select text in the active window, it's highlighted in the normal Macintosh fashion. In other windows, selected text is indicated by dim highlighting (outlining), as shown in the target window in Figure 5-4.



**Figure 5-4**  
Text highlighted in the active window and target window

Editing commands generally act on a selection. (The Find command simply creates a selection—"DRVR" in this example.)

The `$` metacharacter (Option-6) is the current selection character—it signifies the current selection in a window. For example, the following command erases from the current selection or insertion point in the target window to the end of the window:

```
Clear $:∞
```

The infinity character, `∞` (Option-5), is a selection operator that indicates the end of a window, as described in Chapter 6. For interactive editing, press Command-Delete to clear to the end of a file.

---

---

## Defining your own menu commands

The AddMenu and DeleteMenu commands are for adding and deleting menu items. The AddMenu command takes three parameters: the menu name, the item name, and the command text. For example,

```
AddMenu Find 'Top of Window/U' 'Find • "{Active}"'
```

This command adds a “Top of Window” item to the Find menu, with the keyboard equivalent Command-U. When you select the menu item, the corresponding commands are executed. (The Top of Window item moves the insertion point to the top of the active window.)

Invoking a user-defined menu item is the same as entering the command text from a window—variable substitution and command substitution are performed normally. Note, however, that the text of the menu command is processed twice—once when the AddMenu command itself is executed, and again whenever the menu item is executed. This means that you have to be especially careful in your use of quotes. The mysteries of quoting are explained earlier in this chapter in “Quoting Special Characters,” together with further AddMenu examples. You should also pay particular attention to the section “How Commands Are Interpreted.” For further information, and more examples, see the AddMenu command in Part II.

---

---

## Sample scripts

The following examples use most of the Shell’s features to illustrate how you can extend the MPW Shell with your own commands.

---

### “AddMenuAsGroup”

The following script adds an extra feature to the AddMenu command:

```
AddMenuAsGroup - AddMenu, grouping user defined menu items:
#
AddMenuAsGroup [menuName [itemName [command]]]
#
AddMenuAsGroup duplicates the functionality of the AddMenu
command, adding a disabled divider before the first user-
defined menu items in the File, Edit, and Find menus.
#
Unalias
Set Exit 0
Set CaseSensitive 0
If ({#} == 3) AND ("{$1}" =~ /File/ OR "{$1}" =~ /Edit/ ∅
OR "{$1}" =~ /Find/)
 If `AddMenu "{$1}"` == "" # If this is the first addition
 # in {1},
 AddMenu "{$1}" "(-" "" # add the group divider
 End
End
AddMenu {"Parameters"}
```

When adding menu items to the predefined menus, it's nice to add a disabled dotted line item to separate the new menu items from the original ones. The script above automatically adds the separator before the first new item in the File, Edit, and Find menus, the only predefined menus that can be modified by using AddMenu. If you put this script in a file named AddMenuAsGroup, the following alias will override the built-in AddMenu command:

```
Alias AddMenu AddMenuAsGroup
```

---

## "CC"

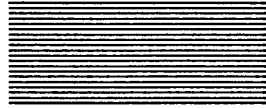
The following script extends the C command by making it possible to compile a number of specified files:

```
CC - Compile a list of files with the C compiler
#
CC [options...] [file...]
#
Note that the options and the files may be intermixed, and
that all options apply to all the files. The individual C
commands are echoed to diagnostic output as they are executed.
#

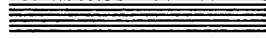
Unalias
Set Exit 0
Set CaseSensitive 0
Set options ""
Set files ""
Set exitStatus 0
Loop
 Break If (#) == 0
 If "{1}" =~ /-[diosu]/ # options with a parameter
 Set options "{options} '{1}' '{2}'"
 Shift 2
 Else If "{1}" =~ /-=/ # other options
 Set options "{options} '{1}'"
 Shift 1
 Else
 Set files "{files} '{1}'"
 Shift 1
 End
End
For i in {files}
 C {options} "{i}" || Set exitStatus 1
End
Exit {exitStatus}
```







## **Chapter 6**



# **Advanced Editing**

This chapter describes the editing operations available as built-in commands, including the use of regular expressions. These commands enable powerful find-and-replace functions, and make it possible to automate editing operations by using scripts.

Menu commands for editing are described in Chapter 3. For a full description of the use of the command language, see Chapter 5.

---



---

## Editing commands

The command language contains editing commands that duplicate the functions of many of the menu commands and provide additional capabilities. The editing commands are listed in Table 6-1. (They're explained in detail in Part II.)

**Table 6-1**  
Editing commands

|                                                                                  |                                                                                             |
|----------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| Adjust [-c <i>count</i> ] [-l <i>spaces</i> ] <i>selection</i> [ <i>window</i> ] | Adjust lines in a selection.                                                                |
| Align [-c <i>count</i> ] <i>selection</i> [ <i>window</i> ]                      | Align text with first line of selection.                                                    |
| Canon [ <i>option...</i> ] <i>dictionaryFile</i> [ <i>inputFile...</i> ]         | Replace a file's identifiers with canonical spellings given in <i>dictionaryFile</i> .      |
| Clear [-c <i>count</i> ] <i>selection</i> [ <i>window</i> ]                      | Delete selected text.                                                                       |
| Copy [-c <i>count</i> ] <i>selection</i> [ <i>window</i> ]                       | Copy selected text to the Clipboard.                                                        |
| Cut [-c <i>count</i> ] <i>selection</i> [ <i>window</i> ]                        | Copy selected text to the Clipboard and then delete the selection.                          |
| Entab [ <i>option...</i> ] [ <i>file...</i> ]                                    | Convert runs of spaces to tabs.                                                             |
| Find [-c <i>count</i> ] <i>selection</i> [ <i>window</i> ]                       | Find and select text.                                                                       |
| Font <i>fontname</i> <i>fontsize</i> [ <i>window...</i> ]                        | Change the font and/or size.                                                                |
| Line [ <i>number</i> ]                                                           | Find line number.                                                                           |
| Mark [ -y   -n ] <i>selection</i> [ <i>window</i> ]                              | Assign the marker <i>name</i> to range of text <i>selection</i> selected in <i>window</i> . |
| Markers [-q] [ <i>window ...</i> ]                                               | Print list of all markers associated with <i>window</i> .                                   |
| Paste [-c <i>count</i> ] <i>selection</i> [ <i>window</i> ]                      | Replace <i>selection</i> with the contents of the Clipboard.                                |
| Replace [-c <i>count</i> ] <i>selection</i> <i>replacement</i> [ <i>window</i> ] | Replace <i>selection</i> with <i>replacement</i> .                                          |
| Revert [ -y ] [ <i>window...</i> ]                                               | Revert window to last saved state.                                                          |
| Tab <i>number</i> [ <i>window...</i> ]                                           | Set a window's tab value to <i>number</i> spaces.                                           |
| Target <i>name</i>                                                               | Make a window the target window.                                                            |
| Translate <i>source</i> [ <i>destination</i> ]                                   | Convert selected characters.                                                                |
| Undo [ <i>window</i> ]                                                           | Undo last command.                                                                          |
| Unmark <i>name...</i> <i>window</i>                                              | Remove the marker(s) <i>name...</i> from the list of markers available for <i>window</i> .  |

If no *window* parameter is specified, editing commands act on the target window (the second window from the front). Therefore, to edit the active window, you'll need to switch to another window for entering your commands. (The Target command makes a window the target window; the Shell variables {Active} and {Target} always contain the full pathnames of the current active and target windows.)

Most editing commands take the following parameters:

- c *count*** You can specify a repeat count with the **-c** option—*count* is the number of times the command should be executed. *Count* may also be the infinity character,  $\infty$  (Option-5), which specifies that the operation should be repeated as many times as possible.
- selection*** Most editing commands act on a **selection**, either the current selection in the target window or another selection that you specify. First, an implicit Find is done to select the specified text. Then the text is modified. The selection syntax is defined in the next section.
- window*** The optional *window* parameter lets you specify the name of the window to be affected by a command, without changing the position of the affected window.

A command modifies the selection only if there were no syntactic errors in the selection, and all regular expressions were matched. Commands run silently unless an error occurs.

---

---

## Selections

*Selection* is a parameter to editing commands, and tells the command what text to select. A selection may be any of the following:

- a line in a file (selected by line number)
- a position in a file
- a specific character pattern
- a selection that begins and ends with any of the above

As an example of the selection syntax, consider the definition of the Find command:

```
Find [-c count] selection [window]
```

Find takes a selection as an argument and selects the argument text (or sets the insertion point). An actual command might take the form

```
Find /shazam/
```

This command finds and selects the first instance of the string "shazam" that appears after the current selection. (The slashes are used to enclose a *pattern*, a special case of a selection, as explained below.) No count is specified, so the command is executed once. No window name is specified, so the command operates on the target window.

Table 6-2 shows all of the selection operators. These are more fully explained in the sections following the table.

**Table 6-2**  
Selection operators

|                                           |                                                                                                |
|-------------------------------------------|------------------------------------------------------------------------------------------------|
| <b>Current selection</b>                  |                                                                                                |
| <b>§</b>                                  | Current selection in the target window (§ is Option-6 on the keyboard)                         |
| <b>Line numbered selections</b>           |                                                                                                |
| <b><i>n</i></b>                           | Line number <i>n</i>                                                                           |
| <b>!<i>n</i></b>                          | Line number <i>n</i> lines after the end of the current selection                              |
| <b>!<i>n</i></b>                          | Line number <i>n</i> lines before the start of the current selection (j is Option-1)           |
| <b>Position (insertion point)</b>         |                                                                                                |
| <b>•</b>                                  | Position before the first character of the file (• is Option-8)                                |
| <b>∞</b>                                  | Position after the last character of the file (∞ is Option-5)                                  |
| <b>Δ<i>selection</i></b>                  | Position before the first character of <i>selection</i> (Δ is Option-J)                        |
| <b><i>selection</i>Δ</b>                  | Position after the last character of <i>selection</i>                                          |
| <b><i>selection</i>!<i>n</i></b>          | Position <i>n</i> characters after the end of <i>selection</i>                                 |
| <b><i>selection</i>!<i>n</i></b>          | Position <i>n</i> characters before the beginning of <i>selection</i>                          |
| <b>Pattern (characters to be matched)</b> |                                                                                                |
| <b>/<i>pattern</i>/</b>                   | Pattern (regular expression)—search forward (see “Pattern Matching,” below)                    |
| <b>\ <i>pattern</i> \</b>                 | Pattern—search backward                                                                        |
| <b>Extended selection</b>                 |                                                                                                |
| <b><i>selection1:selection2</i></b>       | Both selections and everything in between                                                      |
| <b>marked selection <i>name</i></b>       | The <i>name</i> of a marked selection may contain any characters except<br>§ ! j ( : • ∞ Δ / \ |
| <b>Grouping</b>                           |                                                                                                |
| <b>(<i>selection</i>)</b>                 | Controls order of evaluation                                                                   |

A formal definition of selections can be found in Appendix B.

All of the operators group from left to right, and evaluation proceeds from left to right. The selection operators are listed below in order of precedence:

- / and \ Everything within slashes is taken as a regular expression, and evaluated as explained below under “Pattern Matching.”
- ( ) Controls order of evaluation.
- Δ Indicates position.
- ! and j Indicates position (! = after; j = before).
- :

Some examples will illustrate why it’s important to pay attention to the precedence of these operators:

Δ/*begin*!/! means (Δ/*begin*/)!1  
rather than Δ(/*begin*/)!1

That is, the insertion point is located after the “b” of “begin” rather than after the “n”.

/begin/:/end!/! means the selection /begin/:(/end!/!)  
rather than the position (/begin/:/end/)!1

That is, the character after “end” is included in the selection, as shown in Figure 6-1.

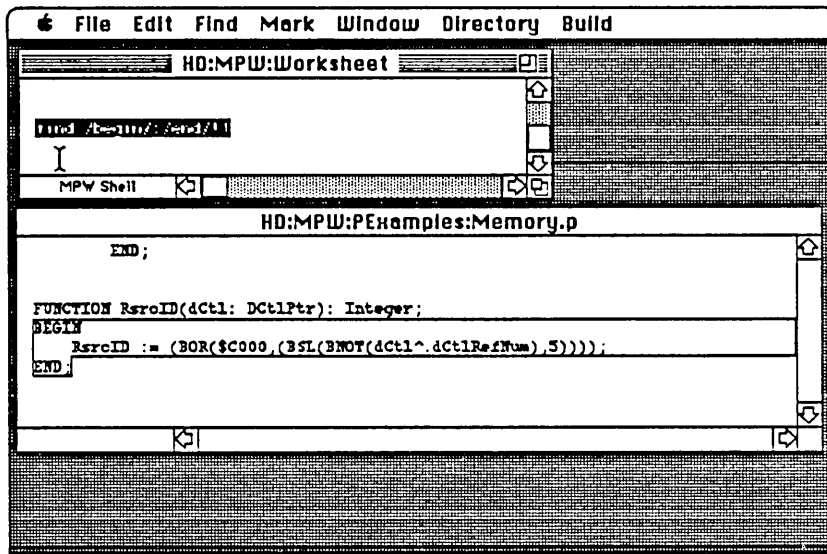


Figure 6-1  
A selection specification

## Current selection (§)

The current selection character, § (Option-6), always indicates the current selection in a window. If no window is specified, § indicates the current selection in the target window. For example, consider the windows shown in Figure 6-2.

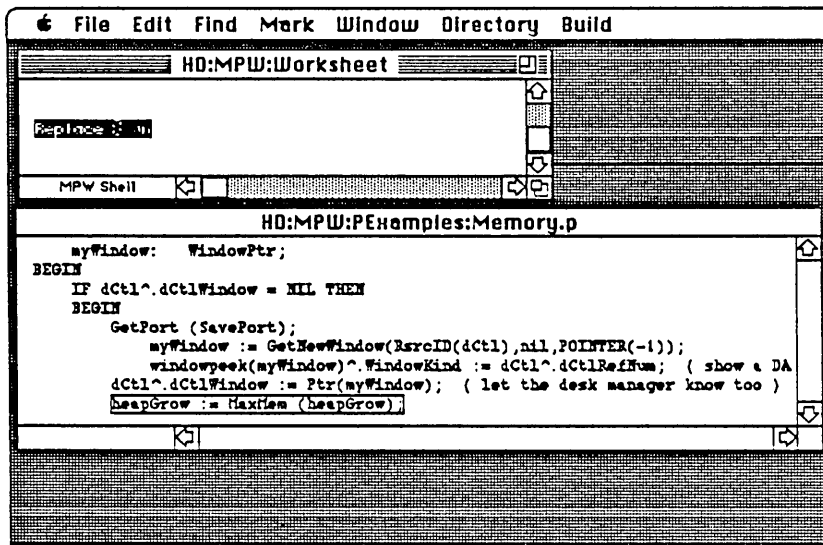


Figure 6-2  
Selections in two windows

The command

```
Replace $ \n
```

would replace the current selection in the target window with a single return (newline) character. (“\n” is a special code for inserting a return—see “Inserting Invisible Characters” later in this chapter.)

Note that the current selection is a dynamic quantity—it’s determined by the last *subexpression* evaluated, and thus represents the current state of a selection as it’s being calculated. For example, consider the command

```
Find /if/:$!1:$!1
```

At various points in the evaluation of the search string “/if/:\$!1:\$!1”, the current selection (\$) has the following different values:

|                        |                                                                                     |
|------------------------|-------------------------------------------------------------------------------------|
| Before calculation     | The pre-existing selection in the target window                                     |
| After “/if/”           | “if”                                                                                |
| After “/if/:\$!1”      | All characters from “if” to (and including) the first character after the “if”      |
| After “/if/:\$!1:\$!1” | All characters from “if” to (and including) the first two characters after the “if” |

---

## Selection by line number

If you give a number, unquoted by slashes, as a selection, it’s taken to be a line number. This may be an absolute line number, or a number of lines relative to the current selection. For example, to select line 3 of a file, you’d use the command

```
Find 3
```

This expression is equivalent to

```
Find '3'
```

but

```
Find 3 or Find '3'
```

is not equivalent to

```
Find /3/ or Find \3\
```

The exclamation mark and inverted exclamation mark (! and ¡) specify a number of lines after or before the current selection. The command

```
Find !3
```

selects a line that is 3 lines beyond the current selection. Note that the *!n* notation specifies a line relative to the *end* of the current selection (that is, *n* lines past the line containing \$Δ); *¡n* specifies a line relative to the *start* of the current selection (*n* lines before the line containing Δ\$).

---

## Position

A **position** is a special case of a selection. Position means the location of the insertion point only. The  $\Delta$  character (Option-J) is used to convey position relative to a selection. For example, consider the commands

```
Find 3
Find Δ 3
Find 3 Δ
```

The first Find command *selects* the entire third line in the target file. The `Find  $\Delta$ 3` and `Find 3 $\Delta$`  commands *place the insertion point* at the beginning and at the end of the third line.

You can also use the `!` and `;` operators to specify a position that's a given number of characters from a selection: *selection! $n$*  specifies a position  $n$  characters after *selection*, and *selection; $n$*  specifies a position  $n$  characters before *selection*.

Notice that this leads to two different uses of the `!` and `;` operators, as in the following example:

```
Find !4!4
```

The first "`!4`" indicates a *selection* that's 4 *lines* beyond the current selection; the second "`!4`" indicates the *position* that's 4 *characters* beyond the end of that selection.

You can specify other positions in a file with the following special notation:

- (Option-8) Position preceding the first character in file
- ∞ (Option-5) Position following last character in file

---

## Extending a selection

A colon is used to join two selections. For example,

```
Find /begin/:/end/
```

This command selects "begin", "end", and everything in between. (See Figure 6-1 above.) Compare this command with

```
Find /begin=end/
```

which looks for a begin-end pair on a single line.

---

## Markers

A marker is a selection that has been given a name. A marker may be used as a selection variable. You can mark as many selections and insertion points as you wish. You can create markers directly by selecting text in a window and then clicking the Mark command in the Mark menu. See "Mark Menu" in Chapter 3 for more information on the interactive use of markers. This section describes the general behavior and programmatic use of markers.

Markers may be as simple as a position in a window, but more often a marker names a range of positions. Markers have the special attribute of being able to remember their assigned position(s) even when you're making editing changes all around them. For example, typing before marked text has the effect of moving both the text and its associated marker toward the end of the window. Editing "inside" the range of a marker will either increase or decrease the range of the marker depending on whether the editing was an insertion or deletion, respectively.

Markers are "sticky." For example, if an insertion point is marked and you type at that point, everything you type will be added to that marker.

If you delete the text encompassing a marker the marker will also be deleted. For example, if the string "xyz" is deleted and the character "y" is marked, the "y" marker will be deleted. However, if the string "xyz" itself is marked as "y", deleting the string "xyz" will result in marker "y" being reduced to an insertion point.

Markers are associated with individual windows. When you switch between windows, the Mark menu is updated to reflect the markers of the new active window.

Markers are persistent. They are saved in the resource fork of the file you are editing, just like the font, tab, and other information about the window.

You can create or delete Markers programmatically by using the following three Shell commands:

|                                                   |                                                                                             |
|---------------------------------------------------|---------------------------------------------------------------------------------------------|
| Mark [-y   -n] <i>selection</i> [ <i>window</i> ] | Assign the marker <i>name</i> to range of text <i>selection</i> selected in <i>window</i> . |
| Markers [ <i>window</i> ]                         | Print list of all markers associated with <i>window</i> .                                   |
| Unmark <i>name</i> ... <i>window</i>              | Remove the marker(s) <i>name</i> ... from the list of markers available for <i>window</i> . |

For example, to mark the currently selected text in the target window with the name "Function B" and to replace any previous marker of that name, you would type

```
Mark -y $ 'Function B'
```

The new marker name will appear in the Mark menu. You might remove the marker later by using the Unmark command:

```
Unmark 'Function B' "{Target}"
```

This command would remove that marker from the target window.



The new selection syntax allows you to jump to a marker programmatically and automatically select the output of a script for a user. To jump to a marker named “george”, you might use a command similar to the following:

```
Find george
```

To automatically select the output of a script for a user, you could use a script similar to the following:

```
Mark SA X #Mark the start of output
Make #Run your Make command
Find X #Select the output of Make
```

For further details on the Shell commands for markers, see Part II.

---

## Pattern

A **pattern** may be either a literal text pattern or a regular expression (defined in the next section). You specify a pattern between the `/.../` and `\...\` delimiters. Forward slashes indicate a search forward, and back slashes indicate a search backward. A forward search begins at the end of the current selection and continues to the end of the file. A backward search begins at the start of the current selection and continues to the beginning of the file. For example, the command

```
Find /myString/
```

searches forward for the literal expression “mystring”. (Recall that to specify case-sensitive pattern matching, you need to set the Shell variable `(CaseSensitive)`, or select the “Case Sensitive” menu item.)

- ❖ *Note:* To locate the insertion point at the beginning of the target window, for instance before executing a Find command, you can use the command

```
Find •
```

In fact, this command is so useful that you may want to add it as a menu command—see the example under the `AddMenu` command in Part II.

---

---

## Pattern matching (using regular expressions)

Regular expressions are a shorthand language for specifying text patterns. Regular expressions are used in editing commands, in the Search command (which searches one or more files for occurrences of a pattern), and in If and Evaluate expressions following the `=~` and `!~` operators. Most of the regular expression operators may also be used in filename generation.

Regular expressions are always used within the pattern delimiters `/.../` or `\...\`.

A special set of metacharacters, called regular expression operators, is used in regular expressions (and in filename generation). The regular expression operators are listed in Table 6-3.

**Table 6-3**  
Regular expression operators

|                                                       |                                                                                                                                                           |
|-------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>c</i>                                              | Any character matches itself (unless it's one of the special characters listed below)                                                                     |
| <i>∂ c</i>                                            | Defeat special meaning of following character ( <i>c</i> is taken literally) except<br><br><i>∂n</i> = return<br><i>∂t</i> = tab<br><i>∂f</i> = form feed |
| '...'                                                 | Literalize enclosed characters                                                                                                                            |
| "..."                                                 | Literalize enclosed characters, except <i>∂</i> , {, and `                                                                                                |
| ?                                                     | Any single character (other than return)                                                                                                                  |
| ≈                                                     | Any string of 0 or more characters, not containing a return                                                                                               |
| [ <i>character...</i> ]                               | Any character in the list                                                                                                                                 |
| [ <i>¬character...</i> ]                              | Any character not in the list ( <i>¬</i> is Option-L on the keyboard)                                                                                     |
| <i>regularExpr*</i>                                   | Regular expression 0 or more times                                                                                                                        |
| <i>regularExpr+</i>                                   | Regular expression 1 or more times                                                                                                                        |
| <i>regularExpr«n»</i>                                 | Regular expression <i>n</i> times (« is Option-\ ; » is Option-Shift-\)                                                                                   |
| <i>regularExpr«n,»</i>                                | Regular expression <i>n</i> or more times                                                                                                                 |
| <i>regularExpr«n<sub>1</sub>,n<sub>2</sub>»</i>       | Regular expression <i>n<sub>1</sub></i> to <i>n<sub>2</sub></i> times                                                                                     |
| ( <i>regularExpr</i> )                                | Grouping                                                                                                                                                  |
| ( <i>regularExpr</i> ) <i>®n</i>                      | Tagged regular expression (where 0 ≤ <i>n</i> ≤ 9)                                                                                                        |
| <i>regularExpr<sub>1</sub>regularExpr<sub>2</sub></i> | <i>regularExpr<sub>1</sub></i> followed by <i>regularExpr<sub>2</sub></i>                                                                                 |
| • <i>regularExpr</i>                                  | Regular expression at beginning of line                                                                                                                   |
| <i>regularExpr</i> ∞                                  | Regular expression at end of line                                                                                                                         |

These characters are considered special in the following circumstances:

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>∂</i>        | Special everywhere except within single quotes ('...')  |
| ? ≈ * + [ « ( ) | Special anywhere except within [...], '...', and "..."  |
| ®               | Special only after a right parenthesis, )               |
| •               | Special as first character of entire regular expression |
| ∞               | Special as last character of entire regular expression  |
| / \             | Special if used to delimit a regular expression         |

Their precedence (from highest to lowest) is as follows:

1. ( )
2. ? ≈ \* + [ ] « ®
3. concatenation
4. • ∞

A formal definition of regular expressions can be found in Appendix B. The rest of this section describes the use of regular expressions for describing selections.

---

## Character expressions

In the simplest case, regular expressions consist of literal characters enclosed in slashes. For example,

```
/what the ?/
```

Notice one complication, however—if the literal character happens to be one of the regular expression operators (such as “?”), it will be specially interpreted rather than taken as a literal character. If you want to specify a literal character that happens to have a special meaning within the context of regular expressions, you’ll have to precede it with the escape character, `\`, or enclose it in quotes. The character `\` has the effect of “literalizing” the character that follows it. For example, to find the literal expression given above, you would use one of the following commands:

```
Find /what the \?/
Find /what the '?'/
Find /'what the ?'/
```

You could also use double quotes, that is “...”.

---

## Wildcard operators

In addition to literal characters, regular expressions can include the operators `?`, `=` (Option-X), and `[ ]`, which are used as follows:

- `?` Any character other than return
- `=` Any string not containing return, including the null string (this is the same as `?*`)
- `[characterList]` Any character in the character list (as defined below)
- `[¬ characterList]` Any character *not* in the list

These operators are also used as wildcards in filename generation. (You can also use the `*`, `+`, and `«...»` operators in filename generation—see “Filename Generation” in Chapter 5.)

A character list is an expression consisting of one or more characters enclosed in square brackets (`[...]`). It matches any character found in the list. The case-sensitivity of characters in the list is governed by the `{CaseSensitive}` variable (which you can set or unset by toggling the Case Sensitive check box in the Find or Replace dialog boxes). A list may consist of individual characters or a range of characters, specified with the minus sign (`-`). For instance, the following two commands are equivalent:

```
Find /[ABCDEF]/
Find /[A-F]/
```

You can also mix the two notations:

```
Find /[0-9A-F$]/
```

- ❖ *Note:* This command specifies any of the characters 0 through 9, A through F, and \$. To specify the `]` or `-` character, place it at the beginning of the list or literalize it with the escape character, `\`.

The negation symbol, `¬` (Option-L), lets you specify any character *not* in the list. For example,

```
Find /[-A-Z]/
```

This example specifies all characters *except* the letters A through Z. (To specify the `¬` character itself, place it anywhere in the list other than the beginning, or literalize it by preceding it with the escape character, `\`.)

---

## Repeated instances of regular expressions

The asterisk character (\*) matches zero or more occurrences of the immediately preceding regular expression. The plus sign (+) matches one or more occurrences of an expression. For example, the command

```
Find /[0-9]+/
```

will find any string of one or more digits.

You can also specify an expression that occurs an explicit number of times with the `<n>` notation:

|                                                             |                                                                                               |
|-------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| <code>regularExpr&lt;n&gt;</code>                           | Regular expression <i>n</i> times                                                             |
| <code>regularExpr&lt;n,&gt;</code>                          | Regular expression at least <i>n</i> times                                                    |
| <code>regularExpr&lt;n<sub>1</sub>,n<sub>2</sub>&gt;</code> | Regular expression at least <i>n<sub>1</sub></i> times and at most <i>n<sub>2</sub></i> times |

For example,

```
Replace -c ∞ /' '«4,»/ @t
```

This command finds any string of four or more spaces, and replaces it with a tab. (The `-c ∞` option specifies a repeat count of “infinity”; that is, it replaces all occurrences of the selection to the end of the document.)

---

## Tagging regular expressions with the @ operator

The @ (Option-R) operator tags a regular expression between parentheses. This operator is useful with the Replace command, for example, in reformatting tables of data. Consider a table with two columns of numbers separated by spaces or tabs:

```
123 456
123 456
123 456
123 456
etc.
```

The following Replace command switches the order of the two columns:

```
Replace -c ∞ /([0-9]+)@1[@t]+([0-9]+)@2/ '@2 @1'
```

Translated into English, this expression means

- `[0-9]+` Match one or more characters in the set “0” to “9”.
  - `([0-9]+)@1` Remember that selection (the expression enclosed in parentheses) as @1.
  - `[ @t]+` Next, match at least one space or tab.
  - `([0-9]+)@2` Then match one or more characters in the set “0” to “9” and remember it as @2.
  - `'@2 @1'` Finally, replace the whole matched string with what was remembered as @2, a space, and what was remembered as @1.
- Note:* The quotes are stripped off, as explained under “Quoting Special Characters” in Chapter 5. The @ operator itself can be disabled only with the escape character, @.

After this sequence is executed, the table will look like this:

```
456 123
456 123
456 123
456 123
etc.
```

---

## Matching a pattern at the beginning or end of a line

In the context of regular expressions, the `•` metacharacter (Option-8) means that the subsequent expression must be matched at the beginning of a line. For example, the regular expression

```
/•main/
```

will match a line that begins with “main” but not a line that begins with “*space* main”. The beginning of a line is either the first character after a return or the first character of the file.

Likewise, the `∞` metacharacter means that the previous expression must be matched at the end of a line. The regular expression

```
/main∞/
```

will match a line that ends with “main” but not a line that ends with “main *space*”. The end of a line is either the last character of a line prior to the return, or the end of the file.

Notice that `•` and `∞` have another meaning within selections. Within a pattern, they indicate the beginning and end of a *line*. Within a selection, they indicate the beginning and end of the *file*.

---

## Inserting invisible characters

You can use the Shell escape character, `∂`, to insert the following special characters in text:

```
∂n return
∂t tab
∂f form feed
```

❖ *Note:* The “Show Invisibles” menu item shows the invisible space, tab, and return characters in a file.

For more information on the escape character, see “Quoting Special Characters” in Chapter 5.

---

## Note on forward and backward searches

Forward and backward searches aren't always completely symmetrical. For example, consider the command

```
Find /?*/
```

This command finds zero or more occurrences of any character other than a return. The first time you execute this command, if the current selection is not at the end of a line, some range of characters will be selected. However, in subsequent invocations, the selection will hang at the end of the line—only an insertion point will be left at the end of the line. This is because the `*` metacharacter matches zero occurrences and the search starts with the character following the current selection—in this case, the insertion point preceding a return. A backward search of the form

```
Find \?*\
```

will never hang at the beginning of a line. This is because a backward search begins with the first character to the left of the current selection and so has the effect of jumping over a return after encountering it.

---

---

## Some useful examples

This section shows some examples of complex use of regular expressions.

---

## Transforming DumpObj output

The DumpObj command, described in Part II, formats the contents of an object file. This example shows how to transform a DumpObj listing, such as the following, back into valid assembly code.

```
000000: 4EBA 06F8 'N...' JSR *+$06FA ; 6004282A
000004: 4EBA 04EA 'N...' JSR *+$04EC ; 60042620
000008: 3B7C 0014 FCC4 '|....' MOVE.W #$0014,$FCC4(A5)
00000E: 266D 0010 '&m..' MOVEA.L $0010(A5),A3
000012: 2653 '&S' MOVEA.L (A3),A3
000014: 0C5B 0000 '.[..' CMPI.W #$0000,(A3)+
000018: 6600 0008 'f...' BNE *+$000A ; 60042152
00001C: 3A1B ':. ' MOVE.W (A3)+,D5
00001E: 6600 0010 'f...' BNE *+$0012 ; 60042160
etc.
```

You could position the insertion point at the beginning of the code and use the following Replace command:

```
Replace -c ∞ /?«41»/ "ðtðt" # replace everything up to the
 # instruction with 2 tabs
```

However, the previous command works only because DumpObj happens to place the instruction at column 42. The following example, by defining some Shell variables, works regardless of the exact column layout:

```
Set hex "[0-9A-F]«4,6»" # 4 to 6 characters in the set 0-9 and A-F
Set space "[ðt]+" # 1 or more spaces or tabs
Set chars "ðð'?'+ðð'" # 1 or more of any character between ð
 # single quotes
```

```
Replace -c ∞ /{hex}::{(space){hex}}«1,3»{(space){chars}{space}}/ "ðtðt"
```

---

## Finding a whole word

The following example illustrates how you could find an exact match for a C identifier that you had previously defined in the variable {ident}:

```
Set tokensep "[_a-zA-Z_0-9]" # a token separator is any character
 # not in the set a-z, A-Z, _, or 0-9

Set CaseSensitive 1 # set to "true"—the case of each
 # character must match
```

The following Find command is not quite right, because it selects not only the matched identifier, but also the token separator on each side of the identifier:

```
Find /(tokensep){ident}(tokensep)/
```

The following Find command selects only the matched identifier. It accomplishes this by adding 1 to the starting position of the selection (*Δselection!1*), and using that as the starting point for a new selection that extends to the beginning of the next token separator:

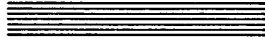
```
Find Δ/(tokensep){ident}(tokensep)/!1:Δ/(tokensep)/
```







## **Chapter 7**



### **Editing Resources With ResEdit**

The chapter describes ResEdit, a stand-alone application for editing resources.

- ❖ *Note:* As in *Inside Macintosh*, resource types are shown within single quotes; for example, 'STR ' (that is, *STRspace*). The quotes are not part of the name.

---

---

## About ResEdit

ResEdit is an interactive, graphically based application for manipulating the various resources in a Macintosh application. It lets you create and edit all standard resource types except 'CODE', and copy and paste all resource types (including 'CODE'). ResEdit actually includes a number of different resource editors: There is a general resource editor, for editing any resource in hex and ASCII format, and there are several individual resource editors for specific types of resources. You can also create your own resource editors to use with ResEdit.

---

## Uses

ResEdit is especially useful for creating and changing graphic resources such as dialog boxes and icons. For example, you can use ResEdit to put together a quick prototype of a user interface and try out different formats and presentations of resources. Anyone can quickly learn to use ResEdit for translating resources into a foreign language without having to recompile the program. You can use ResEdit to modify a program's resources at any stage in the process of program development.

Once you have created or modified a resource with ResEdit, you can use the Resource Decompiler, DeRez, to convert the resource to a textual representation that can be processed by the Resource Compiler, Rez. You can then add comments to this text file or otherwise modify it with the Shell editor. (Rez and DeRez are fully described in the next chapter.)

---

## Extensibility

A key feature of ResEdit is its extensibility. Because it can't anticipate the format of all the different types of resources that you might use, ResEdit has been designed so that you can teach it to recognize and parse new resource types.

There are two ways that you can extend ResEdit to handle new types:

- You can create templates for your own resource types. ResEdit lets you edit most resource types by filling in the fields of a dialog box—this is the way you edit 'BNDL' and 'FREF' resources, for example. The layout of these dialog boxes is determined from a template in ResEdit's resource file, and you can add templates to edit new resource types. Resource templates are described later in this chapter.
- You can also program your own special-purpose resource picker and/or editor and then add it to ResEdit. The *resource picker* is the code that displays all the resources of one type in the resource type window. The *editor* is the code that displays and allows you to edit a particular resource. These pieces of code are separate from the main code of ResEdit. A set of Pascal routines, called ResEd, is available for this purpose—see the *MPW Pascal 2.0 Reference* for information.

---

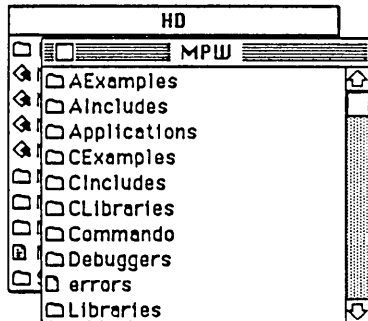
---

## Using ResEdit

From the MPW Shell, you can start ResEdit by entering the command

```
ResEdit
```

(This assumes, of course, that ResEdit is in the Applications folder, or elsewhere in the search path defined by the {Commands} variable.) From the Finder, you can just select and open the ResEdit icon. ResEdit displays a window that lists the files and folders for each disk volume currently mounted (Figure 7-1).



**Figure 7-1**  
Disk volume windows

---

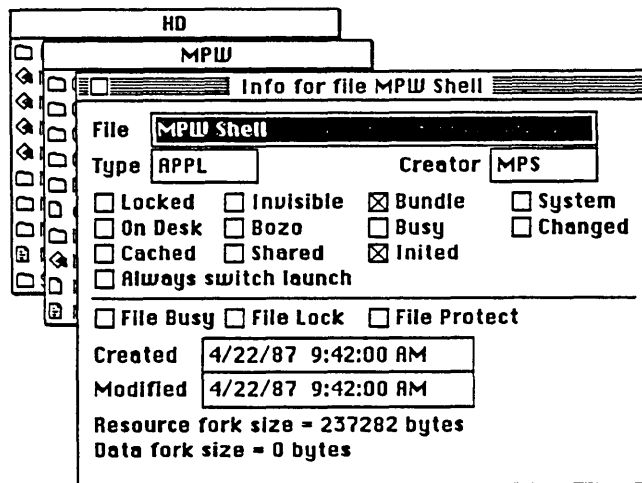
## Working with files

To list the resource types in a file, select and open the filename from the list. (You can select a name by clicking on it or by typing one or more characters of the name.)

When a directory window is the active window, the File menu commands act as follows:

- |       |                                                                                                                       |
|-------|-----------------------------------------------------------------------------------------------------------------------|
| New   | Creates a new file.                                                                                                   |
| Open  | Opens the selected file or folder. (This is the same as double-clicking on the name.)                                 |
| Close | Closes the volume window. (This is the same as clicking the close box.) If it's a 3.5-inch disk, the disk is ejected. |

**Get Info** Displays file information and allows you to change it. Figure 7-2 is an example of the File Info window for the MPW Shell file.



**Figure 7-2**  
A File Info window

**Transfer** Allows you to transfer to an application other than the application that launched ResEdit. (This is an alternative to the Quit command.)

**Quit** Quits from ResEdit and returns to the MPW Shell (or Finder).

---

### Warning

You can edit any file shown in the window, including the System file and ResEdit itself. However, it's dangerous to edit a file that's currently in use. Edit a copy of the file instead. (For example, edit the System file on a non-boot volume.)

---

ResEdit will recognize a new disk when it's inserted, and handle multiple drives. Note that you can also use ResEdit to copy or delete files:

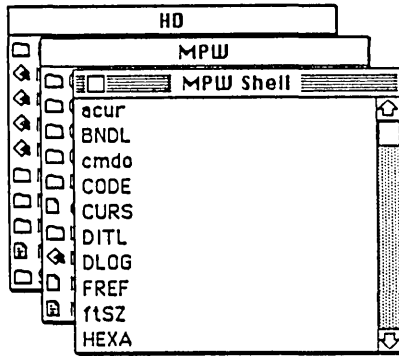
- To delete a file, select the file and choose Clear from the Edit menu.
- To copy a resource file, you must select all of its resources and copy them. Then paste them into a new file. (File attributes are not automatically copied by this operation—you must set them via the Get Info command.) ResEdit cannot copy a data fork.

---

### Working within a file

When you open a file, a window displays a list of all the resource types in that file (Figure 7-3). While this window is the active window, you can create new resources, copy or delete existing resources, and paste resources from other files.

- ❖ *Note:* The resources are displayed by a resource picker. The general resource picker displays the resources by type, name, and ID number; there are also special resource pickers for some resource types. (For example, the 'ICON' resource picker displays the icons graphically.)



**Figure 7-3**  
A ResEdit file window

When a file window is the active window, the File menu commands have the following effects:

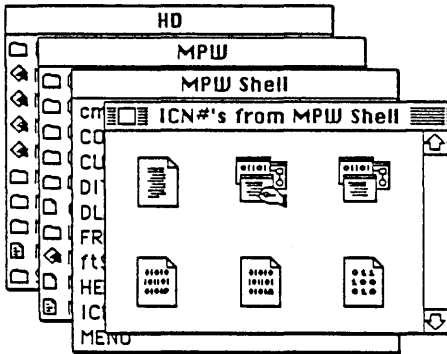
- |              |                                                                                                                                                                                                                                                                                                          |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| New          | Creates a new resource in the open file.                                                                                                                                                                                                                                                                 |
| Open         | Opens a window displaying all resources of the resource type selected. (Select the resource type by clicking on it or by typing its first character.)<br><br><i>Note:</i> If you hold down the Option key while opening a resource type, the resource window will open with the general resource picker. |
| Open General | Opens the general resource picker.                                                                                                                                                                                                                                                                       |
| Close        | Closes the file window and asks if you want to save the changes you have made.<br><br><i>Note:</i> If you've made changes, you should not reboot before closing.                                                                                                                                         |
| Revert       | Changes the resource file back to the version that was last saved to disk.                                                                                                                                                                                                                               |
| Quit         | Quits from ResEdit.                                                                                                                                                                                                                                                                                      |

When a file window is the active window, the Edit menu commands have the following effects:

- |           |                                                                                                                                   |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------|
| Cut       | Removes all resources of the resource types selected, placing them in the ResEdit scrap.                                          |
| Copy      | Copies all resources of the resource types selected into the ResEdit scrap.                                                       |
| Paste     | Copies the resources from the ResEdit scrap into the file window's resource type list.                                            |
| Clear     | Removes all resources of the resource type selected, without placing them in the ResEdit scrap.                                   |
| Duplicate | Creates duplicates of all resources of the resource types selected, and assigns a unique resource ID number to each new resource. |

## Working within a resource type

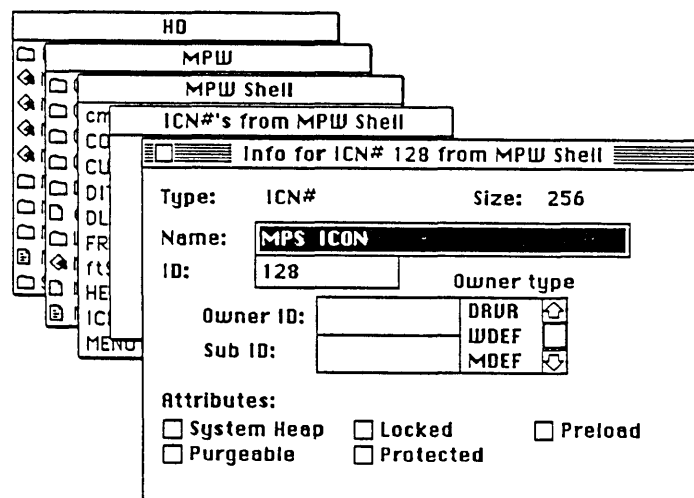
Opening a resource type produces a window that lists each resource of that type in the file (Figure 7-4). This list will take different forms, depending on the particular resource picker; if you hold down the Option key during the open, the general resource picker is invoked.



**Figure 7-4**  
A resource type window

When a resource type window is the active window, the File menu commands have the following effects:

- |              |                                                                                  |
|--------------|----------------------------------------------------------------------------------|
| New          | Creates a new resource and opens its editor.                                     |
| Open         | Opens the appropriate editor for the resource you selected.                      |
| Open a ...   | Allows you to open an editor template of a different type.                       |
| Open General | Opens the general (hex) resource editor.                                         |
| Close        | Closes the resource type window.                                                 |
| Revert       | Changes the file back to what it was before you opened the resource type window. |
| Get Info     | Displays resource information and allows you to change it.                       |



**Figure 7-5**  
A Get Info window for ICN#'s

When a resource type window is the active window, the Edit menu commands have the following effects:

|           |                                                                                                                     |
|-----------|---------------------------------------------------------------------------------------------------------------------|
| Undo      | Undoes the most recent editing command. Undo may or may not be selectable, depending on the specific editor in use. |
| Cut       | Removes the resources that are selected, placing them in the ResEdit scrap.                                         |
| Copy      | Copies all the resources that are selected into the ResEdit scrap.                                                  |
| Paste     | Copies the resources from the ResEdit scrap into the resource type window.                                          |
| Clear     | Removes the resources that are selected, without placing them in the ResEdit scrap.                                 |
| Duplicate | Creates a duplicate of the selected resources and assigns a unique resource ID number to each new resource.         |

---

## Editing individual resources

To open an editor for a particular resource, either double-click on the resource or select it and choose Open from the File menu. One or more auxiliary menus may appear, depending on the type of resource you're editing. Some editors, such as the 'DITL' editor, allow you to open additional editors for the elements within the resource. All the editors use File and Edit menus similar to those described above, but operate on individual resources or individual elements of a resource, and hence vary in their appearance and function as explained below.

If you hold down the Option key while opening a resource, the *general data editor* is invoked. This editor allows you to edit the resource as hexadecimal or ASCII data. If you hold down the Shift and the Option keys while opening, ResEdit shows you a list of all editors and templates.

---

### Caution

Individual editors may not be appropriate for all resource types—inappropriate editors may cause system errors to occur.

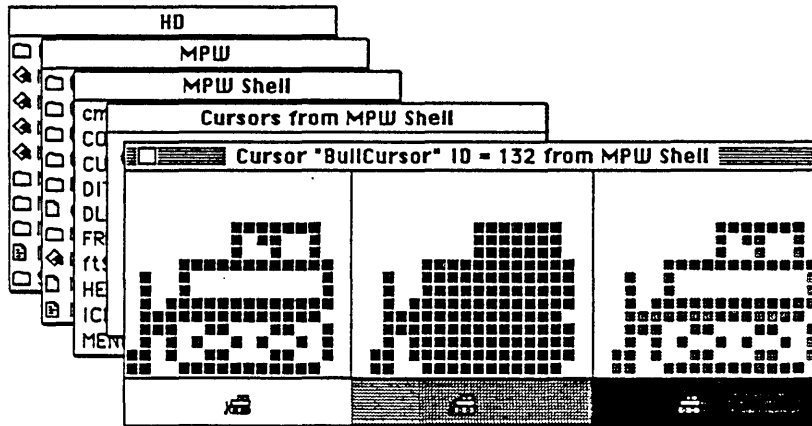
---

The menus for some of the editors are discussed below. The use of the editors not discussed here should be apparent when you run them.

- ❖ *Note:* The general data editor will not edit resources larger than 16K bytes; however, you can *move* larger resources with the Cut, Copy, Paste, and Clear commands as described above.

## 'CURS' (cursor) resources

For 'CURS' resources, the editor displays three images of the cursor (Figure 7-6). You can manipulate all three images with the mouse.



**Figure 7-6**  
Editing 'CURS' resources

In Figure 7-6 the left image shows how the bulldozer cursor will appear. The middle image is the mask for the cursor, which affects how the cursor appears on various backgrounds. The right image shows a gray picture of the cursor with a single point in black—this point is the cursor's hot spot.

The Cursor menu contains the following commands:

- Try Cursor      Lets you try out the cursor by having it become the cursor in use.  
                  (Restore Arrow restores the standard arrow cursor.)
- Data → Mask    Copies the cursor image to the mask editing area.

## 'DITL' (dialog item list) resources

For 'DITL' resources, the editor displays an image of the item list as your program would display it in a dialog or alert box. When you select an item, a size box appears in the lower-right corner of its enclosing rectangle so that you can change the size of the rectangle. You can move an item by dragging it with the mouse.

If you open an item within the dialog box, the editor associated with the item is invoked; for an 'ICON', for example, the icon editor is invoked. If you hold down the Option key while opening, the general data editor is invoked.

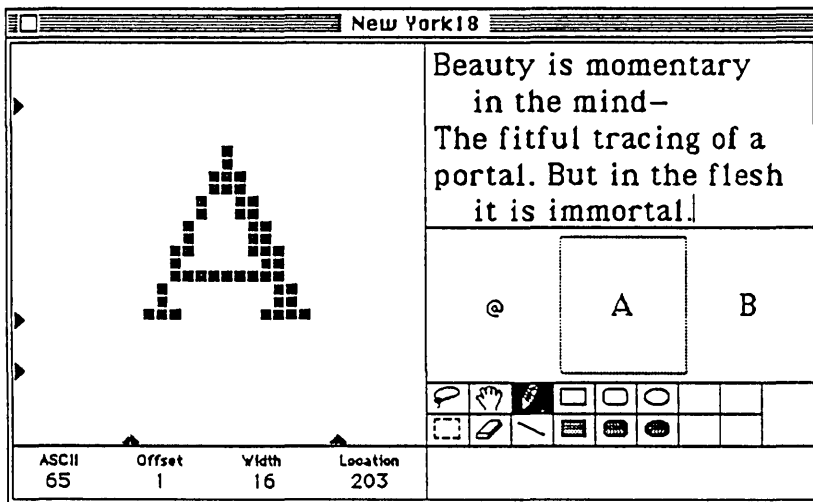


The DITL menu contains the following commands:

- Bring to Front** Allows you to change the order of items in the item list. Bring to Front causes the selected item to become the last (highest numbered) item in the list. The actual number of the item is shown by the 'DITM' editor.
- Send to Back** Like Bring to Front, except that it makes the selected item the first item in the list—that is, item number 1.
- Grid** Aligns the item on an invisible 8-pixel by 8-pixel grid. If you change the item location while Grid is on, the location will be adjusted such that the upper-left corner lies on the nearest grid point above and to the left of the location you gave it. If you change the size, it will be made a multiple of 8 pixels in both dimensions.
- Use RSRC Rect** Restores the enclosing rectangle to the rectangle size stored in the underlying resource. Note that this works on 'ICON', 'PICT', and 'CNTL' items only; the other items have no underlying resources.
- Use Full Window** Adjusts the window size so that all items in the item list are visible in the window.

### 'FONT' resources

For 'FONT' resources, the editor window is divided into four panels: a character editing panel, a sample text panel, a character selection panel, and a set of MacDraw-like graphics tools. These are shown in Figure 7-7.



**Figure 7-7**  
'FONT' editor window

The *character editing panel*, on the left side of the window shows an enlargement of the selected character. You can edit it, as with FatBits in MacPaint, by clicking bits on and off. Drag the black triangles at the bottom of the character editing panel to set the left and right bounds (that is, the character width). The three triangles at the left of the panel control the ascent, baseline, and descent.

The *sample text panel*, at the upper right, displays a sample of text in the font currently being edited. (You can change this text by clicking in the text panel and using normal Macintosh editing techniques.)

The *character selection panel* is below the text panel. You can select a character to edit by typing it (using the Shift and Option keys if necessary), or by clicking on it in the row of three characters shown. (Click on the right character in the row to move upward through the ASCII range; click on the left character to move downward.) The character you select is boxed in the center of the row with its ASCII value shown below it (in decimal).

The *graphic tools panel*, directly below the character selection panel, offers a dozen or more of the familiar MacDraw-type tools including the hand mover, pencil, eraser, circles, and rectangles. In addition, a selection of colors is available.

---

### **Caution**

Changing the ascent or descent of a character changes the ascent or descent for the entire font.

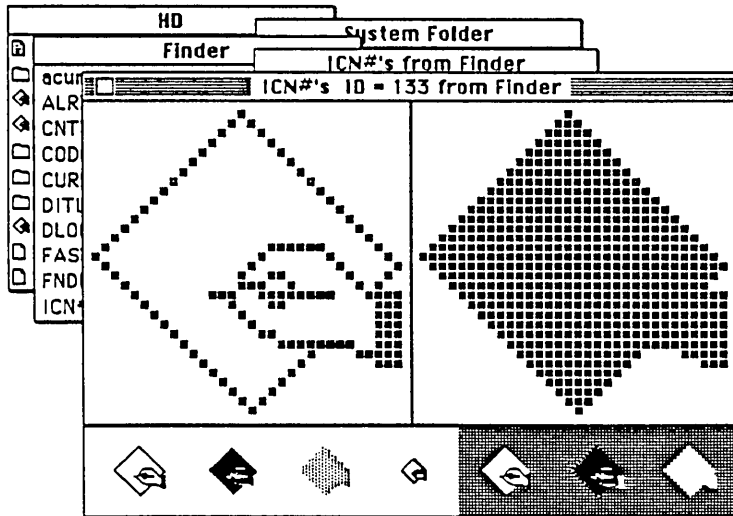
---

Any changes you make in the character editing panel are reflected in the text panel and the character selection panel. Remember that you cannot save the changes until you close the file.

You can also change the name of a font. The font name is stored as the name of the resource of that font family with size 0. This resource does not show up in the normal display of all fonts in a file. To display it, hold down the Option key while you open the 'FONT' type from the file window. You will see a generic list of fonts. Select the font with the name you wish to change, and choose Get Info.

## 'ICN#' (icon list) resources

For 'ICN#' resources, the editor displays two panels in the window (Figure 7-8). The upper panel is used to edit the icon. It contains an enlargement of the icon on the left and an enlargement of the icon's mask on the right. The lower panel shows, from left to right, how the icon will look unselected, selected, and open on both a white and a gray background. It also shows how the icon will appear in the Finder's small icon view.



**Figure 7-8**  
'ICN#' window

To install a new icon for your application when you already have an old one in the Finder's desktop file, follow these steps:

1. Open the file called DeskTop.
  2. Open type 'BNDL' and find the bundle that belongs to your application. (This is the one that has your owner name in it.) Look through the bundle and mark down the type and resource ID of all resources bundled together by the bundle (that is, the 'ICN#'s and 'FREF's).
  3. Go back to the DeskTop window and remove these resources along with your 'BNDL' and signature resource (the resource whose type is your application's signature).
  4. Now close the DeskTop window, save changes, and quit ResEdit. Your new icon will be installed if you have the proper 'BNDL', 'FREF', and 'ICN#' resource numberings.
- ❖ *Note:* To see how 'BNDL', 'FREF', and 'ICN#' resources are interrelated, use ResEdit to look at those resources in an existing application such as the MPW Shell.

Alternatively, you can rebuild the DeskTop file by holding down the Option and Command keys when entering the Finder. (This method is faster and easier, but you will lose your Finder Get Info comments; you will also lose folder names on a non-HFS volume.)

---

---

## Creating a resource template

You can customize ResEdit by creating new templates for your own resource types. The generic way of editing a resource is to fill in the fields of a dialog box—for example, this is the way you edit 'FREF', 'BNDL', and 'STR#' resources. The layout of these dialog boxes is set by a template in ResEdit's resource file. The template specifies the format of the resource and also specifies what labels should be put beside the editText items in the dialog box that's used for editing the resource. You can find these templates by opening the ResEdit file and then opening the type window for 'TMPL' resources. For example, if you open the template for 'WIND' resources (this is the 'TMPL' with name "WIND"), you'll see the template shown in Figure 7-9.

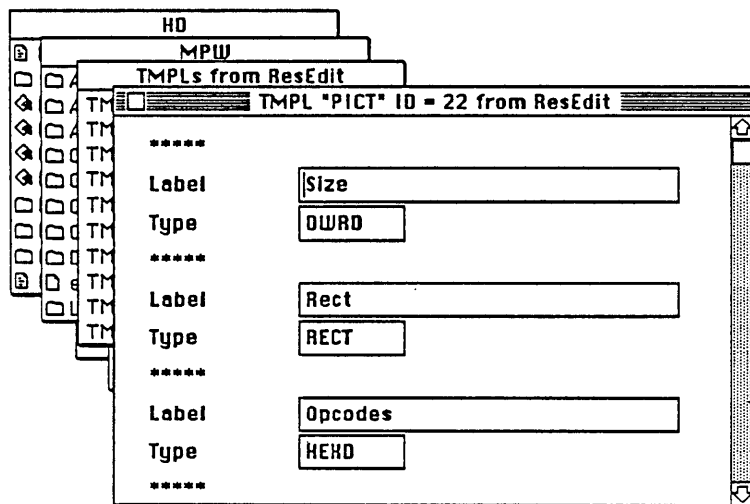


Figure 7-9  
Window template data

The window template, then, consists of the following:

1. A RECT (4 words) specifying the boundary of the window.
2. A word that is the procID for the window (DWRD tells ResEdit to display the word in decimal as opposed to hex).
3. A Boolean indicating whether or not the window is visible. (BOOL is 2 bytes in the resource but is displayed as a radio button in the dialog window used for editing.)
4. Another Boolean indicating whether or not the window has a close box.
5. A long word that is the reference value (refCon) for the window. (DLNG indicates that it should be displayed in the editor as a decimal number.)
6. A Pascal string (PSTR), the title of the window.

You can look through the other templates and compare them with the structure of those resources to get a feel for how you might define your own resource template. (These templates are equivalent to the resource type declarations contained in the (RIncludes) directory—refer also to the DeRez command in Part II.)

These are the types you have to choose from for your editable data fields:

|                  |                                                                                           |
|------------------|-------------------------------------------------------------------------------------------|
| DBYT, DWRD, DLNG | Decimal byte, word, long word                                                             |
| HBYT, HWRD, HLNG | Hex byte, word, long word                                                                 |
| HEXD             | Hex dump of remaining bytes in resource                                                   |
| PSTR             | Pascal string (length byte followed by the characters)                                    |
| LSTR             | Long string (length long followed by the characters)                                      |
| WSTR             | Same as LSTR, but a word rather than a long word                                          |
| ESTR, OSTR       | Pascal string padded to even or odd length (needed for DITL resources)                    |
| CSTR             | C string                                                                                  |
| ECST, OCST       | Even-padded C string, or odd-padded C string (padded with nulls)                          |
| BOOL             | Boolean                                                                                   |
| BBIT             | Binary bit                                                                                |
| TNAM             | Type name (4 characters, like OSType and ResType)                                         |
| CHAR             | A single character                                                                        |
| RECT             | An 8-byte rectangle                                                                       |
| Hnnn             | A 3-digit hex number (where <i>nnn</i> < \$900); displays <i>nnn</i> bytes in hex format. |

ResEdit will do the appropriate type checking for you when you put the editing dialog window away.

The template mechanism is flexible enough to describe a repeating sequence of items within a resource, as in 'STR#', 'DITL', and 'MENU' resources. You can also have repeating sequences within repeating sequences, as in 'BNDL' resources. To terminate a repeating sequence, put the appropriate code in the template as follows:

```
LSTZ
...
LSTE List Zero–List End. Terminated by a 0 byte (as in 'MENU's).
ZCNT
LSTC
...
LSTE Zero Count/List Count–List End. Terminated by a zero-based count that
starts the sequence (as in 'DITL' resources).
OCNT
LSTC
...
LSTE One Count/List Count–List End. Terminated by a one-based count that
starts the sequence (as in 'STR#' resources).
LSTB
...
LSTE Ends at the end of the resource (no example exists in the given templates).
```

The "list-begin" code begins the repeating sequence of items, and the LSTE code is the end. Labels for these codes are usually set to the string "\*\*\*\*\*" Both of these codes are required.

To create your own template, follow these steps:

1. Open the ResEdit file window.
2. Open the 'TMPL' type window.
3. Choose New from the File menu.
4. Select the list separator (\*\*\*\*\*).
5. Choose New from the File menu. You may now begin entering the *label,type* pairs that define the template. Before closing the template editing window, choose Get Info from the File menu and set the name of the template to the 4-character name of your resource type.
6. Close the ResEdit file window and save changes.

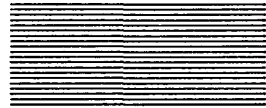
The next time you try to edit or create a resource of this new type, you'll get the dialog box in the format you have specified.

---

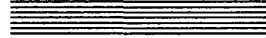
**Warning**

Changing resource templates (and hence resource type descriptions) can cause system crashes if you open older versions of a resource with a new template.

---



## Chapter 8



# Resource Compiler and Decompiler

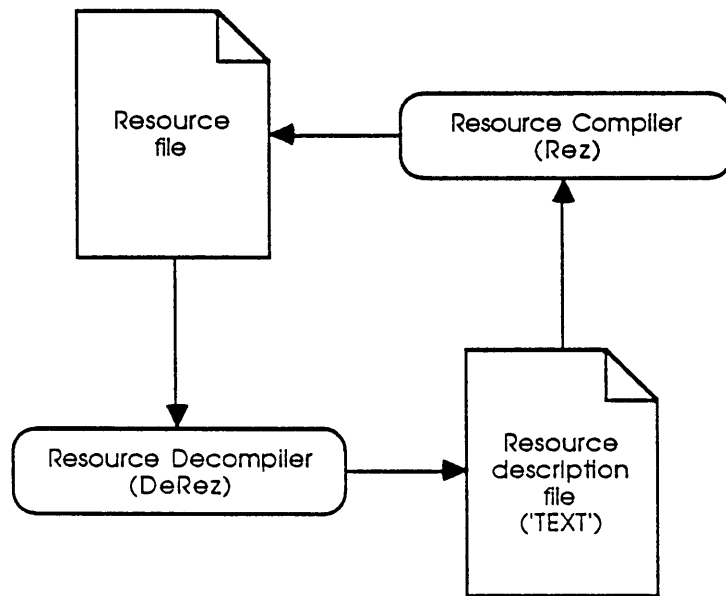
In the Macintosh Programmer's Workshop, you can build a resource graphically with ResEdit, or in text form with the Resource Compiler. This chapter explains the use of the Resource Compiler (Rez) and Resource Decompiler (DeRez). The command line syntax for Rez and DeRez is explained in Part II. General information on resource files is given in the Resource Manager chapter of *Inside Macintosh*.

---

---

## About the Resource Compiler and Decompiler

The Resource Compiler, Rez, compiles a text file (or files) called a **resource description file**, and produces a resource file as output. The Resource Decompiler, DeRez, decompiles an existing resource, producing a new resource description file that can be understood by Rez. Figure 8-1 illustrates the complementary relationship between Rez and DeRez.



**Figure 8-1**  
Rez and DeRez

The Resource Compiler can combine resources or resource descriptions from a number of files into a single resource file. The Resource Compiler also supports preprocessor directives that allow you to substitute macros, include other files, and use if-then-else constructs. (See "Preprocessor Directives" later in this chapter.)

---

### Resource Decompiler

The DeRez tool creates a textual representation of a resource file based on resource type declarations identical to those used by Rez. (If you don't specify any type declarations, the output of DeRez is in the form of raw data statements.) The output of DeRez is a resource description file that may be used as input to Rez. This file can be edited in the MPW Shell, allowing you to add comments, translate resource data to a foreign language, or specify conditional resource compilation by using the if-then-else structures of the preprocessor. You can also compare resources by using the MPW Compare command to compare resource description files.

- ❖ *Note:* MPW also includes a tool, ResEqual, which directly compares resource files. The source for ResEqual is located in the PExamples folder.



---

## Standard type declaration files

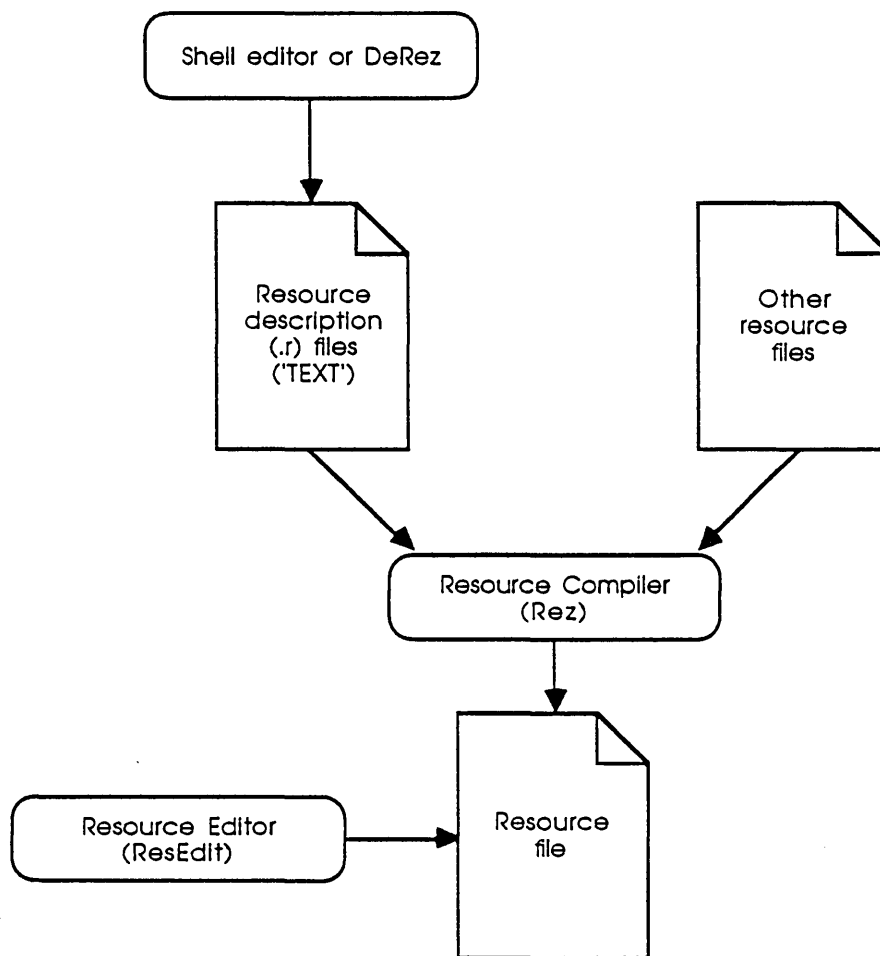
Three text files, Types.r, SysTypes.r, and MPWTypes.r, contain resource declarations for standard resource types. These files are located in the (RIncludes) directory, which is automatically searched by Rez and DeRez (that is, you can specify a file in (RIncludes) by its simple filename). These files contain definitions for the following types:

- Types.r           Type declarations for the most common Macintosh resource types ('ALRT', 'DITL', 'MENU', and so on)
- SysTypes.r        Type declarations for 'DRVr', 'FOND', 'FONT', 'FWID', 'INTL', and 'NFMT' and many others
- MPWTypes.r        Type declarations for the MPW driver type 'DRVW'

---

## Using Rez and DeRez

Rez and DeRez are primarily used to create and modify resource files. Figure 8-2 illustrates the process of creating a resource file.



**Figure 8-2**  
Creating a resource file

Rez can also form an integral part of the process of building a program. For instance, when putting together a desk accessory or driver, you'd use Rez to combine the Linker's output with other resources, to create an executable program file. (See Chapter 9 for details.)

---

---

## Structure of a resource description file

The resource description file consists of resource type declarations (which can be included from another file) followed by resource data for the declared types. Note that the Resource Compiler and Resource Decompiler have no built-in resource types—you need to define your own types, or include the appropriate “.r” files.

A resource description file may contain any number of statements, where a statement is any of the following:

|                       |                                                                                            |
|-----------------------|--------------------------------------------------------------------------------------------|
| <code>include</code>  | Include resources from another file.                                                       |
| <code>read</code>     | Read data fork of a file and include it as a resource.                                     |
| <code>data</code>     | Specify raw data.                                                                          |
| <code>type</code>     | Type declaration—declare resource type descriptions for subsequent resource statements.    |
| <code>resource</code> | Data specification—specify data for a resource type declared in a previous type statement. |

Each of these statements is described in the sections that follow.

A *type declaration* provides the pattern for any associated resource data specifications by indicating data types, alignment, size and placement of strings, and so on. You can intersperse type declarations and data in the resource description file as long as the declaration for a given resource precedes any resource statements that refer to it. An error is returned if data (that is, a resource statement) is given for a type that has not been previously defined. Whether a type was declared in a resource description file or in an include file, you can redeclare it by providing a new declaration later in a resource description file.

A resource description file can also include comments and preprocessor directives:

- **Comments** can be included anywhere where white space is allowed in a resource description file, within the comment delimiters `/*` and `*/`. Note that comments do not nest. For example, this is *one* comment:

```
/* Hello /* there */
```

- **Preprocessor directives** substitute macro definitions and include files, and provide if-then-else processing before other Rez processing takes place. The syntax of the preprocessor is very similar to that of the C-language preprocessor.

---

## Sample resource description file

An easy way to learn about the resource description format is to decompile some existing resources. For example, the following command decompiles only the 'WIND' resources in the Sample application, according to the the type declaration in {RIncludes}Types.r.

```
DeRez Sample -only WIND Types.r > DeRez.Out
```

After executing this command, DeRez.Out would contain the following decompiled resource.

```
resource 'WIND' (128, "Sample Window") {
 (64, 60, 314, 460),
 documentProc,
 visible,
 noGoAway,
 0x0,
 "Sample Window"
};
```

Note that this statement is identical to the resource description in the file Sample.r, which was originally used to build the resource. This resource data corresponds to the following type declaration, contained in Types.r:

```
type 'WIND' {
 rect; /* boundsRect */
 integerdocumentProc, dBoxProc, plainDBox, /* procID */
 altDBoxProc, noGrowDocProc,
 zoomProc=8, rDocProc=16;
 byte invisible, visible; /* visible */
 fill byte;
 byte noGoAway, goAway; /* goAway */
 fill byte;
 unsigned hex longint; /* refCon */
 pstringUntitled = "Untitled"; /* title */
};
```

Type and resource statements are explained in detail in the reference section that follows.

---

---

## Resource description statements

This section describes the syntax and use of the five types of resource description statements: `include`, `read`, `data`, `type`, and `resource`.

---

### Syntax notation

The syntax notation in this chapter follows the conventions given in the preface of this book. Additionally, the following conventions are used:

- Words that are part of the resource description language are shown in *Courier* to distinguish them from other text. The Resource Compiler is not sensitive to the case of these words.
- Punctuation characters such as commas (,), semicolons (;), and quotation marks (' and ") are to be written as shown. If one of the syntax notation characters (for example, [ or ]) must be written as a literal, it is shown enclosed by curly quotes ('...'); for example,

```
bitstring '[' length ']
```

In this case, the brackets would be typed literally—they do *not* mean that the enclosed element is optional.

- Spaces between syntax elements, constants, and punctuation are optional—they are shown for readability only.

Tokens in resource description statements may be separated by spaces, tabs, returns, or comments.

### Special terms

The following terms represent a minimal subset of the nonterminal symbols used to describe the syntax of commands in the resource description language:

| Term                 | Definition             |
|----------------------|------------------------|
| <i>resource-type</i> | <i>long-expression</i> |
| <i>resource-name</i> | <i>string</i>          |
| <i>resource-ID</i>   | <i>word-expression</i> |
| <i>ID-range</i>      | <i>ID[: ID]</i>        |

❖ *Note:* *Expression* is defined later in this chapter under “Expressions.”

A full syntax definition can be found at the end of this chapter and in Appendix D.

---

## Include—include resources from another file

The include statement lets you read resources from an existing file and include all or some of them.

### Syntax

Include statements can have the following forms:

- `include file [ resource-type ['(' resource-name | ID[:ID]')'] ] ;`  
Read the resource of type *resource-type* with the specified resource name or resource ID range in *file*. If the resource name or ID is omitted, read all resources of the type *resource-type* in *file*. If *resource-type* is omitted, read all the resources in *file*.
- `include file not resource-type ;`  
Read all resources **not** of the type *resource-type* in *file*.
- `include file resource-type1 as resource-type2 ;`  
Read all resources of type *resource-type1* and include them as resources of *resource-type2*.
- `include file resource-type1 ['(' resource-name | ID[:ID]')'  
as resource-type2 ['(' ID [, resource-name] [, attributes... ]')'] ;`  
Read the resource of type *resource-type1* with the specified name or ID range in *file*, and include it as a resource of *resource-type2* with the specified ID. You can optionally specify a resource name and resource attributes. (Resource attributes are defined below.)

Some examples follow:

```
include "otherfile"; /* include all resources from the file */
include "otherfile" 'CODE'; /* read only the CODE resources */
include "otherfile" 'CODE' (128); /* read only CODE resource 128 */
```

### AS resource description syntax

The following string variables can be used in the AS resource description to modify the resource information in include statements:

|                             |                                           |
|-----------------------------|-------------------------------------------|
| <code>\$\$Type</code>       | Type of resource from include file.       |
| <code>\$\$ID</code>         | ID of resource from include file.         |
| <code>\$\$Name</code>       | Name of resource from include file.       |
| <code>\$\$Attributes</code> | Attributes of resource from include file. |

For example, to include all DRVR resources from one file and keep the same information, but also set the SYSHEAP attribute:

```
INCLUDE "file" 'DRVR' (0:40) AS
 'DRVR' ($$ID, $$Name, $$Attributes | 64) ;
```

The `$$Type`, `$$ID`, `$$Name`, and `$$Attributes` variables are also set and legal within a normal resource statement. At any other time the values of these variables are undefined.

## Resource attributes

You can specify *attributes* as a numeric expression (as described in the Resource Manager chapter of *Inside Macintosh*), or you can set them individually by specifying one of the keywords from any of the following pairs:

| Default      | Alternative | Meaning                                                                                                                                         |
|--------------|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| appheap      | sysheap     | Specifies whether the resource is to be loaded into the application heap or the system heap.                                                    |
| nonpurgeable | purgeable   | Purgeable resources can be automatically purged by the Memory Manager.                                                                          |
| unlocked     | locked      | Locked resources cannot be moved by the Memory Manager.                                                                                         |
| unprotected  | protected   | Protected resources cannot be modified by the Resource Manager.                                                                                 |
| nonpreload   | preload     | Preloaded resources are placed in the heap as soon as the Resource Manager opens the resource file.                                             |
| unchanged    | changed     | Tells the Resource Manager whether a resource has been changed. Rez does not allow you to set this bit, but DeRez will display it if it is set. |

Bits 0 and 7 of the resource attributes are reserved for use by the Resource Manager and cannot be set by Rez, but are displayed by DeRez.

You can list more than one attribute by separating the keywords with a comma (,).

---

## Read—read data as a resource

The `read` statement lets you read a file's data fork as a resource.

### Syntax

```
read resource-type '(' ID [, resource-name] [, attributes] ')' file ;
```

### Description

Reads the data fork from *file* and writes it as a resource with the type *resource-type* and the resource ID *ID*, with the optional resource name *resource-name* and optional resource attributes (as defined in the preceding section). For example,

```
read 'STR' (-789, "Test String", SysHeap, PreLoad) "Test8";
```

---

## Data—specify raw data

Data statements let you specify raw data as a sequence of bits, without any formatting.

### Syntax

```
data resource-type '(' ID [, resource-name] [, attributes...] ')' '('
 data-string
'');
```

### Description

Reads the data found in *data-string* and writes it as a resource with the type *resource-type* and the ID *ID*. You can optionally specify a resource name, resource attributes, or both.

For example,

```
data 'PICT' (128) (
 $"4F35FF8790000000"
 $"FF234F35FF790000"
);
```

- ❖ *Note:* When DeRez generates a resource description, it uses the data statement to represent any resource type that doesn't have a corresponding type declaration or cannot be disassembled for some other reason.

---

## Type—declare resource type

A type declaration provides a template that defines the structure of the resource data for a single resource type or for individual resources. If more than one type declaration is given for a resource type, the last one read before the data definition is the one that's used—this lets you override declarations from include files or previous resource description files.

### Syntax

```
type resource-type ['(' ID-range ')'] '('
 type-specification...
'');
```

## Description

Causes any subsequent resource statement for the type *resource-type* to use the declaration { *type-specification...* }. The optional *ID-range* specification causes the declaration to apply only to a given resource ID or range of IDs.

*Type-specification* is one of the following:

|                       |                                                               |
|-----------------------|---------------------------------------------------------------|
| bitstring[ <i>n</i> ] |                                                               |
| byte                  |                                                               |
| integer               |                                                               |
| longint               |                                                               |
| boolean               |                                                               |
| char                  |                                                               |
| string                |                                                               |
| pstring               |                                                               |
| wstring               |                                                               |
| cstring               |                                                               |
| point                 |                                                               |
| rect                  |                                                               |
| fill                  | Zero fill                                                     |
| align                 | Zero fill to nibble, byte, word, or long word boundary        |
| switch                | Control construct (case statement)                            |
| array                 | Array data specification—zero or more instances of data-types |

These types can be used singly or together in a type statement. Each of these type specifiers is described in the sections that follow.

❖ *Note:* Several of these types require additional fields—the exact syntax is given in the sections that follow.

You can also declare a resource type that uses another resource's type declaration, by using the following variant of the type statement:

```
type resource-type1 ['(ID-range ')'] as resource-type2 ['(ID ')'];
```

## Data-type specifications

Data-type statements declare a field of the given data-type. They can also associate symbolic names or constant values with the data-type. Data-type specifications can take three forms, as shown in the following example:

```
type 'XAMP' (/* declare a resource of type 'XAMP' */
 byte;
 byte off=0, on=1;
 byte = 2;
);
```

- ❑ The first `byte` statement declares a byte field; the actual data is supplied in a subsequent resource statement.
- ❑ The second `byte` statement is identical to the first, except that the two symbolic names “off” and “on” are associated with the values 0 and 1. These symbolic names could be used in the resource data.
- ❑ The third `byte` statement declares a byte field whose value is always 2. In this case, no corresponding statement would appear in the resource data.
- ❖ *Note:* Numeric expressions and strings can appear in type statements; they are defined later in this chapter under “Expressions.”



**Numeric types:** The numeric types (`bitstring`, `byte`, `integer`, `longint`) are fully specified as follows:

```
[unsigned] [radix] numeric-type [=expr | symbol-definition...];
```

□ The unsigned prefix signals DeRez that the number should be displayed without a sign—that the high-order bit may be used for data and the value of the integer cannot be negative. The unsigned prefix is ignored by Rez but is needed by DeRez to correctly represent a decompiled number. Rez uses a sign if it is specified in the data. Precede a signed negative constant with a minus sign (-); `$FFFFFF85` and `-$7B` are equivalent in value.

□ *Radix* is one of the following string constants:

hex decimal octal binary literal

You can supply numeric data as decimal, octal, hexadecimal, or literal data.

□ *Numeric-type* is one of the following:

`bitstring['length']` Declare a bitstring of *length* bits (maximum 32).

`byte` Declare a byte (8-bit) field. This is the same as `bitstring[8]`.

`integer` Integer (16-bit) field. This is the same as `bitstring[16]`.

`longint` Long integer (32-bit) field. This is the same as `bitstring[32]`.

Rez uses integer arithmetic and stores numeric values as integer numbers. Rez translates booleans, bytes, integers, and longints to bitstring equivalents. All computations are done in 32 bits and truncated.

An error is generated if a value won't fit in the number of bits defined for the type. The valid ranges for values of `byte`, `integer`, and `longint` constants are as follows:

| Type                 | Maximum    | Minimum     |
|----------------------|------------|-------------|
| <code>byte</code>    | 255        | -128        |
| <code>integer</code> | 65535      | -32768      |
| <code>longint</code> | 4294967295 | -2147483648 |

**Boolean type:** A Boolean is a single bit with two possible states: 0 (or false) and 1 (or true). (True and false are global predefined identifiers.) Boolean values are declared as follows:

```
boolean [= constant | symbolic-value...];
```

Type `boolean` declares a 1-bit field; this is equivalent to

```
unsigned bitstring[1]
```

Note that this type is not the same as a Boolean variable as defined by Pascal.

**Character type:** Characters are declared as follows:

```
char [= string | symbolic-value...];
```

Type `char` declares an 8-bit field (this is the same as writing `string[1]`).

Here is an example:

```
type 'SYMB' {
 char dollar = "$", percent = "%";
};
resource 'SYMB' (128) {
 dollar
};
```

**String type:** String data types are specified as follows:

```
string-type ['(' length ')'] [= string | symbol-value...];
```

*String-type* is one of the following:

|              |                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [hex] string | Plain string (no length indicator or termination character is generated). The optional hex prefix tells DeRez to display it as a hex-string. <code>String[n]</code> contains <i>n</i> characters and is <i>n</i> bytes long. Type <code>char</code> is shorthand for <code>string[1]</code> .                                                                                    |
| pstring      | Pascal string (a leading byte containing the length information is generated). <code>pstring[n]</code> contains <i>n</i> characters and is <i>n</i> +1 bytes long. <code>pstring</code> has a built-in maximum length of 255 characters, the highest value the length byte can hold. If the string is too long to fit the field, a warning is given and the string is truncated. |
| wstring      | Word string is a very large pstring. Its length is stored in the first two bytes. Therefore, a <code>wstring</code> can contain up to 65,535 characters. <code>Wstring[n]</code> contains <i>n</i> characters and is <i>n</i> +2 bytes long.                                                                                                                                     |
| cstring      | C string (a trailing null byte is generated). <code>Cstring[n]</code> contains <i>n</i> -1 characters and is <i>n</i> bytes long. A <code>cstring</code> of length 1 can be assigned only the value "", because <code>cstring[1]</code> has room only for the terminating null.                                                                                                  |

Each may be followed by an optional *length* indicator in brackets (`[n]`). *Length* is an expression indicating the string length in bytes. *Length* is a positive number in the range  $1 \leq \textit{length} \leq 2147483647$  for `string` and `cstring`, and in the range  $1 \leq \textit{length} \leq 255$  for `pstring`, and in the range  $1 \leq \textit{length} \leq 65535$  for `wstring`.

❖ *Note:* You cannot assign the value of a literal to a string-type.

If no length indicator is given, a `pstring`, `wstring`, or `cstring` stores the number of characters in the corresponding data definition. If a length indicator is given, the data may be truncated on the right or padded on the right. The padding characters for all string types are nulls. If the data contains more characters than the length indicator provides for, the string is truncated and a warning message is given.

---

### Warning

A null byte within a `cstring` is a termination indicator and may confuse DeRez and C programs. However, the full string, including the explicit null and any text that follows it, will be stored by Rez as input.

---

**Point and rectangle types:** Because points and rectangles appear so frequently in resource files, they have their own simplified syntax:

```
point [= point-constant | symbolic-value...];
```

```
rect [= rect-constant | symbolic-value...];
```

where

```
point-constant = ('x-integer-expr, y-integer-expr')
```

and

```
rect-constant = ('integer-expr, integer-expr, integer-expr, integer-expr')
```

These type-statements declare a point (two 16-bit signed integers) or a rectangle (four 16-bit signed integers). The integers in a rectangle definition specify the rectangle's upper-left and lower-right points, respectively.

## Fill and align types

The resource created by a resource definition has no implicit alignment. It's treated as a bit stream, and integers and strings can start at any bit. The `fill` and `align` type specifiers are two ways of padding fields so that they begin on a boundary that corresponds to the field type. `align` is automatic and `fill` is explicit. Fill and alignment generate zero-filled fields.

**Fill specification:** The `fill` statement causes Rez to add the specified number of bits to the data stream. The fill is always 0. The form of the statement is

```
fill fill-size ['length'];
```

where *fill-size* is one of the following strings:

```
bit nibble byte word long
```

These declare a fill of 1, 4, 8, 16, or 32 bits (optionally multiplied by the *length* modifier). *Length* is an expression  $\leq 2147483647$ .

The following `fill` statements are equivalent:

```
fill word(2);
```

```
fill long;
```

```
fill bit(32);
```

The full form of a type statement specifying a fill might be as follows:

```
type 'XRES' (data-type specifications; fill bit(2););
```

❖ *Note:* Rez supplies zeros as specified by `fill` and `align` statements. DeRez does not supply any values for `fill` or `align` statements; it just skips the specified number of bits, or until data is aligned as specified.

**Align specification:** Alignment causes Rez to add fill bits of zero value until the data is aligned at the specified boundary. An alignment statement takes the following form:

```
align align-size ;
```

where *align-size* is one of the following strings:

```
nibble byte word long
```

Alignment pads with zeros until data is aligned on a 4-, 8-, 16-, or 32-bit boundary. This alignment affects all data from the point where it is specified until the next `align` statement.

## Array type

An array is declared as follows:

```
[wide] array [array-name | ['length']] (' array-list ');
```

The *array-list*, a list of type specifications, is repeated zero or more times. The wide option outputs the array data in a wide display format (in DeRez)—the elements that make up the array-list are separated by a comma and space instead of a comma, return, and tab. Either *array-name* or [*length*] may be specified. *Array-name* is an identifier.

If the array is named, then a preceding statement must refer to that array in a constant expression with the `$$countof(array-name)` function; otherwise DeRez will be unable to decompile resources of this type. For example,

```
type 'STR#' (/* define a string list resource */
 integer = $$countof(StringArray);
 array StringArray {
 pstring;
 };
};
```

The `$$countof` function returns the number of array elements (in this case, the number of strings) from the resource data.

If [*length*] is specified, there must be exactly *length* elements.

## Switch type

The switch statement specifies a number of case statements for a given field or fields in the resource. The format is as follows:

```
switch (' case-statement... ');
```

where a *case-statement* has the following form:

```
case case-name : [case-body ;]...
```

*Case-name* is an identifier. *Case-body* may contain any number of type specifications and must include a single constant declaration per case, in the following form:

```
key data-type = constant
```

Which case applies is based on the key value. For example,

```
type 'DITL' (/* dialog item list declaration from Types.r */
 ...type specifications...
 switch (/* one of the following */
 case Button:
 boolean enabled, disabled;
 key bitstring[7] = 4; /* key value */
 pstring;
 case CheckBox:
 boolean enabled, disabled;
 key bitstring[7] = 5; /* key value */
 pstring;
 etc.
);
};
```

## Sample type statement

The following sample type statement is the standard declaration for a 'WIND' resource, taken from the Types.r file:

```
type 'WIND' {
 rect; /* boundsRect */
 integer documentProc, dBoxProc, plainDBox, /* procID */
 altDBoxProc, noGrowDocProc,
 zoomProc=8, rDocProc=16;
 byte invisible, visible; /* visible */
 fill byte;
 byte noGoAway, goAway; /* has close box*/
 fill byte;
 unsigned hex longint; /* refCon */
 pstring Untitled = "Untitled"; /* title */
};
```

The type declaration consists of header information followed by a series of statements, each terminated by a semicolon (;). The header of the sample window declaration is

```
type 'WIND'
```

The header begins with the `type` keyword followed by the name of the resource type being declared—in this case, a window. You may specify a standard Macintosh resource type, as shown in the Resource Manager chapter of *Inside Macintosh*, or you may declare a resource type specific to your application.

The left brace ({}). introduces the body of the declaration. The declaration continues for as many lines as necessary until a matching right brace (}) is encountered. You can write more than one statement on a line, and a statement may be on more than one line (like the `integer` statement above). Each statement represents a field in the resource data. Recall that comments may appear anywhere where white space may appear in the resource description file; comments begin with `/*` and end with `*/` as in C.

**An aside—symbol definitions:** Symbolic names for data-type fields simplify the reading and writing of resource definitions. Symbol definitions have the form

```
name = value [, name = value]...
```

For numeric data, the “`= value`” part of the statement can be omitted. If a sequence of values consists of consecutive numbers, the explicit assignment can be left out—if *value* is omitted, it's assumed to be one greater than the previous value. (The value is assumed to be zero if it's the first value in the list.) This is true for bitstrings (and their derivatives, `byte`, `integer`, and `longint`). For example,

```
integer documentProc, dBoxProc, plainDBox,
 altDBoxProc, noGrowDocProc,
 zoomProc=8, rDocProc=16;
```

In this example, the symbolic names `documentProc`, `dBoxProc`, `plainDBox`, `altDBoxProc`, and `noGrowDocProc` are automatically assigned the numeric values 0, 1, 2, 3, and 4.

Memory is the only limit to the number of symbolic values that can be declared for a single field. There is also no limit to the number of names you can assign to a given value; for example,

```
integer documentProc=0, dBoxProc=1, plainDBox=2, altDBoxProc=3,
 rDocProc=16,
 Document=0, Dialog=1, DialogNoShadow=2, ModelessDialog=3,
 DeskAccessory=16;
```

---

## Resource—specify resource data

Resource statements specify actual resources, based on previous type declarations.

### Syntax

```
resource resource-type '(' ID [, resource-name] [, attributes] ')' '('
 [data-statement [, data-statement]...]
'');
```

### Description

Specifies the data for a resource of type *resource-type* and ID *ID*. The latest type declaration declared for *resource-type* is used to parse the data specification. *Data-statements* specify the actual data; *data-statements* appropriate to each resource type are defined in the next section.

The resource definition causes an actual resource to be generated. A resource statement can appear anywhere in the resource description file, or even in a separate file specified on the command line or as an #include file, as long as it comes after the relevant type declaration.

### Data statements

The body of the data specification contains one data statement for each declaration in the corresponding type declaration. The base type must match the declaration.

| Base type | Instance types                             |
|-----------|--------------------------------------------|
| string    | string, cstring, pstring, wstring, char    |
| bitstring | boolean, byte, integer, longint, bitstring |
| rect      | rect                                       |
| point     | point                                      |

**Switch data:** Switch data statements are specified by using the following format:

```
switch-name data-body
```

For example, the following could be specified for the 'DITL' type given earlier:

```
...
CheckBox (enabled, "Check here"),
...
```

**Array data:** Array data statements have the following format:

```
(' [array-element [, array-element]...])'
```

where an *array-element* consists of any number of data statements separated by commas.

For example, the following data might be given for the 'STR#' resource defined earlier:

```
resource 'STR#' (280) {
 {
 "this",
 "is",
 "a",
 "test"
 }
};
```

## Sample resource definition

This section describes a sample resource description file for a window. (See the Window Manager chapter of *Inside Macintosh* for information about resources in windows.)

Here, again, is the type declaration given above under "Sample Type Statement":

```
type 'WIND'{
 rect; /* boundsRect */
 integer documentProc, dBoxProc, plainDBox, /* procID */
 altDBoxProc, noGrowDocProc,
 zoomProc=8, rDocProc=16;
 byte invisible, visible; /* visible */
 fill byte;
 byte noGoAway, goAway; /* has close box */
 fill byte;
 unsigned hex longint; /* refCon */
 pstring Untitled = "Untitled"; /* title */
};
```

Here is a typical example of the window data corresponding to this declaration:

```
resource 'WIND' (128,"My window",appheap,preload) { /* Status report window */
 {40,80,120,300}, /* Bounding rectangle */
 documentProc, /* documentProc etc.. */
 Visible, /* Visible or Invisible */
 goAway, /* GoAway or NoGoAway */
 0, /* Reference value RefCon */
 "Status Report" /* Title */
};
```

This data definition declares a resource of type 'WIND', using whatever type declaration was previously specified for 'WIND'. The resource ID is 128; the resource name is "My window". Because the resource name is represented by the Resource Manager as a pstring, it should not contain more than 255 characters. The resource name may contain any character including the null character (\$00). The resource will be placed in the application heap when loaded, and it will be loaded when the resource file is opened.

The first statement in the window type declaration declares a bounding rectangle for the window:

```
rect;
```

The rectangle is described by two points: the upper-left corner and the lower-right corner. The points of a rectangle are separated by commas as follows:

```
{ top, left, bottom, right }
```

An example of data for these coordinates is

```
{ 40, 80, 120, 300 }
```

**Symbolic names:** Symbolic names may be associated with particular values of a numeric type. Notice that a symbolic name is given for the data in the second, third, and fourth fields of the window declaration. For example,

```
integer documentProc=0, dBoxProc=1, plainDBox=2,
 altDBoxProc=3, noGrowDocProc=4,
 zoomProc=8, rDocProc=16; /* windowType */
```

This statement specifies a signed 16-bit integer field with symbolic names associated with the values 0 to 4 and 16. The values 0 through 4 need not be indicated in this case; if no values are given, symbolic names are automatically given values starting at zero, as explained previously.

In the sample window declaration, we gave the values `true` (1) and `false` (0) to two different byte variables. For clarity, we used those symbolic names in the window's resource data; that is,

```
visible,
goAway,
```

instead of their equivalents

```
TRUE,
TRUE,
```

or

```
1,
1,
```



---

---

## Preprocessor directives

Preprocessor directives substitute macro definitions and include files, and provide if-then-else processing, before other Rez processing takes place.

The syntax of the preprocessor is very similar to that of the C-language preprocessor. Each of the preprocessor statements must be expressed on a single line, beginning on a new line and terminated by a return character. *Identifiers* (used in macro names) may be letters (A-Z, a-z), digits (0-9), or the underscore character ( \_ ). Identifiers may not start with a digit. Identifiers are not case sensitive. An identifier may be any length.

---

## Variable definitions

The #define and #undef directives let you assign values to identifiers:

```
#define macro data
#undef macro
```

The #define directive causes any occurrence of the identifier *macro* to be replaced with the text *data*. You can extend a macro over several lines by ending the line with the backslash character (\), which functions as the Rez escape character. For example,

```
#define poem "I wander \
thro\' each \
charter\'d street"
```

(Quotation marks within strings must also be escaped.)

#undef removes the previously defined identifier *macro*. Macro definitions can also be removed with the **-undef** option on the Rez command line.

The following predefined macros are provided:

| Variable | Value |
|----------|-------|
| true     | 1     |
| false    | 0     |

---

## Include directives

The #include directive reads a text file:

```
#include file
```

Include the text file *file*. The maximum nesting is to 10 levels.

For example,

```
#include $$Shell("MPW") "MyProject:MyTypes.r"
```

Note that the #include preprocessor directive (which includes a file) is different from the previously described include statement, which copies resources from another file.

---

## If-Then-Else processing

These directives provide conditional processing:

```
#if expression
[#elif expression]
[#else]
#endif
```

❖ *Note: Expression* is defined later in this chapter; with the #if and #elif directives, *expression* may also include this expression:

```
defined identifier or defined '(' identifier ')'
```

The following may also be used in place of #if:

```
#ifdef macro
#endif macro
```

For example,

```
#define Thai

Resource 'STR ' (199) {
#ifdef English
 "Hello"
#elif defined (French)
 "Bonjour"
#elif defined (Thai)
 "Sawati"
#elif defined (Japanese)
 "Konnichiwa"
#endif
};
```

---

## Print directive

The #printf directive is provided to aid in debugging resource description files:

```
#printf(formatString, arguments...)
```

The format of the #printf statement is exactly the same as the printf statement in the C language, with one exception: There can be no more than 20 arguments. This is the same restriction that applies to the \$\$Format function. The #printf directive writes its output to diagnostic output. Note that the #printf directive *does not* end with a semicolon.

For Example,:

```
#define Tuesday 3
#ifdef Monday
#printf("The day is Monday, day #%d\n", Monday)
#elif defined(Tuesday)
#printf("The day is Tuesday, day #%d\n", Tuesday)
#elif defined(Wednesday)
#printf("The day is Wednesday, day #%d\n", Wednesday)
#elif defined(Thursday)
#printf("The day is Thursday, day #%d\n", Thursday)
#else
#printf("DON'T KNOW WHAT DAY IT IS!\n")
#endif
```

The above file will generate this text:

```
The day is Thursday, day #3
```

---

---

## Resource description syntax

This section describes the details of the resource description syntax. For a complete summary definition, see Appendix D.

---

### Numbers and literals

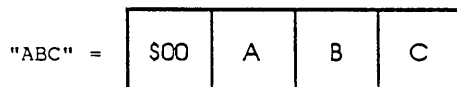
All arithmetic is performed as 32-bit signed arithmetic. The basic constants are

|         |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|---------|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Decimal | <i>mmm...</i>   | Signed decimal constant between 4294967295 and -2147483648.                                                                                                                                                                                                                                                                                                                                                                                                           |
| Hex     | <i>0xhhh...</i> | Signed hexadecimal constant between 0X7FFFFFFF and 0X80000000.                                                                                                                                                                                                                                                                                                                                                                                                        |
|         | <i>\$hhh...</i> | Alternate form for hexadecimal constants.                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Octal   | <i>0ooo...</i>  | Signed octal constant between 01777777777 and 020000000000.                                                                                                                                                                                                                                                                                                                                                                                                           |
| Binary  | <i>0Bbbb...</i> | Signed binary constant between 0B11111111111111111111111111111111 and 0B10000000000000000000000000000000.                                                                                                                                                                                                                                                                                                                                                             |
| Literal | <i>'aaaa'</i>   | A literal can contain one to four characters. Characters are printable ASCII characters or escape characters. If there are fewer than four characters in the literal, then the characters to the left (high bits) are assumed to be \$00. Characters that are not in the printable character set, and are not the characters \ ' and \\ (which have special meanings), can be escaped according to the character escape rules. (See "Strings" later in this section.) |

Literals and numbers are treated in the same way by the Resource Compiler. A **literal** is a value within single quotation marks; for instance, 'A' is a number with the value 65; on the other hand, "A" is the character A expressed as a string. Both are represented in memory by the bitstring 01000001. (Note, however, that "A" is not a valid number and 'A' is not a valid string.) The following numeric expressions are all equivalent:

```
'B'
66
'A'+1
```

Literals are padded with nulls on the left side so that the literal 'ABC' is stored as shown in Figure 8-3.



**Figure 8-3**  
Padding of literals

---

## Expressions

An expression can consist of simply a number or literal. Expressions can also include numeric variables and the system functions:

`$$countof '(' array-name ')'`

\$\$ID expressions can include the expression operators listed in Table 8-1. Table 8-1 lists the operators in order of precedence with highest precedence first—groupings indicate equal precedence. Evaluation is always left to right when the priority is the same. Variables are defined following the table.

**Table 8-1**  
Resource description expression operators

---

| Operator                                                                                                                      | Meaning                                                                                                                                              |
|-------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. ( <i>expr</i> )                                                                                                            | Parentheses can be used in the normal manner to force precedence in expression calculation.                                                          |
| 2. - <i>expr</i><br>~ <i>expr</i><br>! <i>expr</i>                                                                            | Arithmetic (two's complement) negation of <i>expr</i> .<br>Bitwise (one's complement) negation of <i>expr</i> .<br>Logical negation of <i>expr</i> . |
| 3. <i>expr1</i> * <i>expr2</i><br><i>expr1</i> / <i>expr2</i><br><i>expr1</i> % <i>expr2</i>                                  | Multiplication.<br>Division.<br>Remainder from dividing <i>expr1</i> by <i>expr2</i> .                                                               |
| 4. <i>expr1</i> + <i>expr2</i><br><i>expr1</i> - <i>expr2</i>                                                                 | Addition.<br>Subtraction.                                                                                                                            |
| 5. <i>expr1</i> << <i>expr2</i><br><i>expr1</i> >> <i>expr2</i>                                                               | Shift left—shift <i>expr1</i> left by <i>expr2</i> bits.<br>Shift right—shift <i>expr1</i> right by <i>expr2</i> bits.                               |
| 6. <i>expr1</i> > <i>expr2</i><br><i>expr1</i> >= <i>expr2</i><br><i>expr1</i> < <i>expr2</i><br><i>expr1</i> <= <i>expr2</i> | Greater than.<br>Greater than or equal to.<br>Less than.<br>Less than or equal to.                                                                   |
| 7. <i>expr1</i> == <i>expr2</i><br><i>expr1</i> != <i>expr2</i>                                                               | Equal.<br>Not equal.                                                                                                                                 |
| 8. <i>expr1</i> & <i>expr2</i>                                                                                                | Bitwise AND.                                                                                                                                         |
| 9. <i>expr1</i> ^ <i>expr2</i>                                                                                                | Bitwise XOR.                                                                                                                                         |
| 10. <i>expr1</i>   <i>expr2</i>                                                                                               | Bitwise OR.                                                                                                                                          |
| 11. <i>expr1</i> && <i>expr2</i>                                                                                              | Logical AND.                                                                                                                                         |
| 12. <i>expr1</i>    <i>expr2</i>                                                                                              | Logical OR.                                                                                                                                          |

The logical operators `!`, `>`, `>=`, `<`, `<=`, `==`, `!=`, `&&`, `||` evaluate to 1 (true) or 0 (false).

---

## Variables

There are some Resource Compiler variables that contain commonly used values. All Resource Compiler variables start with \$\$ followed by an alphanumeric identifier.

The following variables have string values (typical values are given in parentheses):

|                                                                                          |                                                                                                                                                                         |
|------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| \$\$Version                                                                              | Version number of the Resource Compiler. ("V2.0")                                                                                                                       |
| \$\$Date                                                                                 | Current date. Useful for putting timestamps into the resource file. The format is generated through the ROM call IUDateString. ("Thursday, May 20, 1987")               |
| \$\$Time                                                                                 | Current time. Useful for timestamping the resource file. The format is generated through the ROM call IUTimeString. ("7:50:54 AM")                                      |
| \$\$Shell (" <i>stringExpr</i> ")                                                        | Current value of the exported Shell variable ( <i>stringExpr</i> ). Note that the curly braces must be omitted, and the double quotes must be present.                  |
| \$\$Resource (" <i>filename</i> ", ' <i>type</i> ', <i>ID</i>   " <i>resourceName</i> ") | Reads the resource ' <i>type</i> ' with the ID <i>ID</i> or the name " <i>resourceName</i> " from the resource file " <i>filename</i> ", and returns a string.          |
| \$\$Format (" <i>formatString</i> ", <i>arguments</i> )                                  | Works just like the #printf directive except that \$\$Format returns a string rather than printing to standard output. (See "Print Directive" earlier in this chapter.) |

The following string variables can be used in the AS resource description to modify the resource information in include statements:

|                |                                           |
|----------------|-------------------------------------------|
| \$\$Type       | Type of resource from include file.       |
| \$\$ID         | ID of resource from include file.         |
| \$\$Name       | Name of resource from include file.       |
| \$\$Attributes | Attributes of resource from include file. |

For example, to include all DRVr resources from one file and keep the same information, but also set the SYSHEAP attribute:

```
INCLUDE "file" 'DRVr' (0:40) AS
 'DRVr ($$ID, $$Name, $$Attributes | 64) ;
```

The \$\$Type, \$\$ID, \$\$Name, and \$\$Attributes variables are also set and legal within a normal resource statement. At any other time the values of these variables are undefined.

The following variables have numeric values:

|             |                                                                |
|-------------|----------------------------------------------------------------|
| \$\$Hour    | Current hour. Range 0–23.                                      |
| \$\$Minute  | Current minute. Range 0–59.                                    |
| \$\$Second  | Current second. Range 0–59.                                    |
| \$\$Year    | Current year.                                                  |
| \$\$Month   | Current month. Range 1–12.                                     |
| \$\$Day     | Current day. Range 1–31.                                       |
| \$\$Weekday | Current day of the week. Range 1–7 (that is, Sunday–Saturday). |

---

## Strings

There are two basic types of strings:

- Text string    `"a..."`    The string can contain any printable character except `'` `"` `'` and `\`. These and other characters can be created through escape sequences. (See Table 8-2.) The string `"` is a valid string of length 0.
- Hex string    `$$hh..."`    Spaces and tabs inside a hexadecimal string are ignored. There must be an even number of hexadecimal digits. The string `$$"` is a valid hexadecimal string of length 0.

Any two strings (hexadecimal or text) will be concatenated if they are placed next to each other with only white space in between. (In this case, returns and comments are considered white space.)

Figure 8-4 shows a Pascal string declared as

```
pstring [10];
whose data definition is
"Hello"
```

|      |   |   |   |   |   |      |      |      |      |      |
|------|---|---|---|---|---|------|------|------|------|------|
| \$05 | H | e | l | l | o | \$00 | \$00 | \$00 | \$00 | \$00 |
|------|---|---|---|---|---|------|------|------|------|------|

**Figure 8-4**  
Internal representation of a Pascal string

In the input file, string data is surrounded by double quotation marks (`"`). You can continue a string on the next line. A separating token (for example, a comma) or brace signifies the end of the string data. A side effect of string continuation is that a sequence of two quotation marks (`""`) is simply ignored. For example,

```
"Hello ""out "
"there."
```

is the same string as

```
"Hello out there.";
```

To place a quotation mark in a string, precede the quotation mark with a backslash (`\`).

## Escape characters

The backslash character (\) is provided as an escape character to allow you to insert nonprintable characters in a string. For example, to include a return character in a string, use the escape sequence

```
\r
```

Valid escape sequences are given in Table 8-2.

**Table 8-2**  
Resource Compiler escape sequences

| Escape sequence | Name         | Hex value | Printable equivalent |
|-----------------|--------------|-----------|----------------------|
| \t              | Tab          | \$09      | None                 |
| \b              | Backspace    | \$08      | None                 |
| \r              | Return       | \$0D      | None                 |
| \n              | Newline*     | \$0D      | None                 |
| \f              | Form feed    | \$0C      | None                 |
| \v              | Vertical tab | \$0B      | None                 |
| \?              | Rubout       | \$7F      | None                 |
| \\              | Backslash    | \$5C      | \                    |
| \'              | Single quote | \$3A      | '                    |
| \"              | Double quote | \$22      | "                    |

\*On the Macintosh, newline is identical to return.

You can also use octal, hexadecimal, decimal, and binary escape sequences to specify characters that do not have predefined escape equivalents. The forms are as follows:

| Base | Form       | Number of digits | Example     |
|------|------------|------------------|-------------|
| 2    | \0Bbbbbbbb | 8                | \0B01000001 |
| 8    | \ooo       | 3                | \101        |
| 10   | \0Dddd     | 3                | \0D065      |
| 16   | \0Xhh      | 2                | \0X41       |
| 16   | \\$hh      | 2                | \\$41       |

Some examples are

```
\077 /* 3 octal digits */
\0xFF /* '0x' plus 2 hex digits */
\0xF1\0xF2\0xF3 /* '$' plus 2 hex digits
\0d099 /* '0d' plus 3 decimal digits */
```

❖ *Note to C programmers:* An octal escape code consists of exactly three digits. For instance, to place an octal escape code with a value of 7 in the middle of an alphabetic string, write AB\007CD, not AB\7CD.

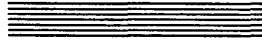
You can use the DeRez command line option `-e` to print characters that would otherwise be escaped (characters preceded by a backslash, for example). Normally, only characters with values between \$20 and \$D8 are printed as Macintosh characters. With this option, however, all characters (except null, newline, tab, backspace, formfeed, vertical tab, and rubout) will be printed as characters, not as escape sequences.







## Chapter 9



**Building an Application,  
a Desk Accessory,  
or an MPW Tool**

---

---

## Overview of the build process

This chapter describes the mechanics of building a program—the steps involved are nearly the same for applications, MPW tools, desk accessories, and drivers. All programmers should read the opening sections of this chapter, which explain the entire build process for an application, the standard case. Later sections explain what's different about building an MPW tool, desk accessory, or driver.

Those new to MPW should first read “Building a Program: An Introduction” in Chapter 2. This brief introduction takes you through the steps of using the Directory and Build menus to build a program.

Building a program consists of the following steps:

1. *Create source files and compile them.* Source files are compiled or assembled to produce object files. (For information on writing programs in Pascal, C, or assembly language, and including the proper interface or include files, see the appropriate MPW language manual.)
  2. *Create additional resources with ResEdit or Rez.* If your program requires any additional resources (other than code resources), you can create them by using the resource editor (ResEdit) or Resource Compiler (Rez). See Chapters 7 and 8 for detailed information.
  3. *Create the final executable file with Link.* The object files are linked together, along with any needed library routines, into either a new resource file or an existing one (replacing the 'CODE', 'DRVr', or other executable resources).
- ❖ *Note:* For building a desk accessory or driver in Pascal or C, an additional step is required—run Rez to create the final 'DRVr' resource. For details, see “Building a Desk Accessory or Driver,” later in this chapter.

Figure 9-1 illustrates the complete process.

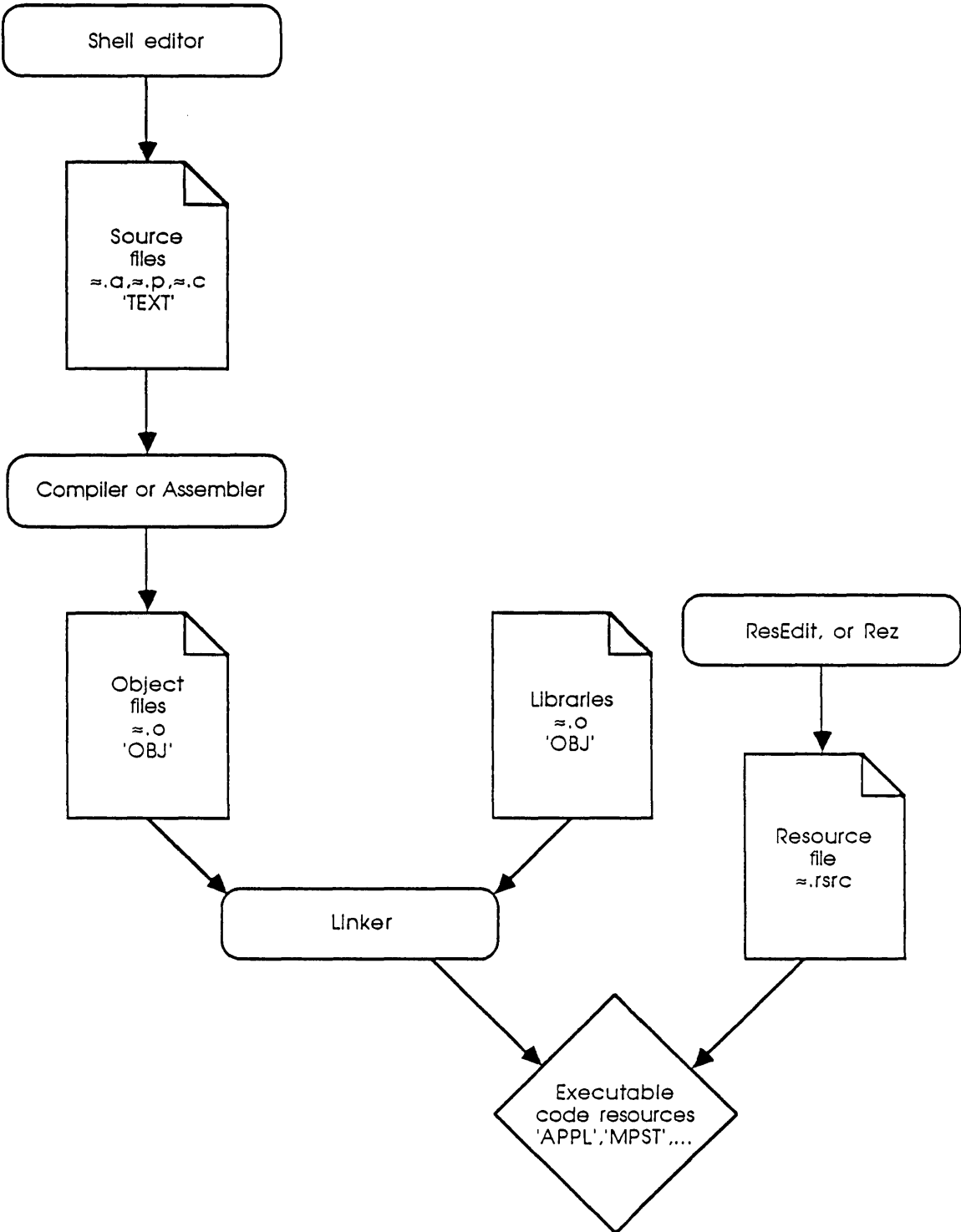


Figure 9-1  
Building a program

For example, the following series of commands compile, "Rez," and link the sample Pascal application Sample.p:

```
Pascal Sample.p
Rez Sample.r -o Sample
Link Sample.p.o @
 {(Libraries)"Interface.o @
 {(Libraries)"Runtime.o @
 {(PLibraries)"Paslib.o @
 -o Sample
```

This process is usually automated by using the Make tool. (See the sample makefiles in the Examples folders, and "Using Make" in Chapter 10.)

❖ *Note:* If you build an application with customized icons for documents (that is, a 'BNDL' resource for bundling 'ICN#' and 'FREF' resources), then you need to use SetFile to set your application's bundle bit:

```
SetFile -a B MyApp
```

See the Finder Interface chapter of *Inside Macintosh* for information.

---

---

## The structure of a Macintosh application

Macintosh files have two forks: a resource fork and a data fork. The **resource fork** contains a number of resources. The **data fork** may contain anything the application puts there. On the Macintosh, a program is a file whose resource fork contains code resources ('CODE' or other executable resources), and in most cases additional resources containing strings, dialogs, menus, and the like. The code resources for applications and tools must contain a main program (an execution starting point). Desk accessories and drivers, by contrast, don't require a main program, but contain collections of routines that are called individually when the desk accessory or driver is used.

The simplest possible application has two resources in the resource fork and nothing in the data fork. The first resource is a 'CODE' resource with ID = 0. (The Linker creates this resource, which contains the jump table and information about the application's use of parameter and global space.) The second resource is a 'CODE' resource with ID = 1, which contains the application's code segment. For more information, see the Segment Loader chapter of *Inside Macintosh*.

---

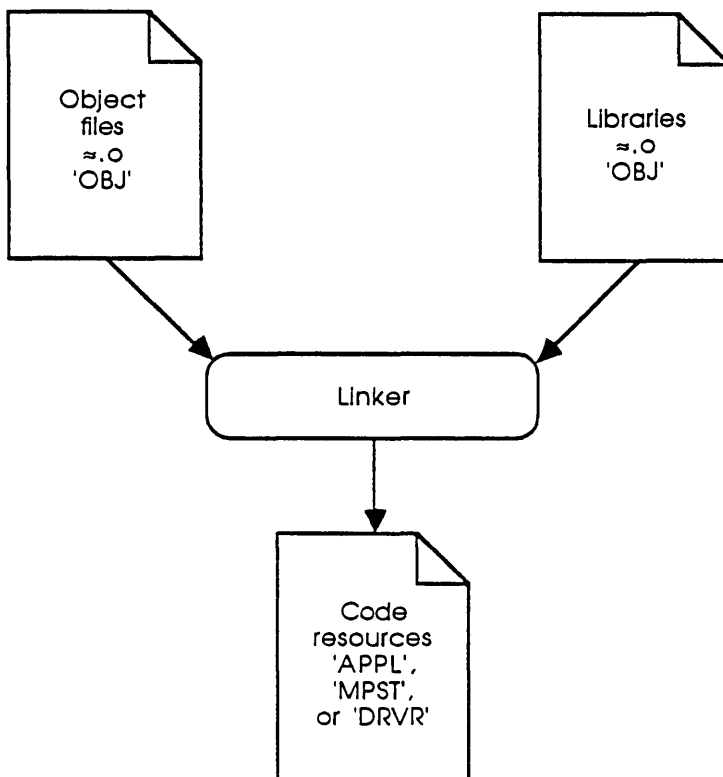
---

## Linking

This section describes how to link an application, MPW tool, desk accessory, or driver.

❖ *Note:* For more information about Linker functions, see “More About Linking” in Chapter 10. The Link command itself is described in Part II. The MPW object file format is described in Appendix F.

The Linker links object files into an application, MPW tool, desk accessory, driver, or other executable resource. The Linker's output is an executable object file. The Linker links together the compiled or assembled object files, along with any needed library routines, into either an existing resource file (replacing the 'CODE', 'DRVR', or other executable resources) or a new one (Figure 9-2).



**Figure 9-2**  
Linking

The Linker resolves all symbolic references, and also controls final program segmentation. A related tool, Lib, provides facilities for modifying and combining object files (libraries).

The Linker's default action is to link an application (type APPL, creator "????"), placing the output segments into 'CODE' resources. You can set a file's type and creator with Link's -t and -c options. (See “File Types and Creators” below.)

---

## What to link with

Applications, tools, and desk accessories should be linked with the libraries listed in Table 9-1. It's wise to link new programs with all of the libraries that might be needed. If unnecessary files are specified, the Linker will display warnings indicating that they can be removed from your build instructions.

**Table 9-1**  
Files to link with

---

### *Inside Macintosh* interfaces

{Libraries}Interface.o

### Runtime support

Link with *one* of the following:

{Libraries}Runtime.o      If no part of the program is written in C  
{CLibraries}CRuntime.o    If any part of the program is written in C

### Pascal libraries

{PLibraries}PasLib.o      Pascal language library  
{PLibraries}SANELib.o    SANE numerics library

### C libraries

{CLibraries}CInterface.o    Macintosh interface for C  
{CLibraries}CSANELib.o    SANE numerics library  
{CLibraries}Math.o        Math functions  
{CLibraries}StdCLib.o      Standard C Library

### Specialized libraries

You may also call routines in the following libraries:

{Libraries}ObjLib.o        Object-oriented programming (Pascal and Assembler)  
{Libraries}ToolLibs.o      Routines for MPW tools

### Desk accessories

{Libraries}DRVRRuntime.o    Driver runtime library

For details about linking tools and desk accessories, refer to "Linking a Tool" and "Linking a Desk Accessory or Driver" later in this chapter.

---

## Linking multilingual programs

When you link programs that use libraries from more than one language, the Linker may detect several duplicate entry points. Normally it doesn't matter which of the duplicate copies of a particular routine get linked with your program. (You can use the Linker's *-w* option to suppress the duplicate definitions warnings.)

However, programs written partly in C and partly in assembly language or Pascal require special precautions. When you link C code with other languages, link with the file CRuntime.o and *not* with Runtime.o. If execution is expected to begin with the C function `main()`, no special action is necessary. However, if your main program is written in assembly language or Pascal, but part of your program is written in C, the object file containing your main program must appear *before* CRuntime.o in the list of object files passed to the Linker.

---

---

## File types and creators

When you execute a command, the Shell determines how to run it based on its file type. Files of type APPL are considered applications and are run as if launched from the Finder. Files of type MPST are considered MPW tools and are run within the Shell environment. Files of type TEXT are taken to be scripts and are interpreted by the Shell. An attempt to run a file of any other type produces an error message. Table 9-2 summarizes file types and creators.

**Table 9-2**  
File types and creators

| Type of program | Type  | Creator    |
|-----------------|-------|------------|
| Application     | APPL  | <i>any</i> |
| MPW tool        | MPST  | 'MPS '     |
| Desk accessory  | DFILD | MOV        |
| Script          | TEXT  | <i>any</i> |

- ❖ *Note:* Each application has its own unique creator (or **signature**)—see the Finder Interface chapter of *Inside Macintosh*. For example, creating a file with the type DFIL and creator DMOV, tells the Font/DA Mover that this file contains desk accessories.

You can set a file's type and creator with the `-t` and `-c` options to `Link`, `Rez`, or `SetFile`.

---

---

## Building a desk accessory or driver

A **desk accessory** is a 'DRVW' resource whose resource name begins with a null character (\$00), and that resides in the System file. To make it convenient to write a desk accessory or driver in Pascal or C, MPW provides the following:

- The library `DRVRRuntime.o`, which contains the glue for the driver routines *open*, *prime*, *status*, *control*, and *close*.
- The resource type 'DRVW', declared in `:RIncludes:MPWTypes.r`. The 'DRVW' resource is a special case of a 'DRVW' resource, and contains constants that point to the addresses of the driver routines in `DRVRRuntime.o`.

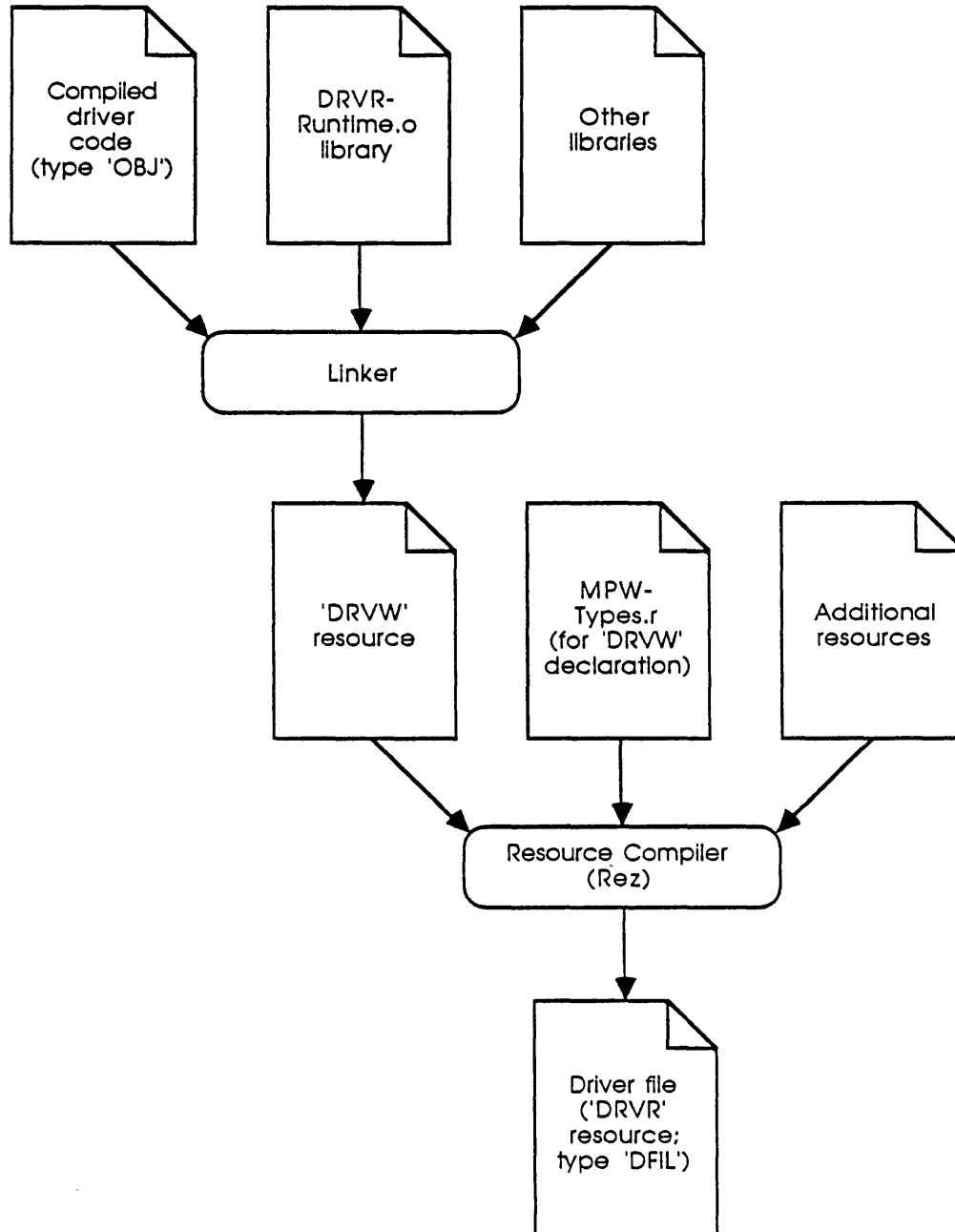
For information on using the 'DRVW' resource and `DRVRRuntime` routines to write a desk accessory, see Chapter 15. The remainder of this section describes how to put together a desk accessory and install it.

Putting together a desk accessory or driver requires two steps:

1. Link your driver code with the `DRVRRuntime` library and with any other libraries you need. The object code is linked into a code resource of type 'DRVW', an intermediate form of the standard 'DRVW' resource.
2. Use the Resource Compiler, `Rez`, to create the final driver file. That is, compile the linked 'DRVW' resource into a standard 'DRVW' resource, using the 'DRVW' type declared in `:RIncludes:MPWTypes.r`, together with any other resources your desk accessory may require.

You then install your desk accessory in the System file by using the Font/DA Mover.

Figure 9-3 illustrates the process of building a desk accessory or other driver.



**Figure 9-3**  
Building a desk accessory with DRVRRuntime

❖ *Note:* Of course, it's still possible to create a desk accessory directly in assembly language, without using DRVRRuntime.



---

## Linking a desk accessory or driver

These things must be done to link a driver or desk accessory:

- The Linker's **-rt** option must be specified. The **-rt** option indicates the link of a desk accessory or driver and sets the resource type and ID. (The default, if no **-rt** option is specified, is to output 'CODE' resources beginning with resource ID 0.)
- When you link a desk accessory or driver, the code must be in a single segment (that is, no jump table is constructed). You can map code from several segments into a single segment with the **-sg** or **-sn** options.
- Desk accessories written in Pascal or C must be linked with `DRVRRuntime.o`, which should appear first in the list of object files.

For example, the following command links the sample desk accessory file `Memory.c.o`, placing the output in the file `Memory`. (This output is the intermediate 'DRVW' resource, which must be converted into a 'DRV' resource as explained in the next section.)

```
Link -w 0
 -rt DRVW=0 0
 -sn Main=Memory 0
 "{Libraries}"DRVRRuntime.o # must appear first 0
 Memory.c.o 0
 "{CLibraries}"CRuntime.o 0
 "{CLibraries}"CInterface.o 0
 -o Memory.DRVW
```

This command has these results:

- The **-rt** option sets the output resource type to 'DRVW' and the resource ID to 0.  
*Note:* This ID must match the ID specified in the `$$resource` statement in the Rez input file. Note also that any additional resources "owned" by the desk accessory must observe a special numbering convention, as described in the Resource Manager chapter of *Inside Macintosh*.
- The **-sn** option combines the segment `Main` into the segment `Memory`.
- The specified files are linked. The `DRVRRuntime.o` library must be the first object file in the link list. This ordering ensures that the main entry point in `CRuntime.o` will be overridden by the `DRVRRuntime.o` entry point. (A Linker warning will call attention to this requirement.) The main entry point in `CRuntime.o` cannot be used for desk accessories.

Desk accessories must not call routines that use global variables, and therefore are less likely to need routines from the Pascal, C, and specialized libraries listed in Table 9-1. In a correct link, the Linker's progress information will report "Size of global data area: 0," and "No data initialization." If global data is somehow allocated, the link will succeed, but the desk accessory will not run correctly.

---

## The desk accessory resource file

The last step in the construction of a desk accessory or driver is to put together the DRVVR header with the linked code. The following example of a Resource Compiler (Rez) input file shows how this is done:

```
#include "Types.r"
#include "MPWTypes.r"

type 'DRVVR' as 'DRVW';

#define DriverID 12 /* The number is irrelevant */

resource 'DRVVR' (DriverID, "\0x00Memory", purgeable) {
 dontNeedLock, /* OK to float around, not saving ProcPtrs */
 needTime, /* Yes, give us periodic Control calls */
 dontNeedGoodbye, /* No special requirements */
 noStatusEnable,
 ctlEnable, /* Desk accessories only do Control calls */
 noWriteEnable,
 noReadEnable,
 5*60, /* Wake up every 5 seconds */
 updateMask, /* This DA only handles update events */
 0, /* This DA has no menu */
 "Memory", /* This isn't used by the DA */
 $$resource("Memory.DRVW", 'DRVW', 0)
};
```

The header information contains the details of the desk accessory's event mask, menu ID, and so on. (See the Device Manager chapter of *Inside Macintosh* and the file MPWTypes.r for information about the format of a 'DRVVR' resource.) The \$\$resource directive then appends the linked object code to the DRVVR header where it belongs.

If your desk accessory has any owned resources, such as 'STR#' or 'WIND' resources, you can add them to your desk accessory's Resource Compiler input.

To build the desk accessory resource, use the Rez command to compile the resources you have specified, and set the file type and creator for a Font/DA Mover document:

```
Rez -c DMOV -t DFIL Memory.r -o Memory
```

The file type DFIL indicates a document file for the Font/DA Mover; the creator DMOV indicates a Font/DA Mover document (suitcase icon).

To install a desk accessory, use the Font/DA Mover to place the desk accessory in the System file. You can do this from MPW as follows:

```
"Font/DA Mover" Memory
```

After exiting the Font/DA Mover, you can execute the desk accessory by selecting its name from the Apple menu.

---

---

## Modifying the Build menu and makefiles

The Directory and Build menus are implemented as AddMenu commands and scripts. This lets you customize these menus to serve your own specific needs. In addition, the makefiles created by using the Create Build Commands menu option (script CreateMake) can be modified as your scripts grow in complexity. See Chapter 3 for a description of the Directory and Build menus.

---

### Variables

The Shell variable *{Program}* is used to remember the name of the most recently built program. It is set by the Create Build Commands... menu item and by each of the Build... menu items. *{Program}* is used as the default program name for the Build menu items.

---

### Scripts

The following scripts implement the Directory and Build menus. A © following the name of the script indicates that the script supports a Commando interface. Size estimates are also provided. These scripts are located in the Scripts: folder. Each is documented in detail in Part II.

|                |    |                                                |
|----------------|----|------------------------------------------------|
| DirectoryMenu  | 2K | Create the Directory menu                      |
| SetDirectory © | 2K | Set the current directory                      |
| BuildMenu      | 2K | Create the Build menu                          |
| CreateMake ©   | 4K | Create a program makefile                      |
| BuildProgram   | 2K | Build the selected program show build commands |
| BuildCommands  |    |                                                |

---

### Files

Items in the Directory and Build menus may create the following files:

|                   |                                                |
|-------------------|------------------------------------------------|
| <Program>.make    | Makefile containing build commands for program |
| <Program>.makeout | Build instructions for current build           |
| {MPW}MPW.Errors   | Diagnostic output from commands run from menus |

---

### UserStartup

The Directory and Build menus are installed by scripts from the UserStartup file. The commands listed below should be in UserStartup. In addition to creating the Directory and Build menus, they create the aliases needed to support the Directory menu.

```
DirectoryMenu `(Files -d "{MPW}"=Examples) ≥ Dev:Null` `Directory`
BuildMenu
```

The parameters to DirectoryMenu become the initial list of directories in the Directory menu. You can replace or augment the Examples directories with your favorite list of directories.

---

## Modifying the makefiles

As the complexity of your program increases, you can modify the makefile created by the Create Build Commands menu item. You might add new dependencies, specify Compiler and Linker options, and so on. The Build menu item will continue to build the program, using the modified instructions.

### Include dependencies

You may want to modify a makefile created by the Create Build Commands menu item, in order to overcome the limitations of the CreateMake script. Makefiles created by CreateMake do not include dependencies on `include` files or Pascal USES files. If you plan to change the Include or Uses files, consider modifying the makefile to express these dependencies.

For example, assuming that the C source file `Count.c` includes the header file `Utilities.h`, add the following dependency rule to the file `Count.make`:

```
Count.c.o f Utilities.h
```

No build rules are required; just add the dependency rule. Several `include` or USES files can be listed in the same dependency rule, or separate rules can be used for each dependency. Don't forget to specify the directory for files located in another directory.

### Library object files

Makefiles created by CreateMake link your program with the selected libraries listed on the CreateMake command page in Part II. The libraries are selected according to the language(s) in which your program is written and according to the program type (application, tool, desk accessory).

If your program calls routines in a library that is not automatically included in the build commands, then modify the makefile to add that library. The library's name should be added in two places: in the dependency rule immediately before the Link command (if you expect to modify the library), and in the list of libraries in the Link command itself.

If you consistently need to add the same library to your makefiles, you can modify the CreateMake script to include it automatically in the dependency and build rules.

---

---

## Building an MPW tool

In addition to traditional Macintosh applications, the Shell provides an environment for a type of program called an **MPW tool**. Tools are similar to desk accessories in many aspects of their behavior. When a tool is run from the Shell, it does not replace the Shell nor erase the screen, but instead runs within the Shell environment and has access to the facilities provided by the Shell. The Assembler, the Compilers, Link, Make, and so on, are all tools in the MPW system.

Chapter 13 is a full exposition on writing MPW tools. For a description of the facilities available to an MPW tool, see Appendix F.

---

## Linking a tool

Linking an MPW tool is the same as linking an application, except that the file type must be set to MPST and the creator to 'MPS ' (*MPSspace*):

```
Link -t MPST -c "MPS " ...
```

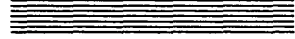
Sample tools are provided in the Examples folders for each of the MPW languages—refer to the sample makefiles for examples of the commands used to build a tool. Note that the sample tools are linked with the files Stubs.a.o or Stubs.c.o. These files contain dummy library routines used to override standard library routines that aren't used by MPW tools, thus reducing the tools' code size.

- ❖ *Note:* As a matter of convenience, tools are usually kept in the {MPW}Tools folder. This allows you to invoke the tool by using its simple name instead of its full pathname. {MPW}Tools is one of the directories that the Shell automatically searches when a command name is given with a partial pathname. (The Shell variable {Commands} contains a comma-separated list of directories to be searched; you can easily modify it to include additional directories.)





## Chapter 10



# Advanced Programming Tools

---

---

## Using Make

The Make tool enables you to keep track of all of the components of a program and their relationships to each other—then, when one component of a program is modified or updated, Make lets you automatically update all other parts of the program that depend upon it. These updates may be such things as compiles, assemblies, links, and resource compiles.

Make reads a **makefile** that describes the dependencies of the various components of a program, and outputs commands on the basis of those dependencies. This section describes how to write a makefile and use Make. (The Make command and command-line options are described in Part II.)

---

### Format of a makefile

A makefile is a text file that describes dependency information for one or more target files. A **target file** is a file to be rebuilt; it depends on one or more **prerequisite files** that must exist or be up-to-date before the target can be rebuilt. For example, an application depends on its source file or files, a number of library files, and resource files. If any of a target's prerequisite files are newer than the target, then the target needs to be rebuilt.

The dependency relationship of target prerequisites can be recursive, that is, a target's prerequisites may themselves be targets with their own prerequisites, and so on. A top-level target (one that is not a prerequisite of any other target) is called a **root**. A makefile may have one or more roots.

A makefile can include dependency rules, variable definitions, and comments. Table 10-1 summarizes the syntax of a makefile, and the sections following the table go into more detail.

**Table 10-1**  
Makefile summary

---

|                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>targetFile... f [ prerequisiteFile... ]<br/>[ ShellCommands... ]</i> | Dependency rule, with or without build commands. ( <i>f</i> is Option-F on the keyboard.) This rule means that <i>targetFile</i> depends upon <i>prerequisiteFile</i> . If any of the prerequisites are newer than the target, the subsequent Shell commands are output so that the target can be made up-to-date with respect to its prerequisites.<br><br><i>Important:</i> Build commands must begin with a space or tab. |
| <i>targetFile... ff [ prerequisiteFile... ]<br/>ShellCommands...</i>    | Dependency rule, requiring its own set of build commands                                                                                                                                                                                                                                                                                                                                                                     |
| <i>.[suffix] f .suffix<br/>ShellCommands...</i>                         | Default rule (specifies suffix dependencies)                                                                                                                                                                                                                                                                                                                                                                                 |
| <i>targetDirectory: ... f searchDirectory: ...</i>                      | Directory dependency rule (used with default rules)                                                                                                                                                                                                                                                                                                                                                                          |
| <i>variableName = stringValue</i>                                       | Variable definition                                                                                                                                                                                                                                                                                                                                                                                                          |
| <i># comment</i>                                                        | Comment                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <i>{name}</i>                                                           | Variable reference                                                                                                                                                                                                                                                                                                                                                                                                           |
| <i>"...", '...', @</i>                                                  | Quotes (as in the Shell)                                                                                                                                                                                                                                                                                                                                                                                                     |
| <i>@Return</i>                                                          | Line continuation character                                                                                                                                                                                                                                                                                                                                                                                                  |



❖ *Note:* Makefile physical input lines may not exceed 255 characters. Logical input lines (made up of one or more physical input lines continued with the continuation character) may be of arbitrary length.

A makefile for the sample Pascal application (Sample) is shown below:

```
Variable Definitions
Libs = "{Libraries}"Interface.o ∂
 "{Libraries}"Runtime.o ∂
 "{PLibraries}"Paslib.o

Dependency Rules
Sample ff Sample.r # Sample depends on Sample.r
Rez Sample.r -o Sample

Sample ff Sample.p.o ∂ # Sample depends on Sample.p.o ∂
 Sample.r # and Sample.r

Link Sample.p.o ∂
 (Libs) ∂
 -o Sample

Sample.p.o f Sample.p
Pascal Sample.p
```

Sample makefiles are contained in the Examples folders for each of the MPW languages (introduced in Chapter 2).

---

## Dependency rules

A **dependency rule** specifies the prerequisite files of a given target file, together with a list of the commands needed for building the target file. These commands will be written to standard output if any one of the prerequisite files is newer than the target file or if the target doesn't exist. The general form of a dependency rule is

```
targetFile... f [prerequisiteFile...]
[ShellCommands...]
```

The first line is called the **dependency line**. It consists of one or more target file names, followed by the *f* (Option-F) character (meaning "is a function of"), followed by a list of prerequisite files separated by blanks or tabs. Make looks at the modification dates of the prerequisite files (and their prerequisites, if any) and decides whether the target needs to be rebuilt.

Because a target's prerequisites may themselves be targets with their own prerequisites, the investigation of prerequisites is recursive and "bottom up." Thus, commands to rebuild lower-level targets are issued, if necessary, before the dependency rule determines whether higher-level targets need to be rebuilt.

All subsequent lines *that begin with a space or tab* are **build command lines**. These are Shell commands that will be output if the target needs updating. (When Make writes these command lines to standard output, the initial space or tab is omitted.) If the dependency rule omits the build commands, then the rule expresses only the target's dependencies. The build lines for the target are assumed to be supplied by another rule.

For example,

```
Sample.p.o f Sample.p
 Pascal Sample.p
```

The first line in the example is a dependency rule for the Pascal object file `Sample.p.o`. This rule states that `Sample.p.o` depends on the source file `Sample.p`.

The second line is the associated build command line. If `Sample.p` is newer than `Sample.p.o`, or if `Sample.p.o` doesn't exist, the command `Pascal Sample.p` is written to standard output.

More than one target file name can appear on the left-hand side of an “*f* rule.” Each target file on the left-hand side depends on all of the files listed on the right-hand side (and has the same build commands, if specified). If more than one target file is specified, it's exactly as if a separate dependency rule had been given for each target. The built-in Make variable `{Targ}` has the value of the current target.

You can also state multiple dependency lines for a single target—multiple dependencies mean that the target depends on all of the prerequisite names that appear on all of the lines. If no build commands are specified for a dependency line, the build commands are taken from the target's other dependency rules, or default rules, if present.

❖ *Note:* With the standard “single-*f*” form of the dependency rule, only one sequence of build commands may be specified for any given target. Thus, one dependency rule specifies a target's build commands, and additional rules simply specify additional dependencies.

### Double-*f* dependency rules

Double-*f* dependency rules are slightly different from the standard single-*f* rules. Syntactically, a double-*f* dependency entry is the same as a single-*f* entry, except that ***ff*** is used in place of ***f***. The difference in use is that more than one double-*f* rule is expressed for an individual target and that each double-*f* rule requires its own set of build commands. For example,

```
TargetFile ff A B D
 build commands-1
TargetFile ff C D
 build commands-2
```

If the target is out-of-date with respect to one or more dependency sets, each of the corresponding sets of build commands will be output (in the order they appear in the makefile). That is, if `TargetFile` is out-of-date with respect to both `A` and `C`, then both sets of build commands are output. (In single-*f* rules, only one set of build commands can be specified for any one target.)

If `TargetFile` is out-of-date with respect only to `B`, then only the first set of build commands is output. If `TargetFile` is out-of-date with respect to `D`, then both sets of build commands are output, because `D` appears in the dependency sets for both. In other words, the same file can appear as a prerequisite in more than one double-*f* dependency set.

Double-*f* rules are useful for separately building code and resources, as shown in the makefile for `Sample`. (For more examples, see the sample makefile at the end of this section.)

The build commands may be left off a double-*f* rule if they are to be supplied by default rules.

## Default rules

Default rules express dependencies between pairs of files whose names are the same but whose suffixes differ. They have the following form:

```
.[suffix1] f .suffix2
 ShellCommands...
```

(Note that the period must be present for a default rule to be recognized. The period is taken as part of the suffix.)

The power of default rules is that many specific dependencies and build commands can be expressed by a single rule, thereby eliminating the need to specify many similar dependency rules. Make has built-in default rules for assemblies and for C and Pascal compiles. You need to specify only the dependencies not covered by default rules.

Default rules are applied only when no build commands have been given for a particular target. You can override the built-in default rules by placing your own versions of the default rules in the makefile. You can augment the default rules for a particular file by additional dependency rules, as long as these dependency rules do not include build commands.

Default rules of the form

```
. f .suffix
```

specify dependencies between files with any name and files with the same name followed by the given suffix.

❖ *Note:* Default rules of this form slow down Make processing, because the empty left-hand side of the rule causes it to match against all filenames.

**Built-in default rules:** A compiled or assembled object file is dependent on its source file—this dependency is typically handled by the built-in default rules.

Additional object file dependencies may result from other units that you use or refer to in your source file—these may be `include` files, C header files, or Pascal USES files. These additional dependencies can be expressed by dependency rules with no build line component, leaving the build lines and object-to-source dependency implied by the default rules.

The data fork of the Make tool contains the following built-in default rules:

```
.a.o f .a
 {Asm} {AOptions} {DepDir}{Default}.a -o {TargDir}{Default}.a.o
.c.o f .c
 {C} {COptions} {DepDir}{Default}.c -o {TargDir}{Default}.c.o
.p.o f .p
 {Pascal} {POptions} {DepDir}{Default}.p -o {TargDir}{Default}.p.o
```

{Asm}, {Pascal}, and {C} are built-in Make variables. Their initial values are

```
{Asm} Asm
{Pascal} Pascal
{C} C
```

{AOptions}, {POptions}, and {COptions} are initially null; you can customize the built-in default-rule build commands by defining these variables in your makefile. For instance, you might want to specify the location of your Pascal include files by adding a *-i pathname* option to the default rules by a variable definition of the form

```
POption = -i pathname
```

Or you may want to indicate the use of a different C compiler by changing the value of the {C} variable.

You can redefine the {Asm}, {Pascal}, {C}, {AOptions}, {POptions}, and {COptions} variables. Variable definitions can be overridden in your makefile, on the command line (with Make's **-d** option), or by an exported Shell variable. See "Variables in Makefiles" below for information.

{Default} is another built-in variable; its value is the common part of the filenames matched by a default rule (defined dynamically when Make applies the default rule). The {Default} variable is what allows you to write a generic default rule without referring to a specific filename. Because its value is set dynamically by Make, its value cannot be overridden in your makefile.

{DepDir} and {TargDir} are built-in Make variables that allow default rules to work with the target and prerequisite files in different (or the same) directories:

{DepDir}    The directory component of the prerequisite name

{TargDir}    The directory component of the target name

❖ *Note:* {DepDir} and {TargDir} have values only when used in the build commands of default rules for which directory dependency rules were applied. In all other cases these variables evaluate to the null string so that they won't interfere with the normal behavior of default-rules. Directory dependency rules are explained in the section that follows.

**Directory dependency rules:** Normally, default rules work only within a single directory, that is, the target and prerequisite files will have the same directory component because the default rules change only the suffixes of the filenames. Directory dependency rules allow default rules to be applied across directories. Just as default rules imply changing a filename suffix between a target filename and a prerequisite filename, directory dependency rules imply changing the directory prefix of the filenames. Directory dependency rules have the form

```
targetDirectory: ... f searchDirectory: ...
```

Directory dependency rules are identified by dependency names that end in colons (that is, directory names). For example,

```
ObjFiles: f SrcFiles:
```

The above rule, together with the standard default rules, would mean, for example, that ObjFiles:*name.c.o* depends upon SrcFiles:*name.c*.

No build commands may be given for a directory dependency rule. More than one directory name may appear on either side of the rule. The current directory can be specified by a single colon (:) on either side of a directory dependency rule.

Directory dependency rules are applied only during the processing of default rules. If Make is applying a default rule and encounters a target name with a directory component, Make checks for a directory dependency rule for that directory. If one exists, Make tries prerequisite filenames with the directory prefixes given on the right-hand side of the rule. The names are tried in the order that they appear in the rule; thus more than one directory name on the right-hand side of a directory dependency rule constitutes a list of directories to search.

❖ *Note:* If default rules are meant to be applied from a directory A: to a directory B: and also within A: (that is, from A: to A:), then A: should appear on both the left and right sides of the directory dependency rule. For example,

```
A: f A: B:
```

---

## Variables in makefiles

You can use exported Shell variables and built-in Make variables within makefiles. You can also define variables within a makefile or on the Make command line.

### Shell variables

Make automatically defines exported Shell variables before it reads the makefile, so you can use Shell variables in dependency lines and build commands.

If Make doesn't recognize a variable reference in a build command line, the build line is left unchanged when it is output, so that it can be processed later by the Shell. (Unidentified variables in dependency lines are reported as errors.)

---

### Caution

Exported Shell variables override Make variables with the same names. An attempt to redefine a Shell variable in the makefile results in a warning message.

---

### Defining variables within a makefile

Variable definitions are makefile entries of the form

```
variableName = stringValue
```

Subsequent appearances of *{variableName}* will be replaced by *stringValue*. You can use line continuations to make a *stringValue* arbitrarily long. When a *stringValue* is continued across lines, any comments, blanks, or tabs at the end of the continuation line, and at the beginning of the line after the continuation, are replaced by a single blank. Thus variable values can conveniently contain lists of files. Note that variable values may contain references to other variables.

One common use of variables is to provide parameters to the directory portion of filenames so that you can easily adapt a makefile to different directory setups.

❖ *Note:* Make variables in build command lines are not expanded until Make generates the build commands—because command generation follows all dependency rule and variable definition processing, there's no requirement that variable definitions appear before their references in build command lines.

You can define a variable on the command line with Make's `-d` option; this option overrides any definition of the variable within the makefile, thus allowing the definition in the makefile to function as a default.

## Built-in Make variables

The following built-in Make variables have values that are dynamically assigned (and that cannot be overridden) as Make generates the build commands:

- {Targ}            The complete filename of the target on the left-hand side of the dependency rule whose build commands are being processed
- {NewerDeps}      A list of the names of all of the target's direct prerequisites that were newer than the target; that is, the files that caused the target to be out of date

These built-in variables can be used only in build command lines, because they have no value when dependency lines are processed. They cannot be overridden.

When default rules are applied, the following variables are also defined:

- {Default}        The common part of the filenames matched by a default rule
- {TargDir}        The directory component of the target name
- {DepDir}        The directory component of the prerequisite name

❖ *Note:* When expanding the built-in variables {Targ}, {NewerDeps}, {TargDir}, {DepDir}, and {Default} in build commands, Make automatically quotes their values, if necessary, because they will represent filenames or parts of filenames. Don't quote them yourself.

---

## Quoting in makefiles

The Make command supports several of the Shell's quoting conventions. Quoted items can appear in dependency lines, variable definition lines, and build command lines. The following quoting characters are used:

- ∂            Quotes the subsequent character; that is, the ∂ is removed and the subsequent character is taken to be a literal character (except when ∂Return is used at the end of a line as a continuation character).
- '...'
- "..."      Quotes the enclosed string, but {...} variable references are expanded, and the escape character ∂ is processed. The double quotes are removed.

Quotes are processed as follows:

- In dependency lines and in the name part of variable definitions, quotes literalize the quoted characters (useful for file or variable names).
- On the right-hand side of variable definitions, quoted items are passed through "as is," so that the quoting will take effect when the variable is expanded.
- In build command lines, quoted items are passed through as is, so that the quoting will take effect when the build commands are executed by the Shell.

## Line continuation character

Like Shell commands, dependency and variable definition lines can be continued over more than one line with ∂Return. ∂Return causes the ∂, any blanks preceding the ∂, the return, and any leading blanks on the next line to be replaced with a single space. Comments at the ends of such continued lines do not comment out the continuation character.

---

## Comments in makefiles

The number sign (#) indicates a comment. Everything from the # to the end of the line is ignored. Comments always end at the next return, even if the return is preceded by a \.

Comments may appear in dependency lines, variable definitions, and build command lines, or on lines by themselves. Comments in build command lines are passed through to standard output where they are processed as comments by the Shell.

---

## Executing Make's output

Make generates a set of commands, which must be executed separately to perform the actual updates. You can automatically execute Make's command output by calling Make from a Shell script. The simplest form of such a script consists of the two commands

```
Make {"Parameters"} > MakeOut.
MakeOut
```

The first command executes Make, using the parameters passed to the script. (See the description of the {"Parameters"} variable in Chapter 5 under "Variables.") Output (that is, build commands) is redirected to the file MakeOut. The second line of the script executes MakeOut.

---

## The order in which Make builds targets

Make builds the top-level target and its prerequisite subtargets in a recursive "bottom up" fashion. The top-level target (or targets) may be specified on the Make command line. If no target is specified on the command line, the default top-level target is the first target appearing on the left side of a dependency rule in the makefile.

The prerequisites of the top-level target (and subtargets) are investigated in a recursive "bottom up" order starting with the first prerequisite mentioned in the target's prerequisite list. After the first prerequisite (and its own prerequisites) have been investigated, the target's next prerequisite is investigated. The next prerequisite will be the next one mentioned in the current dependency rule or in the next dependency rule in the file that has the same left-side target.

Thus the important orderings within a makefile are: the first target mentioned (the default top-level target) and the order of prerequisites for any given target. Otherwise, the order in which targets are mentioned is not important.

Please note, however, that once a target has been investigated by Make it is not revisited even if it appears somewhere else in the top-level target's prerequisite dependency hierarchy. In other words, while a file may appear as a prerequisite of a number of program components, Make will rebuild it only once (if necessary) when it is first encountered in the recursive "bottom up" traversal of the dependency hierarchy.

Remember that a makefile may have one or more top-level targets (or roots), that is, it may describe how to build more than one object. (The -r option will identify all the roots.) Running Make will rebuild only the targets you specify on the command line, or the default target if none are specified.

---

## Debugging makefiles

When Make doesn't seem to be doing what you expect, the next step is to debug your makefile. The following procedures are helpful in debugging makefiles:

1. Use Make's `-v` option to generate verbose diagnostic output. This output tells you which files don't exist, which files are up-to-date, which files need rebuilding, and why they are out-of-date. It also points out which files don't have build rules and, thus, are assumed to be artificial targets (targets that are abstract and not really built—see, for example, Note 8 in the Make example that follows this section).
2. Use Make's `-s` option to show the structure of your target's dependency relations. This option displays the complete structure of dependencies, including those generated by default rules. A target (or subtarget) that doesn't appear or that has no prerequisites may indicate a typographical error in the dependency line for that target (or in the line for one of the targets that depend on it). A target that appears at the wrong level in the dependency graph indicates an error in your dependency specification.
3. Use the `-u` option to find unreachable targets. These may be parts of your target dependencies that did not get connected to the rest of the dependency hierarchy because of a bad or mistyped rule.

### Problems due to command generation before execution

Make generates commands that must be separately executed to perform the actual updates. Because Make must determine what build commands to generate before any targets are actually built, the possibility of "phase errors" is introduced; that is, unexpected behavior may occur when generated commands alter the assumptions that Make used to determine whether targets were out of date. (You won't experience these problems unless you have build commands that do things such as deleting files that Make thinks are already up to date.)

### Problems with different specifications for the same file

You'll experience problems with Make if you use different pathname specifications for the same file (that is, pathnames with different degrees of volume and directory qualification). Make uses the name strings exactly as encountered in dependency lines, so different name strings will result in different entries. (This is done for the sake of performance—no calls are made to the file system, except to inquire about the date of targets that are supposed to be built.) If there is more than one name specification for the same file, each name results in a different Make target, and the resulting dependency relations will be wrong.

### Problems with default rules

An error message may appear saying that no rules were available to build something that should have been covered by a default rule. This situation may result from any one of the following problems:

- The default rule may not have matched against anything, and was thus not applied; for example, the default rule

```
.p.o f .p
```

cannot be applied if the `.p` file does not exist either in the file system or in the makefile dependency specification.



- There may be a typographical error in the filename, so that the default rule could not be applied. You should be able to detect such errors by inspecting the output of Make's `-s` and `-v` options.
- There may be a typographical error in a default rule that was given in the makefile, in which case you may not see any dependencies generated by the rule when you use the `-s` option on the Make command line.

---

## An example

This section lists the makefile used to build an experimental version of the Make tool itself (represented in this makefile by the `MakeX` target). A series of explanatory notes follows the listing. These notes describe in detail a number of the Make features that were used.

```
Variables

ToolDir = {Boot}ToolUnits: See note ①
MakeUses = {ToolDir}MacInterfaces.p.o ∂ See note ②
 {ToolDir}MemMgr.p.o ∂
 {ToolDir}SymMgr.p.o ∂
 {ToolDir}Utilities.p.o ∂
 {ToolDir}IOInterfaces.p.o ∂
 {ToolDir}CursorCtl.p.o ∂
 {ToolDir}ErrMgr.p.o ∂
 {PInterfaces}IntEnv.p ∂
 {PInterfaces}MemTypes.p ∂
 {PInterfaces}QuickDraw.p ∂
 {PInterfaces}OSIntf.p

MakeObjs = Make.p.o ∂
 {ToolDir}Stubs.a.o ∂
 {ToolDir}CallProc.a.o ∂
 {ToolDir}Utilities.p.o ∂
 {ToolDir}Utilities.a.o ∂
 {ToolDir}IOInterfaces.p.o ∂
 {ToolDir}IOInterfaces.a.o ∂
 {ToolDir}MemMgr.p.o ∂
 {ToolDir}MemMgr.a.o ∂
 {ToolDir}SymMgr.p.o ∂
 {ToolDir}SymMgr.a.o ∂
 {ToolDir}CursorCtl.p.o ∂
 {ToolDir}CursorCtl.a.o ∂
 {ToolDir}ErrMgr.p.o ∂
 {ToolDir}MacInt.a.o ∂
 {ToolDir}MacInterfaces.p.o

Libs = {Libraries}Runtime.o ∂
 {PLibraries}PasLib.o ∂
 {Libraries}Interface.o

LinkOpts = -w # no warnings (duplicates due to Stubs.a.o)
 See note ③

SourceFiles = Make.p ∂
 DefaultRules ∂
 Makefile
```

```

Default Rule Customizations
POptions = -i {Boot}ToolUnits: See note ④
Dependency Rules
MakeX ff {MakeObjs} {Libs} See note ⑤
 Link {LinkOpts} -p -b -o MakeX ∂
 -t MPST -c "MPS " ∂
 {MakeObjs} {Libs} ≥LinkMsgs

MakeX ff defaultRules
 Duplicate -d defaultRules MakeX -y # copy default rules into Make's data fork

MakeX ff {MakeObjs} {Libs} defaultRules
 SetFile MakeX -m . -d . #set last-mod and creator dates

Make.p.o ff {MakeUses} See note ⑥
 Delete MakeLoad -i #delete Make's Load/Dump file if out-of-date

Make.p.o ff Make.p
 Save Make.p ≥Dev:Null || Set Status 0 #save source before compile if changed

Make.p.o ff {MakeUses} #will be augmented by default rules

{ToolDir}MacInterfaces.p.o f
 {PInterfaces}MemTypes.p ∂ See note ⑦
 {PInterfaces}QuickDraw.p ∂
 {PInterfaces}OSIntf.p ∂
 {PInterfaces}ToolIntf.p ∂
 {PInterfaces}PasLibIntf.p ∂

{ToolDir}MemMgr.p.o f
 {ToolDir}Utilities.p.o ∂
 {ToolDir}MacInterfaces.p.o ∂
 {PInterfaces}MemTypes.p ∂

{ToolDir}SymMgr.p.o f
 {ToolDir}MemMgr.p.o ∂
 {PInterfaces}MemTypes.p ∂

{ToolDir}Utilities.p.o f
 {PInterfaces}MemTypes.p ∂

{ToolDir}IOInterfaces.p.o f
 {ToolDir}Utilities.p.o ∂
 {ToolDir}MacInterfaces.p.o ∂
 {PInterfaces}MemTypes.p ∂

Backup f See note ⑧
 Duplicate -y = MakeSrc: #backup to Sony

Restore f
 Duplicate -y MakeSrc:= : #restore from Sony

Listings f {SourceFiles} See note ⑨
 Print -h -r -ls .85 -s 8 -b -hf helvetica -hs 12 {NewerDeps}
 Echo "Last listings made `Date`" >Listings

```

## Notes on Make's makefile

- ① The exported Shell variable {Boot}, used in the definition of {ToolDir}, is automatically defined by Make when invoked.
- ② Several variables—{MakeUses}, {MakeObjs}, {Libs}, and {SourceFiles}—are used for lists of filenames. This is a convenience because the lists are used in several places later in the makefile; it also helps to reduce errors. Note that you can temporarily remove any file from the list by placing a comment character at the beginning of the line for the file.
- ③ The {LinkOpts} variable is used to specify Linker options (and is used only once). This usage is handy because the definition in the makefile functions as a default that can be overridden from the command line with the **-d** option, as in

```
Make -d LinkOpts='-w -l >Map'
```

- ④ The {POptions} definition gives a value to one of the variables used in the default rules, customizing the built-in default rules for Pascal compiles for this particular makefile.
- ⑤ The three sets of *ff* rules for MakeX (an experimental version of the Make tool) handle (a) the Make link (which creates MakeX's code resources), (b) the copying of the default rules to MakeX's data fork (Make reads the built-in default rules from its own data fork), and (c) the setting of the creation and modification dates. The link will take place only if the MakeX objects or libraries change. The default rules will be copied only if the rules have changed. And the setting of the dates will take place if either of the first two rules was activated. (Note that the third rule has the union of the dependency relations of the first two.)
- ⑥ The three sets of *ff* rules for Make.p.o control the compilation of the main source for Make, with some interesting side effects. The Make source uses the Pascal Compiler's SLOAD option, which creates a symbol table for the Uses that can be loaded much faster than the Uses are normally processed. The first of the *ff* rules is used to delete this load file (MakeLoad) if it has been invalidated by a change in the Uses files. This rule is interesting in that it deletes rather than builds something. The second of the *ff* rules saves the Make source before it is compiled, only if the source file has changed. The last of the *ff* rules does the actual compile. Note that this last rule has no explicit build commands, so it will be augmented by the built-in default rules, which will add a dependency relation (on the source file Make.p), and will supply the actual build commands for the compile.
- ⑦ The dependency rules for MacInterfaces, MemMgr, SymMgr, Utilities, and IOInterfaces describe dependencies between various utility units used by Make. Several dependencies on library interface files are given. Dependencies among the utility units themselves are described by indicating a dependency on the object files of the lower-level (predecessor) units. These dependencies could have been expressed as dependencies on the source files of the lower-level units (because it is the source files that are read in a Uses list). However, expressing these dependencies on the object files has the nice property of ensuring that the lower-level units have been successfully compiled before the higher-level units are built.
- ⑧ The Backup, Restore, and Listings targets are additional **roots** (top-level targets) in Make's makefile, and thus represent other things that can be built besides MakeX itself. Note that the Backup and Restore targets do not actually get built by their build rules; thus they are *artificial targets* and will always generate build commands if they are specified on the Make command line. Note also that they do not have any dependency relations.

- ⑨ The build rules for the Listings target demonstrates the use of the {NewerDeps} variable. The prerequisite of Listings is a list of the Make source files. The first build command prints the {NewerDeps} files. {NewerDeps} contains the names of the prerequisites that are newer than the target, that is, the source files that have changed since listings were last made. The last line of the build rules simply writes the current date into a file called Listings, which is the name of our target—this action results in a file that remembers when listings were last made. (The fact that the date is written into the file is unnecessary but convenient; the Echo itself is enough to change the file's last-modified date.)
- ❖ *Note:* There are several implicit builds that will be generated as needed by the default rules. For example, the {MakeObjs} variable includes several assembly-language object files. Because {MakeObjs} appears as a prerequisite of the link step, these assemblies will be generated, if necessary, before the link.

---

---

## More about linking

This section supplements the information given under the description of the Link command in Part II and in Chapter 9 under “Linking.” This section may be of interest after you're familiar with the major MPW tools and are ready to optimize your programs or build procedures.

---

### Linker functions

After a source file has been assembled or compiled into an object file, it contains

- Object code (relocatable machine language).
- Symbolic (named) references to all identifiers whose locations were not known at compile time. (These include references to routines from separate compilations and libraries, and references to global variables.)

The Linker performs the following functions:

- Sorts code and data modules into segments, by segment name. (Within a segment, modules are placed in the order in which they occur in the input files.) The **-sg** and **-sn** options allow you to change segmentation at link time.  
*Note:* A **module** is a contiguous region of memory that contains code or static data. A module is the smallest unit of memory that is included or removed by the Linker. A **segment** is a named collection of modules.
- Omits unused (“dead”) code and data modules from the output file. (These modules can be listed with the Linker's **-uf** option, and deleted from libraries with the Lib command's **-df** option.)
- Provides (together with the Segment Loader) a jump table architecture that supports relocation of code and data at run time. (See the Segment Loader chapter of *Inside Macintosh* for more information about the jump table.)
- Constructs jump-table entries only when needed; that is, only when a symbol is referenced across segments. This means the jump table will be minimum size.
- Edits instructions to use the most efficient addressing mode. A5-relative (jump table) addressing is used across segments, and PC-relative addressing is used within a segment.

- Provides (with the data initialization interpreter) support for relocation of data references at run time. (The **data initialization interpreter** is the module `_DATAINIT` in the libraries `Runtime.o` and `CRuntime.o`.)
- Generates a cross-reference listing of link-time (object-level) names (`-x` option).
- Generates a location map for debugging or performance analysis (`-l` option).

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.

- ❖ *Note:* If Linker errors or a user interrupt cause the output file to be invalid, then the Linker sets the file's modification date to "zero" (Jan. 1, 1904, 12:00 a.m.). This action guarantees that the Make command will recognize that the file needs to be relinked, and that the MPW Shell will not run the file.

---

## Segmentation

Segmenting a program makes it possible for unused parts of the program to be unloaded and purged from memory, thus freeing up memory space. You specify the name of a segment by placing a directive in your program's source file—see the appropriate MPW language reference manual for information. Each segment is linked into a separate code resource.

- ❖ *Note:* For a desk accessory or driver, the code must be in a single segment, and no jump table is constructed. Segmentation applies only to applications and MPW tools.

The Linker sorts object code into load segments by name, allowing you to organize your source code for clarity and understanding. You can specify the same segment name more than once—the Linker collects code for a given segment name from all of the Linker input files and places it into a single segment in the output file.

---

### Caution

Segment names are case sensitive. For example, "Seg1" and "SEG1" are not equivalent names. If you aren't sure about the cases used, you can use the Linker's `-p` option to get a listing.

---

By default, resources created by the Linker are given resource names identical to the corresponding segment names. Link provides options for combining and renaming segments at link time (`-sg` and `-sn`). If you don't specify a segment name before the first routine in your file, the main segment name ("Main") is assumed there. Normally, you should give the main segment the name `Main`.

By default, segments are limited to 32760 bytes. This limit ensures compatibility with all versions of the Macintosh ROM. Larger segments are allowed with the Linker's `-ss` option.

- ❖ *Note:* Object code is placed in a segment in the order that it's encountered in the input file. For segments larger than 32K, the order is important because PC-relative offsets are limited to 32K by MC68000 instructions.

For more information about segmentation, see the Segment Loader chapter of *Inside Macintosh*.

## Segments with special treatment

When linking a main program, the Linker creates two segments that don't appear in the input object files:

- The jump table ('CODE' resource, ID=0), which is unnamed.
- The global data area (no resource), which is named %GlobalData and appears only in the link map file (described below). You can't change the name %GlobalData at link time.

There are also two segments that have special conventions:

- The segment that contains the main program entry point ('CODE' resource, ID=1), usually named Main.
- A segment named %A5Init, which contains the initial values for the global data area and code that moves these initial values to the global data area. Applications should unload this segment to avoid memory fragmentation. You can unload the %A5Init segment by calling UnloadSeg with the address of entry point \_DATAINIT as its parameter; for example,

```
UnloadSeg (&_DATAINIT) ;
```

In C and Pascal, this call should be the first statement in the application. In assembly language the call to UnloadSeg should follow the call to \_DataInit.

---

## Setting resource attributes

Resources have attributes that control when and how they are loaded. The default resource attribute values set by the Linker are shown in this table.

| Resource attributes |      |         |                                   |
|---------------------|------|---------|-----------------------------------|
| 'CODE' resource     | Hex  | Decimal |                                   |
| 0 (JumpTable)       | \$20 | 32      | resPurgeable                      |
| 1 ("Main")          | \$34 | 52      | resPurgeable+resLocked+resPreLoad |
| Others              | \$20 | 32      | resPurgeable                      |

- ❖ *Note:* For linking MPW tools (programs with output file type MPST and output file creator 'MPS '), all segments default to resPurgeable. Make sure that you *do not* set the resLocked bit for a tool.

The Linker option **-ra** sets the resource attributes. Some useful resource attribute values are

```
$20 32 resPurgeable
$10 16 resLocked
$08 8 resProtected
$04 4 resPreLoad
```

For more information about resource attributes, see the Resource Manager chapter of *Inside Macintosh*.

The Linker also sets the resChanged attribute (when a changed resource is in memory, and needs to be updated in the file). The Linker does not check or enforce settings for the other resource attribute bits.

---

## Controlling the numbering of code resources

Normally, you don't need to worry about which segments are given which resource numbers. However, you may want to control the assignment of resource numbers to optimize program load time, to implement a specialized code manager, or to match the numbering produced by another linker.

The Linker creates and numbers code resources based on the order in which it encounters the segment names in the command-line parameters and the input object files. If you can't easily predict the order in which the names appear in the object files, you may want to force the ordering with command-line options that contain dummy segment-mapping directives. For example, the following sequence of Linker options forces Main to come first, followed by Init, Body, and Term:

```
Link -sn dummy1=Main # must contain the main code module @
 # or entry point @
 -sn %A5Init=Init @
 -sn dummy3=Body @
 -sn dummy4=Term @
 etc.
```

The "old" segment names may be either "dummy" names (which don't appear in the object files) or actual mappings, such as the mapping of the %A5Init code into the segment Init.

---

## Resolving symbol definitions

This section describes how the Linker resolves references to symbols. For a more detailed discussion of local and external symbols, see Appendix F.

Symbols in object files are either local or external. A **local** (module, entry point, or segment) can be referenced only from within the file where it is defined. An **external** (module, entry point, or segment) can be referenced from different object files. An **entry point** is a location (offset) within a module. (The module itself is treated as an entry point with offset zero.) A **reference** is a location within one module that will contain the address of another module or entry.

### Multiple external symbol definitions

If the object files contain more than one definition for an external symbol, the first definition is used, and all references are treated as references to the first definition. This lets you selectively override entry points in libraries so that you can substitute new versions of code. When subsequent definitions are encountered, a warning is generated.

### Unresolved external symbols

Occasionally, you may find that an external symbol is unresolved because a reference was generated with case sensitivity set one way, whereas the definition was generated with different case rules. When this happens, you can avoid recompiling by using the Linker option **-ma** (module alias). Whenever the Linker encounters an unresolved symbol, it checks the list of module aliases in an attempt to resolve it.

---

## Linker location map

If you specify the Linker option **-l**, the Linker writes a **location map** to standard output. The map is produced in location ordering, that is, sorted by *segNum*, *segOffset*.

The format is divided into the following fields:

```
name segName segNum,segOffset [@JTOffset] [#] [E] [C] [fileNum,defOffset]
```

For example,

```
 seg Main 1
TEFROMSCRAP Main 1,422 2,12892
TETOSCRAP Main 1,444 2,12946
%_BEGIN Main 1,46C 3,3398
%_INIT Main 1,46E 3,3420
etc.
 size Main 7CA

 seg %GlobalData 0
#0001 %GlobalData 0,0 # 3,2332
__PASHEAP %GlobalData 0,C 3,2886
PASHEAP %GlobalData 0,30 E 3,2892
QUICKDRAW %GlobalData 0,FE E 4,4826
_SAGlbls %GlobalData 0,FE # 4,4834
etc.
 size %GlobalData 26C

 seg %A5Init 3
_DATAINIT %A5Init 3,0 4,6338
_DataInit %A5Init 3,0 @ 32 E 4,6558
#0001 %A5Init 3,C8 # 4,6574
_A5Init %A5Init 3,C8 E 4,6586
 size %A5Init C8
```

- JTOffset* is a hex number giving the distance from the memory location pointed to by register A5 to the jump-table entry of the symbol.
- The number sign (#) indicates a local symbol, that is, not necessarily a unique name.
- The symbol "E" indicates an entry point in the immediately preceding module.
- The symbol "C" indicates that this module contains initialized data.
- FileNum* and *defOffset* are hex numbers giving the file number and offset within the file where the symbol is defined. They are included only if the **-lf** option is also specified.

The map of static global variables is presented as a data segment named *%GlobalData*. The offsets in this segment are positive—the zero byte is furthest below A5, and the highest-offset byte is the byte immediately below A5. In order to translate these positive offsets into negative offsets from A5 (as shown by the debugger), you need to subtract the size of *%GlobalData* from the offset.

No map information is provided for the data initialization descriptors, which are appended to segment *%A5Init*.



---

## Optimizing your links

Because of the complexity of the Linker's functions, the Link step is often the longest single step during incremental rebuilding of your program. The following steps can substantially speed up the Linker's performance:

- *Use a RAM cache.* The Linker must open and close many object files (particularly with the **-bf** option). Experience has shown that large links run up to four times faster when you use a RAM cache of 64K or more on machines with at least 1 megabyte or more of RAM. (Use the Control Panel desk accessory to check your RAM cache settings—if you change the setting, you must restart the MPW Shell to have the new setting take effect.)
- *Use the Lib utility to combine input files.* You can use the Lib command to reduce the number of input files so that the **-bf** option is not needed. Using Lib can give a 10–15% improvement in link speed. See “Library Construction” later in this chapter.
- *Eliminate unneeded files.* You can eliminate unneeded input to the Linker by heeding the warning “File not needed for link,” and deleting the files that are so identified. This means customizing your link lists, rather than relying on a generic makefile for linking.
- *Eliminate unneeded references.* You can also eliminate unneeded input by using Lib to remove unreferenced modules. Experience has shown that producing a specialized library file can increase Linker speed by as much as 40%. See the next section, “Library Construction.”

---

---

## Library construction

The Lib tool enables library construction by allowing you to combine object code from different files and languages into a single object file. For example, you can combine assembly-language code with C or Pascal. The Lib tool was used for this purpose in constructing the libraries distributed with MPW.

The Lib tool and its options are described in Part II. This section explains some aspects of using Lib.

Lib reorganizes the input files, placing the combined library file in the data fork of the output library file. By default, the library output file is given type 'OBJ ' and creator 'MPS '. Lib's output is logically equivalent to the concatenation of the input files, except for its optional renaming, resegmentation, and deletion operations, and the possibility of overriding an external name (as in Link). Lib *doesn't* combine modules into larger modules, nor does it resolve cross-module references. This limitation guarantees that the output of a link that uses the output of Lib is the same as that of a link using the “raw” files produced by the Compilers and Assembler.

Object files that have been processed with Lib result in significantly faster links than the “raw” object files produced by the Compilers and Assembler. There are several reasons for the speed improvements:

- Code and Data modules are separated into different sections, and Code modules are further sorted by segment name. These actions improve the performance of Link, which must sort input modules into output code resources.
- All of the named symbols in the object file are gathered into a single Dictionary area at the start of the file. This makes the output file smaller and simplifies the processing needed by Link to resolve references.

- When several object files are combined, multiple instances of a symbol definition are replaced by a single definition. Again, this makes the output file smaller and simplifies the processing by Link.

Lib correctly handles file-relative scoping conventions, such as nested procedures in Pascal, `static` functions in C, or `ENTRY` names in Assembly; that is, it never confuses references to an external symbol with references to a local symbol of the same name, even if the two symbols are in two files combined with Lib.

---

## Using Lib to build a specialized library

Each of the language libraries has files that you may or may not need to link with, depending on the functions your program calls. (See Appendix A.) Once you determine which files are needed for linking a particular program, you can greatly improve the performance of subsequent links by combining libraries into a single specialized library file. In building a specialized library, you can use Lib to

- change segmentation (with the `-sg` and `-sn` options)
- change the scope of a symbol from external to local (with the `-dn` option)
- delete unneeded modules (with the `-dm` option)

Lib's renaming, resegmentation, and deletion operations give you detailed control over external names, the contents of library files, and the segmentation of object code. To use these features, you may need to review some of the material in Appendix F in order to understand how modules and entry points are represented in object files. The `DumpObj` command is also useful in exploring the contents and structure of the library files provided with MPW.

## Removing unreferenced modules

You can eliminate unneeded input to Link by using Lib to remove unreferenced modules. You can determine the number of unreferenced modules from the Linker's progress information. (Use the `-p` option.) The Linker reports the total number of symbols read, as well as the number of active symbols (that is, the symbols in the output), and the number of visible symbols (that is, the symbols requiring jump-table entries). For example,

```
155 active and 54 visible entries of 714 read.
```

The difference between the total read and the number of active symbols is the number of unreferenced (and unneeded) symbols. Most of these unreferenced symbols represent standard library functions that your particular program doesn't require.

Unreferenced modules can be removed in three steps:


1. Use the Linker's `-uf` option to produce a file containing the unreferenced names.
2. Use the `-uf` file produced by Link as the input to Lib, using the Lib option `-df` to produce a specialized library that contains only the modules that your program requires.
3. Use the output of Lib as the input to subsequent links.

## **Guidelines for choosing files for a specialized library**

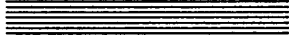
The choice of files to include in a specialized library file is largely dictated by “stability” issues: Files that are unlikely to change for many builds are the best candidates. “Stable” files include the library files provided by Apple for the ROM interfaces and for language support. Files that are under development are best left as single files.

Should you find it necessary to change one of the component files of a specialized library, you don’t always need to immediately rebuild the specialized library. You can simply include the newer version of the object file in the link list, before the specialized library file that contains the older version. You’ll get some warning messages about duplicate symbols, but all references will be correctly moved to the first definition encountered by the Linker. Later, after the file is stable again, you can rebuild the library.





# Chapter 11



## Debugging With MacsBug

This chapter describes the theory and operation of MacsBug, the Macintosh MC68xxx debugger. It also describes the syntax of commands accepted by MacsBug.

---

---

## About MacsBug

MacsBug is a line-oriented, single-Macintosh debugger. It resides in RAM along with the program being debugged. The capabilities of MacsBug include

- displaying and setting memory and registers
- disassembling memory
- stepping and tracing through both RAM and ROM
- monitoring system traps
- displaying and checking the system and application heaps

MacsBug obtains control when certain MC68000 exceptions occur. You can then examine memory, trace through the program, or set up break conditions and execute the program until those conditions occur.

MacsBug works with the following hardware configurations:

- 512 kilobytes to 4 megabytes of RAM
  - 64K or 128K ROMs (Macintosh Plus) or 256K ROMs (SE and II)
  - Motorola 68000 or 68020 processors
  - Motorola 68881 floating-point coprocessor
- ❖ *Note:* MacsBug does not work on the Macintosh XL. Macintosh XL users should use MacsBug.XL, which is also provided.

---

---

## Installing MacsBug

MacsBug is not a normal Macintosh application. Instead, MacsBug installs itself once at boot time and remains active until shutdown. Installation occurs if the following conditions are met:

1. MacsBug exists and is named "MacsBug".
2. It is on a startup (bootable) disk.
3. It is in the current System Folder. The System Folder is, by definition, the folder that contains a system file named System and a system file named Finder.

MacsBug is shipped on the MPW distribution disk #1; you must move MacsBug to the System Folder to install it.

After a successful installation, the message "MacsBug installed" is displayed below the "Welcome to Macintosh" message. The startup application (normally the Finder) is then launched as usual.

Once MacsBug is installed, the only way to remove it is to reboot. To prevent the installation of MacsBug during a boot, hold the mouse button down while booting. To permanently override the installation of MacsBug, simply rename it or remove it from the disk.

- ❖ *Using HFS with the 64K ROM:* If you have MacsBug on an HD-20 (HFS) Startup disk on a machine with the original 64K ROM, there is a seeming conflict with the above mouse-down command, because holding the mouse button down at boot also forces the Macintosh to boot from the floppy disk rather than switch-launching to the HD-20. However, a skillful mouser can accomplish either command simply by knowing that MacsBug is installed first, right after the “Welcome to Macintosh” hello message, and that the HFS code looks for a mouse-down only after the “MacsBug installed” message.

---

---

## Theory of operation: a technical aside

This section provides background information about how MacsBug works. This information is important if you are interested in implementing your own Macintosh debugger; most other readers can skip this section.

---

### The boot process

The state of the world when MacsBug is about to be loaded is fairly complete. The interrupt system, Memory Manager, and ROM-based I/O drivers have already been initialized by the ROM boot code. The boot code initializes the Event Manager, the Font Manager, the Resource Manager, and the file system. (Although the Toolbox is initialized at this point, MacsBug does not use the Toolbox.) The 'DSAT' table is loaded in and the string “Welcome to Macintosh,” contained therein, is displayed.

Next, the loading process of MacsBug takes place as follows: First the boot-blocks code reserves some space (1024 bytes) for MacsBug's own global variables. Then this code looks for the file specified in the boot blocks, as described above. If the file is not found, then the global space is deallocated and the boot process continues normally without installing a debugger.

If MacsBug is found, the data fork (not the resource fork!) of the file is loaded onto the current stack, which is located immediately below the main screen buffer in memory.

- ❖ *Historical note:* For reasons relating to the original Lisa Workshop, the first block (512 bytes) of the MacsBug data fork is stripped off during this loading process.

The boot code then JSRs to MacsBug itself. MacsBug begins its installation process by checking to see if the mouse button is down. If it is, MacsBug aborts the installation and lets the boot process continue without installing itself. If the button is not down, MacsBug determines which kind of machine and microprocessor it is running on, and configures itself accordingly.

At the successful completion of the installation, the message “MacsBug installed” is posted below the “Welcome to Macintosh” message. The boot process then continues by loading 'INIT' resources from the System file.

- ❖ *Technical note:* The boot code looks for 'INIT' resources 0-31 and JSRs to them. These 'INIT' resources are used to set up the keyboard maps ('INIT's 0 and 1), install patches (of type 'PTCH') to ROM code, and so on. 'INIT' 31 extends the system further by looking for any files of type INIT in the System Folder. This facility allows you to install your own startup code without changing the System file.

Finally, the startup application is launched. The startup application is typically the Finder, but can be set to any other application via the Finder's “Set Startup” menu item.

---

## Memory usage

During installation, MacsBug obtains further memory from below the main screen buffer for use as its own screen memory. (MacsBug obtains memory from the same general area in RAM as do RAM disks and caching utilities, based on the location of BufPtr, a Macintosh global variable.) MacsBug offers a full screen display with 40 lines saved. This display uses about 22 K of memory on a Macintosh Plus or Macintosh SE. The display on a Macintosh II requires about 62 K.

The total RAM requirement of MacsBug is approximately as follows:

| Macintosh Plus and SE |             | Macintosh II |             |
|-----------------------|-------------|--------------|-------------|
| Global space          | 1 K         | Global space | 1 K         |
| Screen space          | 22 K        | Screen space | 62 K        |
| Code space            | <u>27 K</u> | Code space   | <u>27 K</u> |
| Total                 | 50 K        | Total        | 90 K        |

MacsBug may not work with some memory-intensive applications on a Macintosh 512K. For example, using MacsBug and the MPW Pascal Compiler on a Macintosh 512K severely limits the size of programs that may be compiled. Two solutions are possible:

1. Remove MacsBug to free up about 50K of RAM.
2. Add additional RAM via the Macintosh Plus logic board upgrade or a compatible third-party hardware upgrade to 1 megabyte or more of RAM.

---

## MacsBug exceptions

When installed, MacsBug puts pointers to itself in many of the hardware exception vectors in addresses \$0000 0000 through \$0000 00FF. It then remains dormant until one of "its" exceptions occurs. The following is the list of exceptions to which MacsBug responds; each is numbered one greater than the corresponding Macintosh System Error number.

| Exception # | Assignment                                                    |
|-------------|---------------------------------------------------------------|
| 2           | Bus error (rarely seen on the Macintosh)                      |
| 3           | Address error (not aligned to a word boundary)                |
| 4           | Illegal instruction (bit pattern not recognized)              |
| 5           | Zero divide                                                   |
| 6           | CHK instruction (array index out of bounds)                   |
| 7           | TRAPV instruction (overflow)                                  |
| 9           | Trace (used to single-step in MacsBug)                        |
| 10          | Line 1010 emulator (the A-trap handler for all Toolbox traps) |
| 11          | Line 1111 emulator (68xxx coprocessor trap interface)         |
| 28          | Level 4 interrupts                                            |
| 29          | Level 5 interrupts                                            |
| 30          | Level 6 interrupts                                            |
| 31          | Level 7 interrupts                                            |
| 47          | Trap \$F instruction (used for setting programmer breaks)     |

MC68000 exception processing is described in the *Motorola 68000 Programmer's Reference Manual*.



Any time an A-trap or other exception listed above occurs, MacsBug intercepts the trap and can thus stop or display the current state of the machine. Single-stepping through 68xxx instructions is possible because MacsBug can set the Trace bit in the status register of the microprocessor. MacsBug saves the ROM-based A-trap handler address in the long word immediately preceding its own A-trap handler routine. Thus, if you need to access the real ROM A-trap handler when MacsBug is installed, you can look at the long word before the address of the current handler.

---

---

## Using MacsBug

The simplest way to get into MacsBug is to generate an exception by pressing the interrupt button. (The interrupt button is the rear button of the programmer's switch on the Macintosh, or the minus key on the numeric keypad on the Macintosh XL.)

To see the application screen while the debugger is active, press the tilde/back quote key (~/`) in the upper-left corner of the keyboard. To restore the debugger's display, press any character key. Repeated presses toggle between the two screens, allowing easy viewing of both the actual code (MacsBug screen) and the results (main screen).

The best way to enter the debugger programmatically is to set a breakpoint in your program by using the system trap called Debugger at the point where you want MacsBug to get control. There are two ways to use this trap. Calling trap \$A9FF drops into MacsBug and displays the message "USERBRK". It then does a normal exception entry into MacsBug (unless you have toggled the DX command—see "Break Commands" later in this chapter).

If you want to display custom debugging information, declare and call the trap with bit 10 set (\$ABFF). When this latter trap is encountered, MacsBug assumes that the top of the user's stack has a pointer to a Pascal string. It prints out the string, displays the message "USERBRK," and does a normal exception entry into MacsBug. As \$ABFF is a procedure call, MacsBug takes care of popping the string pointer off the stack.

Here is a summary of how to declare and use this trap on a per language basis.

---

### Assembly language

#### Declaration

```
_Debugger OPWORD $A9FF ; predefined in the file ToolTraps.a
_DebugStr OPWORD $ABFF ; not predefined - define yourself
```

#### Example calls

```
_Debugger ; enters MacsBug and displays USERBRK

STRING PASCAL ; Asm directive to make sure to push a
 ; Pascal string
PEA #'Entered main loop' ; push address of string on stack
_DebugStr ; enters MacsBug and displays message
```

---

## Pascal

### Declaration

```
PROCEDURE Debugger; INLINE $A9FF;
PROCEDURE DebugStr(str: str255); INLINE $ABFF;
```

### Example calls

```
Debugger; {enters MacsBug and displays USERBRK}
DebugStr('Entered main loop'); {Enters MacsBug and Displays message}
```

---

## MPW C

### Declaration

```
#include <strings.h> /* Required for c2pstr() */
pascal void Debugger() extern 0xA9FF;
pascal void DebugStr(aString) char *aString; extern 0xABFF;
```

### Example calls

```
Debugger(); /*enters MacsBug and displays USERBRK*/
DebugStr(c2pstr("Entered main loop"));
 /*enters MacsBug, displays message*/
```

When MacsBug gets control, it disassembles the instruction indicated by the program counter and displays the contents of the registers. If the exception was caused by a \$A9FF or \$ABFF instruction, MacsBug displays the message "USERBRK", advances the PC to the next instruction, and then disassembles the instruction and displays the registers. It then displays the greater-than symbol (>) as a prompt, indicating that it is ready to accept a command.

- ❖ *Note:* There are two other ways to enter MacsBug: by using 'FKEY' and 'INIT' resources. With ResEdit, a skilled user can create a custom resource of either type whose sole function is described by two simple MC68000 instructions: \$A9FF \$4E75 ( \_Debugger and RTS; that is, the sequence to enter MacsBug).

---

### Warning

Another way to generate an exception that was popular in the past was to add a line such as

```
DC.W $FECE ; generate a line 1111 exception
```

at the point in your program where you wanted MacsBug to get control. (Any value \$F000 through \$FFFF could have been used.) *This method should not be used any more*, as these instructions have been reserved by Motorola for use in their coprocessor interface for the 68020 microprocessor. (For example, in the future these "exceptions" could actually be MC68881 floating-point instructions!)

---

---

---

## The MacsBug command language

Commands consist of a one- or two-character command name followed by a list of zero or more parameters (depending on the command). A return character repeats the last command entered, unless otherwise specified in the command description.

Parameters can be numbers, text literals, symbols, or simple expressions. All parameters can be entered as expressions. Parameters are represented by descriptive words and abbreviations such as *address*, *number*, and *expr*.

MacsBug commands can be divided into five groups: memory, break, A-trap, heap zone, and disassembly commands.

---

### Numbers

As is fitting for a debugger, *all numbers are hex unless otherwise specified*. Decimal numbers are preceded by a number sign (#). Hexadecimal numbers can optionally be preceded by a dollar sign (\$). Numbers can be signed (+ or -). A hex word (four hex characters) preceded by a less-than symbol (<) is sign-extended to a long word.

Here are some numbers in different formats—the formats shown are the same as those displayed by the CV (Convert) command, described later in this chapter.

| Number | Unsigned hex | Signed hex  | Decimal |
|--------|--------------|-------------|---------|
| \$FF   | \$000000FF   | \$000000FF  | #255    |
| 37     | \$00000037   | \$00000037  | #55     |
| -FF    | \$FFFFFF01   | -\$000000FF | #-255   |
| #100   | \$00000064   | \$00000064  | #100    |
| +10    | \$00000010   | \$00000010  | #16     |
| #-32   | \$FFFFFFE0   | -\$00000020 | #-100   |
| <FFFA  | \$FFFFFFFA   | -\$00000006 | #-6     |

---

### Strings

A text literal is a one- to four-character ASCII string bracketed by single quotes ('). If a string is longer than four characters, only the first four characters are used. When used by MacsBug, text literals are right justified in a long word. Here are some examples:

| String | Stored as  |
|--------|------------|
| 'A'    | \$00000041 |
| 'Fred' | \$46726564 |
| '1234' | \$31323334 |

---

## Symbols

The following symbols are generally used to represent the MC68xxx registers:

|                 |                                                 |
|-----------------|-------------------------------------------------|
| RA0, RA1,...RA7 | The contents of address registers A0 through A7 |
| RD0, RD1,...RD7 | The contents of data registers D0 through D7    |
| PC              | The contents of the program counter             |
| SR              | The contents of the status register             |

- ❖ *Note:* In any expression where you want to use the value of one of the main registers, use the Rxx form, as shown above. If you specify A0 for example, it will be interpreted as address A0, a valid hex address, not as register A0.

In addition, the following symbols are used for frequently referenced locations:

- . A period (“dot”) gives the last address referenced
- TP “thePort”—the address of the current QuickDraw port

---

## Expressions

Expressions are formed by operators acting on numbers, text literals, and symbols. The operators are

- + Addition (infix); assertion (prefix)
- Subtraction (infix); negation (prefix)
- @ or \* Indirection operator (two different prefix operators with identical functionality)  
*Note:* The indirection operator uses the long integer at the location pointed to by the operand.
- & Address operator (prefix)
- < Add sign-extended number (infix); sign extension (prefix)

Expressions are evaluated from left to right. All operators are of equal precedence. There is no way to alter the order of evaluation. Here are some valid expressions:

```
RA7+4
3A700-@10C
TP+#24
-RA0+RA1-'FRED'+@@4C50
RA5<FE34
```

(RA5<FE34 is the same as RA5+FFFFFFE34—useful in looking at global variables.)

---

---

## General commands

**?**

**(Help):** Displays a short list of MacsBug commands and their parameters.

**DV**

**(Display Version):** Displays the version, the date and time of creation, and the signature of MacsBug. For example,

```
MACSBUG 5.1B1 17-May-86 00:05:10 <DKA>
```

**RB**

**(Reboot):** Reboots the system.

**ES**

**(Exit to Shell):** Invokes the trap ExitToShell, which causes the current shell to be launched. (The "current shell" is usually the Finder, but you can change it by editing the Finder field of the boot blocks.) The current shell must reside in the System Folder and is logically distinct from the startup application.

❖ *Technical note:* ES may not work with applications that override important system traps. This problem occurs because the application heap gets initialized promptly upon calling the trap ExitToShell; the initialization usually trashes any system patches that were located there. However, there is a hook called IAZNotify, called by InitApplZone, that you can use to restore the world before purging the otherwise necessary routines.

**EA**

**(Exit to Application):** Relaunches the application. This is a faster method than calling ES and relaunching from the Finder.

---

---

## Memory commands

**CV** *expr*

**(Convert):** Displays *expr* as unsigned hexadecimal, signed hexadecimal, signed decimal, text, and binary.

**DM** [ *address* [ *number* ] ]

**(Display Memory):** Displays *number* bytes of memory starting at *address*.

*Number* is rounded up to the nearest 16 bytes. If *number* is omitted, 16 bytes are displayed. If *address* and *number* are both omitted, the next 16 bytes are displayed. The dot symbol ( . ) is set to the address of the beginning of the last block displayed.

If *number* is set to certain four-character strings, memory is symbolically displayed as a data structure that begins at *address*. The strings and the data structures they represent are

'IOPB' Input/Output parameter block for file I/O  
'WIND' Window record  
'TERC' TextEdit record

(Refer to *Inside Macintosh* for a description of these data structures.)

You can usually terminate a DM command by pressing the Backspace key.

**SM** *address expr...*

**(Set Memory):** Places the specified values, *expr...*, into memory starting at *address*. The size of each value depends on the "width" of each expression. The width of a decimal or hexadecimal value is the smallest number of bytes that holds the specified value (four-byte maximum). Text literals are from one to four bytes long; extra characters are ignored. Indirect values are always four bytes long. The width of an expression is equal to the width of the widest of its operands. The dot symbol ( . ) is set to *address*.

**DB** [ *address* ]

**(Display Byte):** Displays a single byte of memory located at *address*. This command automatically calls the convert routine as well, allowing you to see flags easily. DB is useful for looking at the contents of memory-mapped I/O registers. (Using DM will read larger portions of memory; this can have undesired side effects on the peripheral chips being examined.)

**SB** *address [ expr ]*

**(Set Byte):** Places the value of *expr* into the byte located at *address*. If no *expr* is given, then it clears the byte to zero. Like DB, this command is useful for debugging memory-mapped I/O registers.

**Dn** [ *expr* ]

**(Data Register):** Displays or sets data register *n*. If *expr* is omitted, the register is displayed. Otherwise, the register is set to *expr*.

**An** [ *expr* ]

**(Address Register):** Displays or sets address register *n*. If *expr* is omitted, the register is displayed. Otherwise, the register is set to *expr*.

PC [ *expr* ]

**(Program Counter):** Displays or sets the Program Counter. If *expr* is omitted, the program counter is displayed. Otherwise, the PC is set to *expr*.

SR [ *expr* ]

**(Status Register):** Displays or sets the status register. If *expr* is omitted, the status register is displayed. Otherwise the status register is set to *expr*.

TD

**(Total Display):** Displays all of the MC68000 registers and the PC, and disassembles the current instruction that is about to be executed upon stepping, tracing, or going.

TF

**(Total Floating Point):** Displays all of the MC68881 registers if the floating-point chip is present. TF displays register values in both decimal and scientific notation.

F*n* [ *expr* ]

**(Floating Data Register):** Displays or sets the specified MC68881 floating-point data register *n*. You can directly set the register to a decimal or scientific notation value. If *expr* is omitted, the register is displayed. Otherwise, the register is set to *expr*.

FI [ *expr* ]

**(Floating Instruction Address Register):** Displays or sets the MC68881 floating-point instruction address register. If *expr* is omitted, the register is displayed. Otherwise the register is set to *expr*.

FC [ *expr* ]

**(Floating Control Register):** Displays or sets the MC68881 floating-point control register. If *expr* is omitted, the register is displayed. Otherwise the register is set to *expr*.

FS [ *expr* ]

**(Floating Status Register):** Displays or sets the MC68881 floating-point status register. If *expr* is omitted, the register is displayed. Otherwise the register is set to *expr*.

CS [ *address1* [ *address2* ] ]

**(Checksum):** Checksums the bytes in the range *address1* through *address2* and saves that value. The checksum is an exclusive OR of the bytes in the range specified. If *address2* is omitted, CS checksums 16 bytes, starting at *address1*. If *address1* and *address2* are both omitted, it calculates the checksum for the last range specified, saves that value, and compares it to the previous checksum for that range. If the checksum hasn't changed, CS prints CHKSUM T; otherwise it prints CHKSUM F.

❖ *Note:* For checksumming memory in conjunction with A-traps, see the AS command. For checksumming after every 68xxx instruction, see the SS command.

---

---

## Break commands

**BR** [ *address* [ *count* ] ]

**(Break):** Sets a breakpoint at *address*. *Count* specifies the number of times that the breakpoint should be executed before stopping the program. If *count* is omitted, the program is stopped the first time the breakpoint is hit. If *address* is omitted, all breakpoints are displayed. You can set a maximum of eight different breakpoints.

**CL** [ *address* ]

**(Clear):** Clears the breakpoint at *address*. If *address* is omitted, all breakpoints are cleared.

**G** [ *address* ]

**(Go):** Executes instructions starting at *address*. If *address* is omitted, execution begins at the address indicated by the program counter. Control does not return to MacsBug until an exception occurs.

**GT** *address*

**(Go Till):** Sets a one-time breakpoint at *address*, then executes instructions starting at the address indicated by the program counter. This breakpoint is automatically cleared after it is hit.

**T**

**(Trace):** Traces through one instruction. Traps are treated as single instructions.

If the next instruction to be executed is a JSR to a currently unloaded segment, you will see the LoadSeg (\$A9F0) trap instead of the JSR. Tracing through that instruction will not work normally. If you wish to trace through the LoadSeg trap, you need to set a low-memory global at location \$12D to a nonzero value. Do a SB 12D 1 to enable tracing through the LoadSeg call. Next, Go (G). You will break at an RTS instruction. Trace once (T) to see the absolute location that you are about to jump to. Trace again and you will be at the first step of the routine that is now loaded into memory. To turn off tracing through LoadSeg calls, simply execute SB 12D to clear the LoadSeg low-memory flag.

**S** [ *number* ]

**(Step):** Steps through *number* instructions. If *number* is omitted, just one instruction is executed. Traps are not considered to be single instructions. Step will display each instruction as it is executed.



**SS** *address1* [ *address2* ]

**(Step Spy):** Calculates a checksum for the specified memory range, then does a Go; it then checks the checksum before each 68xxx instruction is executed, and breaks into MacsBug if the checksum does not match. If *address2* is omitted, SS checksums the long word at *address1*. This feature is turned off by entering MacsBug via the programmer's switch or by SS terminating when the checksum has changed.

Step Spy is very slow. Step Spy is nevertheless useful for detecting what routines are stepping on a specific place in memory. If checking memory at every A-trap is sufficient for your needs, use the AS command, described below. (The slow motion capability of SS, however, can be useful in its own right to examine how the Finder zooms windows, for example. Think of it as a tool to study graphics algorithms.)

**ST** *address*

**(Step Till):** Steps through instructions until *address* is encountered. Unlike Go Till, this command does not set a breakpoint. Thus it can be used to step through, and stop in, ROM.

**MR** [*offset* ]

**(Magic Return):** This command lets you quickly step through an entire procedure. It does this by replacing the return address on the stack with an address in MacsBug. When the procedure returns, control first goes to MacsBug. MacsBug restores the original return address and then executes an RTS in trace mode. MacsBug breaks at the instruction after the call to this procedure.

If *offset* is omitted, then MacsBug assumes the return address is on the top of the stack. If *offset* is present, then it is interpreted relative either to the stack pointer (A7) or the frame pointer (A6). If *offset* is less than A6, then MacsBug assumes that the return address is at A7 + *offset*. If *offset* is equal to A6, then MacsBug assumes that A6 is the frame pointer placed on the stack by the LINK instruction as a procedure preamble. If *offset* is greater than A6, then it is assumed to be the frame pointer of a procedure located higher in the calling order.

Here are some examples:

MR

Use MR if the called procedure has not changed the stack.

MR *n*

Use this command if the called procedure has pushed *n* bytes onto the stack.

MR RA6

Use this command if the called procedure has done a LINK instruction.

MR @RA6

Use this command to break after the caller of the called procedure returns.

**DX**

**(Debugger Exchange):** Normally, if either the \$A9FF or the \$ABFF A-trap (two forms of the debugger trap) is executed, program execution halts and the debugger is activated. DX allows you to control whether or not program execution halts. Note that the \$ABFF trap will still print a string; thus with debugger entry disabled, an effect similar to that of the AT command occurs—that is, the Macintosh screen alternates between the debugger and the program. The default is to stop at debugger traps.

---

---

## A-trap commands

The A-trap commands are used to monitor “1010 emulator” traps, used to call the Macintosh ROM. These commands take up to six parameters (*trap1*, *trap2*, *address1*, *address2*, *D1*, and *D2*). These parameters indicate which traps and other conditions should be monitored:

- *Trap1* and *trap2* specify the range of the traps. If only *trap1* is specified, the command is invoked for *trap1*. If *trap1* and *trap2* are specified, the command is invoked for all traps in the range *trap1* through *trap2*. The defaults are \$A000 and \$AA00.  

|                  |                                   |
|------------------|-----------------------------------|
| \$A000–\$A0FF    | Operating system traps            |
| \$A800–\$A9FF    | Toolbox traps                     |
| 0–\$6F           | Shortcut expressions for OS traps |
| \$70 and greater | Interpreted as Toolbox traps      |
- *Address1* and *address2* specify a range of memory addresses within which traps should be monitored. The defaults are 0 and \$FFFFFFFF.
- *D1* and *D2* specify the values of data register 0 within which traps should be monitored. They are treated as unsigned numbers. The defaults are 0 and \$FFFFFFFF.

Thus, if no parameters are given, all traps are monitored.

A-trap commands allow two commands to take place simultaneously. The trick to using the A-trap commands is to know that there are separate flags for tracing and breaking, and that separate globals are used for storing the general trap range (GTR) and the breaking trap range (BTR):

- Any A-trap command (AA, BA, AT, AB, AS, AH, AR) sets the tracing flag. In addition, any command except AS can supply a trap range, which is always stored in the GTR variable.
- Executing an AB or BA also sets the breaking flag. It also saves the trap range you supplied in both the GTR and BTR variables.

In the previous release of MPW, any A-trap command would clear all flags, but in MPW 2.0 only AX clears all flags. If you are a “casual” A-trap user, execute AX before executing any A-trap commands in order to avoid undesired breaks. However, for the MacsBug power user, combined A-trap commands can be very useful.

Here is an example of how you can use combined A-trap commands. If you wish to view (trace) all of the file system traps called from your application but also want to break at the next Open call that you make, you should type (and in this order!):

```
>BA Open
>AA 0 17 (shorthand for file system traps)
```

The AA command is entered second so that its range overwrites the GTR supplied by the BA command. This way you can view (trace) the “wider” range of traps while breaking on the “smaller.”

**BA** [ trap1 [ trap2 [ address1 [ address2 [ D1 [ D2 ] ] ] ] ] ]

**(Break in Application):** Causes a break when the conditions specified by the parameters are satisfied and the trap is being called from the application rather than from the ROM. *Address1* and *address2* are automatically set to ApplZone and BufPtr. Therefore you can use this command to get back to the application when in ROM. Simply type BA, (return),and then G, (return). MacsBug will be entered at the next trap called by code located in the application heap. To break on ROM calls as well (or traps called from the system heap or elsewhere), use AB, described below.

**AA** [ trap1 [ trap2 [ address1 [ address2 [ D1 [ D2 ] ] ] ] ] ]

**(Application A-trap Trace):** Traces and displays each A-trap called from the application heap without breaking if the conditions specified by the parameters are satisfied. AA continues to display A-traps until you press the interrupt button. AA allows you to monitor only the traps that the application calls, and thus can be useful for checking and measuring performance. To monitor all traps called, including calls made from inside the ROM and traps called from the system heap, use the AT command.

**AB** [ trap1 [ trap2 [ address1 [ address2 [ D1 [ D2 ] ] ] ] ] ]

**(A-trap Break):** Causes a break when the conditions specified by the parameters are satisfied. AB without any parameters will stop at the very next trap executed anywhere by the Macintosh. To stop at the next trap called by the current application, use BA instead.

**AT** [ trap1 [ trap2 [ address1 [ address2 [ D1 [ D2 ] ] ] ] ] ]

**(A-trap Trace):** Traces and displays each A-trap without breaking, when the condition specified by the parameters is satisfied. AT continues to display all A-traps until you press the interrupt button. If you wish to see just the traps called by the current application, use AA instead.

For example, to see all QuickDraw calls displayed, regardless of who calls them, you could type

```
>AT A86C A8FB
```

**AH** [ trap1 [ trap2 [ address1 [ address2 [ D1 [ D2 ] ] ] ] ] ]

**(A-trap Heap Zone Check):** Checks the heap zone for consistency just before executing each trap in the specified range. If an inconsistency is found, it displays the addresses of the two memory blocks in question.

**AR** [ trap1 [ trap2 [ address1 [ address2 [ D1 [ D2 ] ] ] ] ] ]

**(A-trap Record):** Whenever the parameter constraints are satisfied by an A-trap call, information about the call is recorded. The trap name, PC, A0, D0, and the time are always saved. If the call was for an OS trap, 32 bytes pointed at by A0 are recorded; otherwise 32 bytes pointed at by A7 (the stack pointer) are saved. To display the current saved information, type AR with no arguments.

This command is especially useful for tracking down crashes in the Macintosh ROM. For example, the command

```
>AR 0 1000 @2AA @114
```

records traps 0 through 1000 (all traps), from ApplZone (\$2AA) through HeapEnd (\$114), so it will record the last trap call made from anywhere in the application heap (the application's code).

**AS** *address1* [ *address2* ]

**(A-trap Spy):** Calculates a checksum for the specified memory range, checks it before each A-trap that is called, and breaks into MacsBug if the checksum does not match. If *address2* is not specified, AS checksums the long word at the given address. Use SS if you want the range of memory to be checked before every 68xxx instruction rather than before every A-trap only. AS is turned off by AX.

**AX**

**(A-trap Clear):** Clears all A-trap commands and turns off heap scrambling.

---

---

## Heap zone commands

The heap zone commands act upon the current heap zone. When MacsBug is started up, the current heap zone is the application heap zone. You can set the current heap zone by using the HX command. Several commands cause MacsBug to scramble the heap zone. When you scramble the heap zone, all the relocatable blocks are rearranged. (See HS below.) Scrambling is useful for finding illegally used pointers to relocatable data structures.

**HX** [ *address* ]

**(Heap Exchange):** Sets the current heap to *address*. If no *address* is given, then HX toggles the current heap zone between the system heap zone and the application heap zone. In any case, HX displays the resulting current heap address.

**HC**

**(Heap Check):** Checks the consistency of the current heap zone, and displays the addresses of inconsistent memory blocks as well as the address of the current heap.

**HS**

**(Heap Scramble):** Scrambles the heap zone by calling CompactMem each time a Memory Manager trap is called. Heap Scramble is useful for finding nil handles, dangling pointers, and other mistakes made in memory management. It also performs a heap check after each scramble. If the heap looks questionable, MacsBug will inform the user. Use AX to turn Heap Scramble off.

## HD [ *mask* ]

**(Heap Dump):** *Mask* is optional. Whether or not *mask* is used, it displays each block in the current heap zone in the following form:

*blockAddr type size [flag MP\_location] [\*] [refNum restype ID]*

- *blockAddr* points to the start of the memory block.
- *type* is one of the following letters:
  - F Free block
  - P Pointer
  - H Handle to a relocatable block
- *size* is the physical size of the block, including the contents, the header, and any unused bytes at the end of the block.
- For handles (*type* H), *flag* is either blank if not purgeable or a P if purgeable. Then *MP\_location* is displayed, which is the address of the master pointer to the relocatable block.
- The asterisk (\*) marks any locked object (nonrelocatable blocks and locked relocatable blocks).
- For resource file blocks, three additional fields are displayed: the resource's reference number, resource type, and ID number. If *mask* is omitted, then the dump is followed by a summary of the heap blocks. If *mask* is present, then only certain types of heap blocks are displayed. *Mask* should be one of the following text strings:
  - 'h' Relocatable blocks (*handles*)
  - 'p' Nonrelocatable blocks (*pointers*)
  - 'f' Free blocks
  - 'r' All resource blocks
  - 'XXXX' Only resource blocks of the type 'XXXX'

If *mask* is used, the heap summary takes this form:

CNT ### <# of blocks of *mask* type> <# bytes in those blocks>

You can prematurely terminate an HD command by pressing the Backspace key.

The dot address (.) is set to the last block of memory displayed by HD.

## HT [ *mask* ]

**(Heap Total):** Displays just the summary line from a heap zone dump. *Mask* works just as it does with the HD command (described above).

## SC

**(Stack Crawl):** Assumes that LINK / UNLK A6 has been religiously performed at the beginning and end of each procedure or function. (The \$D+ directive in Pascal, and the -g and -ga options in C, force these instructions to be performed.) The output format is as follows:

SF @<stack frame location> <address of call to procedure>

For example,

SF @0D633C ProcName+3A

means that the currently executing procedure or function has its local stack frame at \$D633C and was called from ProcName+\$3A (which is not the return address!). If the program counter is in the ROM, SC may not work properly.

---

---

## Disassembler commands

### SX

**(Symbol Exchange):** Determines whether or not symbols are displayed. By default, symbols are turned on. SX affects any command that takes an address. Using symbols allows you to disassemble an instruction list (IL) or break (BR) on a procedure or function name. For example,

```
>IL ProcName+58
```

disassembles code starting at 58 bytes (hex) into the procedure called ProcName, and

```
>BR ProcName+58
```

sets a breakpoint at the same location. (This also works for GT, ST, DM, and so on.)

When searching for symbols, MacsBug searches the current heap (set by the HX command). MacsBug goes through the current heap's memory, looking for locked blocks of memory. Then, within locked blocks, MacsBug first looks for a LINK A6 instruction followed by a matching UNLK A6 instruction. Then MacsBug looks for either an RTS or a JMP (A0) instruction.

Immediately following one of these last two instructions should be an 8-character symbol. This symbol must be exactly eight characters long; it should be padded with blanks if it is less than eight characters. Some compilers set the high bit on the first character of the symbol, but MacsBug clears this bit. In addition, if the high bit of the second character is set, MacsBug expects a 16-character name (used mainly for method names in MacApp-generated code). To see all of the symbols that are valid at any given moment, use the SD command (described next).

Turning symbols off is helpful for two reasons. First, every symbol lookup traverses the current heap, and therefore may degrade the speed of the disassembly. Secondly, if you prefer always seeing a dump of code in hex rather than symbols (useful when looking at ROM code, for example), turning off symbols will guarantee a hex dump of your code. This hex dump displays the main opcode word followed by two extension words which may or may not apply to the particular instruction disassembled.

### SD [ *address* ]

**(Symbol Dump):** Displays a list of the procedure names that can be found in the current heap zone. The search criteria are based on looking in each block of memory whose locked bit is set. In addition, a LINK A6 and its matching UNLK A6 must be found, followed by either a JMP (A0) or an RTS. The 8-character debugging name follows. Valid debug symbols must consist of ASCII characters in the range \$20-\$5F (space-underscore), inclusive. This command optionally allows you to specify a starting location for the symbol dump.

## DH *number*

**(Disassemble Hex):** Disassembles the hex byte, word, or long word input. Typing just one byte allows you to see the general class of instructions, as *number* is left-aligned in a long word padded to the right with zeros. (Typing DH 10, DH 20, and DH 30, for example, shows by induction that these instruction groups are the Move.B, Move.W, and Move.L classes, respectively.)

This command is useful as a poor man's assembler. For example, if you wanted to use the RESET instruction and could not remember what its opcode was, you could type DH 4E71 as a first guess and DH would display NOP. Trying DH 4E70 as a second guess would reveal the actual RESET instruction.

## ID [ *address* ]

**(Instruction Disassemble):** Disassembles one line at *address*. If *address* is omitted, the next logical location is disassembled. ID sets the dot symbol ( . ) to the *address*.

If the code has symbols compiled with it via the \$D+ directive in Pascal or the -g option in C, and symbols have been turned on with the SX command, each address is automatically displayed as a routine name plus an offset.

## IL [ *address* [ *number* ] ]

**(Instruction List):** Disassembles *number* lines starting at *address*. If *number* is omitted, a screenful of lines (typically 16) is disassembled. If both *number* and *address* are omitted, a screenful of lines is disassembled starting at the next logical location. This command sets the dot symbol ( . ) to the *address*.

If the code has symbols compiled with it via the \$D+ option in Pascal or the -g option in C, and symbols have been turned on with the SX command, each address is automatically displayed as a routine name plus an offset.

You can prematurely terminate an IL command by pressing the Backspace key.

## F *address count data* [ *mask* ]

**(Find):** Searches *count* bytes from *address*, looking for *data*, after ANDing the target with *mask*. As soon as a match is found, the *address* and value are displayed, and the dot symbol ( . ) is set to that *address*. To search the next *count* bytes, simply press Return. The size of the target is determined by the width of *data*; it is limited to 1, 2, or 4 bytes.

For example, to find a RESET instruction in a program loaded into a Macintosh Plus, you could type

```
>F CB00 EFFFF 4E70
```

where CB00 is the beginning of the application heap, EFFFF represents the length of the application heap (roughly), and 4E70 is the RESET instruction.

## WH *expr*

**(Where):** Takes an expression, which can be a symbolic name, and displays the location of the first routine that it finds whose name matches the expression. ROM symbolic names are 10-character names; and RAM symbolic names are 8-character names.

If *expr* is less than \$AA00, this command displays the address corresponding to the trap with that number. All of the following commented commands, for example, give the same result:

```
>WH EXITTOSHELL ; full name
>WH A9F4 ; full trap word
>WH 1F4 ; shortcut
>WH 40F6D8 ; address of ExitToShell in the 128K ROM
```

Namely,

| Trap word | Address | Name        |
|-----------|---------|-------------|
| A9F4      | 40F6D8  | EXITTOSHELL |

The shortcut method of inputting trap numbers interprets \$0-\$6F as OS traps, and all other traps as Toolbox traps.

If *expr* is preceded by the address operator (&), then the expression is forced to be evaluated as an address. This feature is useful for examining system patches whose addresses are often less than \$AA00, the default address boundary.

If *expr* is greater than or equal to \$AA00 and less than RomBase, then the address is interpreted as a user routine in RAM, and a symbolic location and code segment will be displayed if possible.

If *expr* is in ROM then the trap whose code is closest to that address is displayed.

WH is useful for finding out where you were when an error occurred. If the address expression is in RAM and the WH function returns "PRGM AT \$\$\$\$" you can use the command HD 'CODE' to list the code segments. Then, by comparing the locations of 'CODE' segments and the current PC, you can determine which segment you are in.

---

---

## MacsBug summary

### General commands

? (Help)  
DV (Display Version)  
RB (Reboot)  
ES (Exit to Shell)  
EA (Exit to Application)

### Memory commands

CV *expr* (Convert)  
DM [ *address* [ *number* ] ] (Display Memory)  
SM *address* *expr*... (Set Memory)  
DB [ *address* ] (Display Byte)  
SB *address* [ *expr* ] (Set Byte)



**Dn** [ *expr*] (Data Register)  
**An** [ *expr*] (Address Register)  
**PC** [ *expr*] (Program Counter)  
**SR** [ *expr*] (Status Register)  
**TD** (Total Display)  
**TF** (Total Floating-Point Display)  
**Fn** [ *expr*] (Floating-Point Register)  
**FC** [ *expr*] (Floating-Point Control Register)  
**FI** [ *expr*] (Floating-Point Instruction Address Register)  
**FS** [ *expr*] (Floating-Point Status Register)  
**CS** [ *address1* [ *address2*]] (Checksum)

### Break commands

**BR** [ *address* [ *count*]] (Break)  
**CL** [ *address*] (Clear)  
**G** [ *address*] (Go)  
**GT** *address* (Go Till)  
**T** (Trace)  
**S** [ *number*] (Step)  
**SS** *address1* [ *address2*] (Step Spy)  
**ST** *address* (Step Till)  
**MR** [ *offset*] (Magic Return)  
**DX** (Debugger Exchange)

### A-trap commands

**BA** [ *trap1* [ *trap2* [ *addr1* [ *addr2* [ *D1* [ *D2* ]]]]]] (Break in Application)  
**AA** [ *trap1* [ *trap2* [ *addr1* [ *addr2* [ *D1* [ *D2* ]]]]]] (Application A-trap Trace)  
**AB** [ *trap1* [ *trap2* [ *addr1* [ *addr2* [ *D1* [ *D2* ]]]]]] (A-trap Break)  
**AT** [ *trap1* [ *trap2* [ *addr1* [ *addr2* [ *D1* [ *D2* ]]]]]] (A-trap Trace)  
**AH** [ *trap1* [ *trap2* [ *addr1* [ *addr2* [ *D1* [ *D2* ]]]]]] (A-trap Heap Zone Check)  
**AR** [ *trap1* [ *trap2* [ *addr1* [ *addr2* [ *D1* [ *D2* ]]]]]] (A-trap Record)  
**AS** *address1* [ *address2*] (A-trap Spy)  
**AX** (A-trap Clear)

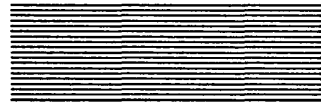
### Heap zone commands

**HX** [ *address*] (Heap Exchange)  
**HC** (Heap Check)  
**HS** [ *trap1* *trap2*] (Heap Scramble)  
**HD** [ *mask*] (Heap Dump)  
**HT** [ *mask*] (Heap Total)  
**SC** (Stack Crawl)

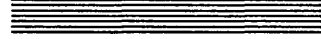
### Disassembler commands

**SX** (Symbol Exchange)  
**SD** [ *address*] (Symbol Dump)  
**DH** *number* (Disassemble Hex)  
**ID** [ *address*] (Instruction Disassemble)  
**IL** [ *address* [ *number*]] (Instruction List)  
**F** *address* *count* *data* [ *mask*] (Find)  
**WH** *expr* (Where)





## **Chapter 12**



# **Performance Measurement Tools**

---

---

## About performance measurement tools

MPW 2.0 provides a set of performance measurement tools to aid professional software developers in measuring and improving the performance of their applications. In essence, the Program Counter (PC) register is sampled just often enough to obtain a statistically accurate estimate of the program's actual use of time. The code is measured in "buckets" of two or more bytes and the result is output to a text file, PerfDump. You can then analyze these results by running a report generator, PerformReport. PerformReport merges the output file with a linkmap of the measured code resources to produce a list of procedures, sorted by the number of PC samples found within the procedure.

---

### Warning

The performance measurement tools are designed for temporary inclusion in an application, desk accessory, or driver for purposes of measuring performance. They are not designed for inclusion in commercial products, because they rely on low-level system mechanisms that are not guaranteed to function correctly on all future machines.

---

The memory management strategy for the performance tools is based on the assumption that developers wishing to measure performance will likely have a machine larger than the smallest target machine for their applications. Thus, they can use performance tools that require some additional memory without severely impacting the application's memory management strategy.

The best way to use these tools depends upon your particular environment and the code you want to test. These considerations are discussed in the section "Implementation Issues" later in this chapter.

---

## Components of performance tools

The performance tools consist of the following pieces:

- **A library file (PerformLib.o):** This file is in the {Libraries} folder. Link with this file.
- **Interface files for Pascal (Perf.p) and C (Perf.c):** These files are in the interface folders: {PInterfaces} for Pascal, and {CIncludes} for C. These are the files that you use or include in the source files for your application. These interfaces depend only on the standard Macintosh memory types files: MemTypes.p for Pascal and Types.h for C.

An assembly-language interface has not been provided for the performance tools. Assembly-language programmers can use either the Pascal or the C interface. Both go directly to the Pascal and assembly-language implementation in PerformLib.o.

- **Sample programs, makefiles, and instructions for execution:** These files are in the Examples folders: {PExamples} for Pascal, and {CExamples} for C. For more details about the sample programs, see the appropriate language reference manuals. Instructions for running the TestPerf sample programs are included in the Examples folder.

- **PerformReport (a tool for analyzing performance data and producing reports):** This tool is found in the {MPW}Tools: folder. For detailed information about the tool, see the command pages in Part II. For detailed instructions on how to run this tool, see the instructions in the examples folder. Examples of the output from this tool are discussed below.
- **ROM map files:** You'll find three ROM maps in the folder {MPW}ROM.Maps: MacIIROM.map, MacSEROM.map, and MacPlusROM.map. These files are combined with the link map file for your application, to provide symbolic information about the performance data. The files are used as input to the report generator.

---

## Requirements for using performance tools

To use the performance tools, you need to add calls to these routines in your application, desk accessory, or driver:

- **InitPerf** specifies the types of measurements to be performed, and allocates storage. This should be called once near the beginning of your code.
- **TermPerf** stops measurements (if active), and frees the storage. TermPerf is called once after InitPerf succeeds, and measurement is finished.
- **PerfControl** starts and stops measurements. PerfControl must be called once (after InitPerf) to start measurements. Use PerfControl to avoid taking measurements in idle loops, dialog boxes, alerts, and other places where the user response time determines performance.
- **PerfDump** stops measurements (if active), and writes the performance data to an output file. Called after measurements are collected for reporting.

For more details about these routines, see the appropriate language reference manual.

---

---

## How performance measurement works

The performance measurement tools are designed to give you useful information about the performance of a program without severely altering the user responsiveness or memory requirements—that is, without changing the characteristics of what is being measured. However, the act of measurement necessarily alters what is being measured in the ways summarized below.

---

### Program Counter sampling

The fundamental idea behind the performance measurement tools is to sample the Program Counter (PC) register frequently enough to obtain a statistically accurate estimate of the actual program performance, but infrequently enough so that overall performance is not affected. The performance measurement tools use the Vertical Blanking signal (VBL) on 64K ROMs and the Time Manager on 128K and larger ROMS.

The Time Manager allows 1 ms resolution in sampling, but this imposes about a 20% performance degradation. A value of 4 to 10 ms reduces the performance degradation to 4 to 10%. Use of the VBL signal on old ROMs imposes a sampling rate of approximately 60 times per second (16 ms).

## **A restriction**

If your application directly uses the VIA Timer1 (or some software that uses it, such as the sound generator or the Time Manager) then you might not be able to use these performance measurement tools. On old ROMs, the performance measurement tools may not work correctly with programs that make use of VBL tasks.

---

## **Warning**

If you set the sampling interval too low for your machine, the performance tools may crash or cause your program to run very slowly. It is best to start with a high sampling interval, such as 10 or 20 ms, and decrease it only after experience allows you to predict the effect of the shorter interval. For example, if measurements taken with a sampling rate of 10 ms cause your program to run 10% slower, then it is probably safe to increase the sampling rate to every 5 ms at a cost of having the program run 20% slower.

---

## **Bucket counts**

The performance tools require 2 bytes of memory for a counter for each "bucket" of code that is measured. For instance, for a 100K tool or application, using a bucket size of 16 bytes, about 12,800 bytes are required for the counters. If the ROM is measured, an additional 8K, 16K, or 32K bytes (for 64K, 128K, or 256K ROMs) is required.

If your program spends a substantial amount of time outside CODE segments and ROM, then you may want to measure RAM "misses." Because RAM can be quite large, a second (generally larger) bucket size can be specified for RAM "misses." And you can control the amount of RAM to be measured by using a low address to start setting up buckets and a high address for the last bucket. If the RAM misses are measured, additional memory is required.

The sum of all memory required for counters is allocated as a single contiguous block at the time PerfInit is called. For this reason, you should call PerfInit fairly early in your initialization, before memory becomes fragmented.

In addition to the memory for bucket counters, the performance tools will use one master pointer for a handle to some information, and will allocate a few small structures with NewPtr calls.

---

---

## Using performance measurement tools

This section provides a general overview of the steps necessary to install the performance measurement routines into your code. The MPW C and MPW Pascal references provide specifics for each of these steps.

You need only make a few changes to install the performance tools in your code. The changes are basically the same, whether you are developing an application, a desk accessory, an MPW tool, or a driver. It is even possible to install performance tools in ROM.

Here are the steps:

1. *Install under conditional compilation.* After measuring the performance of your program, you will probably want to make changes, test the changes for correctness, and then repeat the measurements to verify the performance improvements. While making and testing changes, it is very important not to include the performance tools, unless you are confident that the changes do not introduce any new bugs. If your code should terminate early, for any reason, then the normal system recovery techniques (in MacsBug, calls such as G STOPTOOL under the MPW Shell or ES from an application) would not work. In such a case, within a few milliseconds after the system tries to reuse the memory occupied by the performance tools, a timer interrupt occurs and a system error results. The system error would probably force rebooting the system. For this reason, it is advisable to include the performance tools under a conditional flag. Please consult the appropriate language reference.
2. *Include the interface.* In the main body of your code you need to reference the interface file for the performance tools.
3. *Provide a pointer to a block of variables.* For an application or MPW tool, you should declare a global variable. If you are developing a desk accessory, driver, or ROM that does not have global variables, then you need to be somewhat creative in finding a location for the pointer. The choices include: a local variable on the stack (assuming the stack frame will persist long enough), a field of a block allocated and locked down in the heap, or a low memory location. In any event, the address of the location allocated for the pointer must be passed to the performance routines.
4. *Initialize the performance tools.* Somewhere near the beginning of your code, when large chunks of memory are available, you need to initialize the performance tools.

---

### Warning

Once your code has initialized the performance routines successfully, it is important that the termination routine described below be called before your code terminates. Failure to do so will almost always result in a fatal system crash.

---

5. *Turn on the measurements.* After initialization has succeeded, you can start measurements at any point in your code. The call that starts (and stops) measurements returns the current on-off state as a Boolean value, in order to support a wide variety of user interfaces to the tools. You can call `PerfControl` with a second argument of `false` in order to turn performance measurements off. This is useful for disabling sections of code that you don't want to measure, such as the event loop of an application, a dialog box where user response time will dominate the compute time, parts of the application that rely on the VIA timer, and so on.
6. *Dump the results.* When you reach the end of the code to be measured, you must make a call to have the performance counters written into a text file. If the dump routine encounters any I/O, memory management, or other system errors, it returns a nonzero return code. You can examine this code to determine the nature of the problem.
7. *Terminate cleanly.* After dumping the counters to a text file, you must terminate the performance measurement tools cleanly—that is, remove the interrupt routine and free the memory associated with the performance global variables and counters.

---

---

## Performance reports

When your code has completed its execution, a call is made to `PerfDump` in the performance measurement tools, which generates a performance output file showing the results of the bucket counts. You can then analyze this data by using the tool `PerformReport`. Examples of both the performance output and report files appear in this section. See Part II for a command page describing the tool `PerformReport`.

---

### Performance output file

The results of the performance tests are output to a performance data file when `PerfDump` is called. This file is a text file containing the bucket locations and counts. You should call `PerfDump` at the very end of the test, so that no interference with program I/O should occur. The performance output file is not opened until `PerfDump` is called.

Below is an example of a performance output file as generated by a call to `PerfDump`. Some repeated lines have been omitted, as indicated by "...".

Notice that the performance data is arranged on a per segment basis. Only nonzero buckets are reported; in other words, missing buckets had a hit count of zero. (`PerfDump` has an option to produce a bar graph to the left of the Hits column. That option was not exercised in this example.)



Performance Parameters

=====

Bytes per bucket, Code and ROM: 8  
Bytes per bucket, RAM: 4  
Sampling Interval: 4 ms

Performance Summary

=====

Total hits outside of the sampled segments: 2  
Maximum hits in one bucket: 872  
Total hits in all buckets: 3222

Performance Data

=====

Offset Hits | Segment 117 size 20000

=====

|       |     |  |
|-------|-----|--|
| 52F8  | 1   |  |
| 5300  | 1   |  |
| 53C8  | 12  |  |
| 53E8  | 1   |  |
| 53F0  | 2   |  |
| 5400  | 1   |  |
| 5428  | 1   |  |
| 5728  | 872 |  |
| ...   |     |  |
| 1B830 | 9   |  |
| 1B838 | 53  |  |
| 1B840 | 41  |  |
| 1B848 | 61  |  |
| 1B850 | 41  |  |

Offset Hits | Segment 253 size 1FFFFFF

=====

|        |     |  |
|--------|-----|--|
| D6A0   | 1   |  |
| 287D0  | 40  |  |
| 1E6134 | 1   |  |
| 1E8990 | 1   |  |
| 1FB20C | 101 |  |

Offset Hits | Segment 13 size B68 name STUDIO

=====

Offset Hits | Segment 12 size 71E name SACONSOL

=====

...  
Offset Hits | Segment 5 size D0 name ROMSEG2

=====

|     |    |  |
|-----|----|--|
| 70  | 2  |  |
| 88  | 5  |  |
| 90  | 10 |  |
| ... |    |  |
| B0  | 4  |  |
| B8  | 2  |  |
| C0  | 3  |  |

```

Offset Hits | Segment 4 size 136 name ROMSEG1
=====
 10 9 |
 18 3 |
 20 5 |
...
110 19 |
118 58 |
120 46 |

Offset Hits | Segment 3 size 8C name SEG2
=====
 50 3 |
 58 3 |
 60 9 |
 68 1 |
 70 14 |
 78 1 |

Offset Hits | Segment 2 size D0 name SEG1
=====
 10 2 |
 18 18 |
 20 4 |
...
 A8 7 |
 B0 12 |
 B8 18 |

Offset Hits | Segment 1 size 101C name Main
=====
F38 43 |
F40 116 |
F48 78 |
F50 19 |
F58 77 |
F60 66 |
F68 56 |

```

---

## Analyzing the results with PerformReport

Once the performance data file has been generated, you are ready to run the report generator, a tool called PerformReport. This tool merges the performance output file with a linkmap of the measured code resources to produce a list of procedures, sorted by the number of PC-samples found within the procedure. (See Part II for more information on PerformReport.) An example of the contents of this file is shown below.

```
PerformReport -- Merges Linker Output and Performance Dump January 30, 1987
```

```
Reading Link Map file: "temp3"
```

```
Reading Performance Measurements file: "Perform.Out"
```

```
PerformReport Parameters:
```

```
 8 bytes per bucket, ROM and CODE.
 4 bytes per bucket, RAM.
 2 hits outside code measured.
3224 hits total, 0.0% outside the segments.
 872 maximum hits in one bucket.
```

```
Procedures by possible hits (showing Probable % of time):
```

| Num Segment      | Procedure          | Def   | Prob  | Poss | Prob% |
|------------------|--------------------|-------|-------|------|-------|
| 117              | Main : ATRAP68020  | 497   | 436   | 872  | 28.9% |
| 117              | Main : CHKSLOT     | 0     | 436   | 872  | 13.5% |
| 117              | Main : DSPATCH     | 0     | 0     | 872  | 0.0%  |
| 117              | Main : RSECT       | 474   | 13    | 26   | 15.1% |
| 1                | Main : %I_MUL4     | 399   | 14    | 56   | 12.8% |
| ...              |                    |       |       |      |       |
| 4                | ROMSEG1 : ROMW100  | 5     | 0     | 0    | 0.1%  |
| 117              | Main : LVL1INT     | 1     | 0     | 1    | 0.0%  |
| 117              | Main : TFSDISPATCH | 1     | 0     | 0    | 0.0%  |
| 117              | Main : LVL2INT     | 0     | 0     | 1    | 0.0%  |
| Total Reported = |                    | 67.6% | 32.1% |      | 99.8% |

```
PerformReport: That's All Folks!
```

---

## Adding identification lines to a data file

After emitting a title line, and giving the names of the files being read, PerformReport has an option (-e) to echo lines from the head of the measurements file until the phrase "Performance Data" is encountered. This option allows you to add identification lines at the head of performance measurement files. Various parameters are gathered from lines that begin with special keywords. Here are the keywords with the phrases they head:

```
Bytes Bytes per bucket
Total Total hits
Maximum Maximum hits
Performance Performance Data
```

You are free to add comment lines at the head of a data file, as long as the comment lines do not begin with these keywords.

---

## Interpreting the performance report

PerformReport translates the bucket hit information into procedure-based information. Because procedures can span buckets, there may be some uncertainty about how bucket hits are related to procedure hits. PerformReport attempts to deal with this uncertainty by classifying hits into several categories:

- |                   |                                                                                                                                                                                                                                                                                                               |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Definite          | When a bucket is completely contained in a single procedure, all hits in the bucket are counted as definite hits in that procedure.                                                                                                                                                                           |
| Possible/Probable | When a bucket is partially contained in several procedures, all hits in the bucket are counted as possible hits in each procedure; in addition, the hits in the bucket are counted as probable hits in a particular procedure, based on the amount of the bucket that is covered by the particular procedure. |

Please realize that the concept of probable hits is not intended to give an accurate statistical picture of the situation. What happens in practice is that buckets are frequently covered by two procedures, and almost all of the hits occur in one procedure or the other. The intent behind “possible and probable” hits is to give you some feeling for the accuracy of the resulting data.

In the Pascal example TestPerf.p, the possible hits are few relative to the definite hits, except for the “%I\_DIV4” procedure, which has zero definite hits, but shares a bucket with “%I\_MUL4”. In fact, there are no divide operations in the sample program; therefore all hits belong to the multiply operations.

If the percentage of definite hits becomes too low, you should consider reducing the requested bucket size.

---

---

## Implementation issues

The performance tools have been designed to work “as is” for most common application, desk accessory, and driver runtime environments. However, because Macintosh has an open architecture, it is possible that actions taken or assumptions made by application code will conflict with the needs of the performance tools. This section discusses possible conflicts, and how to resolve them.

---

### Locking the interrupt handler

You must lock down both code and data for the performance tools while performance measurements are being taken. Code for the trap handler must be locked down because the timer interrupts occur asynchronously. Data for the counters must be locked down because handles cannot be assumed to be valid during interrupt processing. The data area for counters cannot be “grown” at interrupt time, because the Memory Manager may not be in a consistent state.

---

## Segmentation

The code that must be locked down at execution time has been placed in segment "Main" and occupies about 1 kilobyte of space. This is because segment "Main" is usually guaranteed not to be unloaded at run time.

If your application's "Main" segment is too full to allow the performance tools to be linked correctly, then you may retarget the code in PerformLib.o by using the Lib tool. However, your application must not have an "unload all segments" routine in its idle procedure. One good segment to retarget to is "PerfMain", because this segment contains some of the other pieces of the performance tools.

These MPW commands illustrate how to retarget the code in PerformLib.o:

```
Duplicate {Libraries}PerformLib.o temp
Lib -o {Libraries}PerformLib.o -sn Main=PerfMain temp
Delete temp
```

The first command line creates a copy of PerformLib.o in temp. The second line replaces the original PerformLib.o with the output of Lib. The `-sn` option causes all code originally placed in segment "Main" to be in segment "PerfMain". The third line deletes the file temp.

---

## Dirty code segments

Because A5 is not valid at interrupt time, and there are no low memory globals assigned to performance measurement, the interrupt routine stores some data values in its code space, including the pointer to the locked-down data. Thus, if your application uses checksums to detect code segments attacked by errors, the performance tools will cause erroneous checksum failures. The easiest fix is simply not to checksum the "Main" segment (or whichever segment you choose).

---

## Movable code resources

The code for the trap handler, and the data area for the counters, must be locked down during performance measurement.

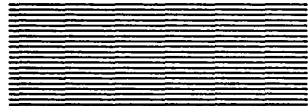
In counting "hits" in code resource segments, the performance interrupt routine checks that the handle to a measured resource is locked. If it is not locked, the resource is assumed to be "unloaded" and PC values are not checked for being within the resource.

---

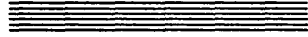
## Macintosh II ROMs and 24-bit addresses

The performance tools currently assume that the Macintosh II is running in 24-bit mode, with the upper 8 bits of PC-values in a "don't care" state. Currently, when doing performance measurements on the Macintosh II ROM, it appears that some code is running with 24-bit addresses and other code is running with 32-bit addresses. As a result, performance tools need to mask PC values to 24 bits to get consistent values. This situation may change in the future as the Macintosh Operating System more fully supports the 32-bit mode.





## **Chapter 13**



# **Writing an MPW Tool**

This chapter provides information specific to writing an integrated MPW tool. You'll also need to refer to the following:

- "Building an MPW Tool" in Chapter 9 for information about the mechanics of linking and installing a tool.
- The appropriate language reference manual for details of the Integrated Environment routines available in the various language libraries. (The **Integrated Environment** is a set of routines, modeled on the C language, that provide parameter passing, access to variables, and other functions to MPW tools.)

---

---

## Shell facilities

Tools running within the MPW Shell environment are provided with many facilities, including

- parameter passing
- access to Shell variables
- a set of preopened files for text-oriented input and output
- I/O to windows and selections
- a means for returning status results
- signal handling (for user aborts, and so on)
- exit processing

---

## Parameters

Parameters are passed to tools by the Shell. Every program is passed at least one parameter: the name of the program itself. This parameter is always the first parameter (technically, parameter 0) and is useful for error messages or other special action.

The text that follows the command name on the command line is first analyzed by the Shell for any special processing, such as filename generation or variable substitution. (See "How Commands Are Interpreted" in Chapter 5.) This text is then split up into individual words and placed in a convenient data structure for programmatic access.

- C In C, the main program is actually passed three parameters, named `argc`, the argument count; `argv`, the argument vector; and `envp`, the environmental pointer [described below under "Environment (Shell) Variables"]. The value of `argc` includes the command name (parameter 0), and is thus always one more than the number of parameters to the command. `argv` is a pointer to a zero-terminated array of pointers to the parameters, each of which is in C string (zero-terminated) format. (See Figure 13-1.)



Pascal

In Pascal, the parameters are accessible as the unit global variables argc and argv from the IntEnv (Integrated Environment) unit. As in C, the value of argc is one more than the parameter count; argv is a pointer to a null-terminated array of Pascal string pointers.

Assembly language

The Integrated Environment routine, \_RTInit, can be used to access the command parameters in assembly language. The addresses of argv and argc are passed to \_RTInit, which initializes them. See Appendix I of the *MPW Assembler 2.0 Reference* for details about this routine.

C and Pascal examples are shown in Figure 13-1.

C Sample.o -a Sample

Pascal Sample.p -a Sample

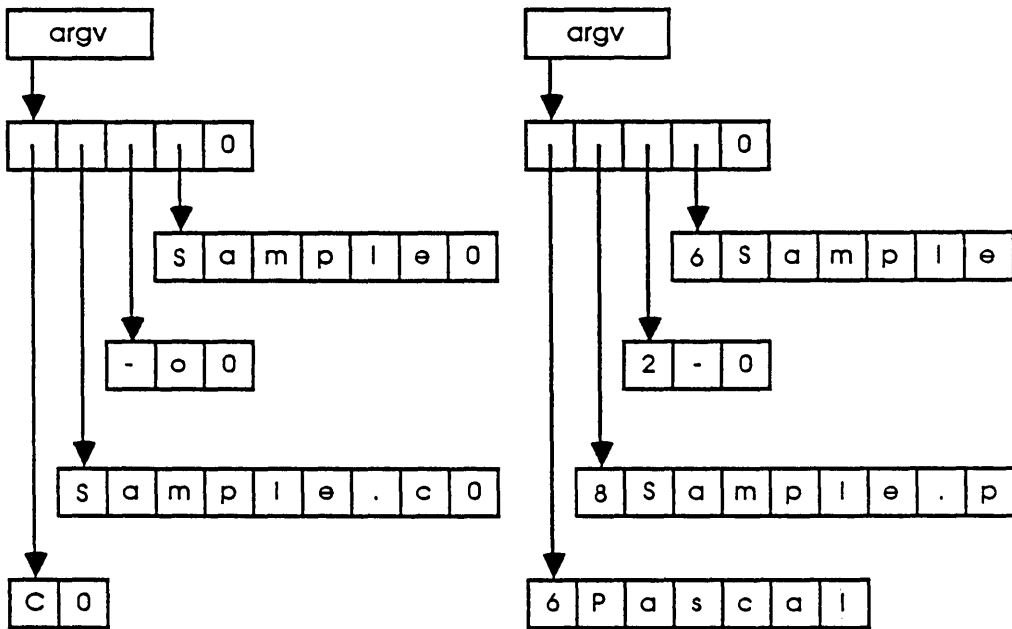


Figure 13-1  
Parameters in C and Pascal

---

## Environment (Shell) variables

The MPW Shell maintains a set of state variables that can be made available to tools with the `Export` command. When a tool is run, the Shell makes a copy of the names and string values of all exported variables and passes this list to the program. The tool can then determine the value of a variable by one of two methods:

- doing a linear search of the list of variables until the desired variable name is found
- using the `getenv` function

Because only a copy is passed, a tool cannot alter the Shell's value of a variable.

|                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| C                 | Shell variables are accessible in C via the third parameter to the main program, called <code>envp</code> (the environment pointer). <code>envp</code> is a pointer to a zero-terminated array of pointers to <i>name/value</i> C-string pairs. (Each pair is of the form <i>name 0value 0</i> .) The C library provides the <code>getenv</code> routine, which, given a variable name, looks up its value.                                                                                              |
| Pascal            | Pascal programmers are provided with another <code>IntEnv</code> unit global variable, also called <code>envp</code> . The structure used is the same as that for C, except that pointers to strings are forced to even byte boundaries by zero padding, if necessary. To facilitate the lookup of values for given Shell variables, a routine called <code>IEGetEnv</code> is provided in the <code>IntEnv</code> unit.                                                                                 |
| Assembly language | The <code>Integrated Environment</code> routine, <code>_RTInit</code> , can be used to access Shell variables in assembly language. The address of <code>envp</code> is passed to <code>_RTInit</code> , which initializes it. You can choose Pascal or C strings. You can use <code>getenv</code> for C strings, or <code>IEGetEnv</code> for Pascal strings (from the <code>PasLib</code> library). See Appendix I of the <i>MPW Assembler Reference</i> for details on calling <code>_RTInit</code> . |

---

## Standard I/O channels

Before starting a tool, the Shell sets up three text I/O channels that the tool can use to communicate with the outside world. These are

- standard input
- standard output
- diagnostic output (standard error)

By default, these channels are connected to the "console" (that is, any window on the screen). Program input may be typed (or selected) and entered in any window; program output appears immediately after the command in the same window. This input and output may be taken from or directed to other files by specifying I/O redirection (`<`, `>`, `>>`, `≥`, `≥≥`) on the command line. When the Shell encounters the I/O redirection notation, it opens or creates the necessary files, removes the redirection notation from the command line so that it doesn't appear in the program's parameter list, and then arranges for the open files to be passed to the program. When the tool finishes, the Shell flushes any buffered output and closes the files.

## I/O buffering

When using I/O routines provided by the language libraries, varying degrees of buffering are expected to occur on the standard I/O channels:

- Input from the console is buffered until the Enter key is pressed. If there is a selection when Enter is pressed, the selected text is used to satisfy the console read request; otherwise, the entire line that contains the insertion point is given to the reader.

*Note:* The MPW method of reading input creates a difficulty for interactive tools that write prompting text and pause to read a response entered on the same line: The tool will receive the prompt back as part of the line read, unless there was a selection when Enter was pressed.

- When input is taken from a file, the I/O package will, by default, read the data from the disk in 1K blocks.
- Text written to standard output is also buffered 1K at a time before being sent to a file or to the console. (As a convenience, when a read request is issued from the console, all output buffers are flushed so that any prompting text will appear before the program pauses waiting for input.)
- Text written to the diagnostic channel is buffered one line at a time, so that error messages and progress information appear in a timely manner while the program is executing.

Note that this buffering can cause apparently anomalous behavior: In particular, if both standard output and diagnostic output are directed to the console, the order of the output on the screen may not match the order in which the data was written. This change in order may result because the separate buffers are flushed at different times, as illustrated in Figure 13-2. You can circumvent this problem by flushing standard output before writing to diagnostic output.

❖ *Note:* Figure 13-2 shows the output conventions in C and Pascal. Assembly-language programmers must do their own buffering, or call C or Pascal routines.

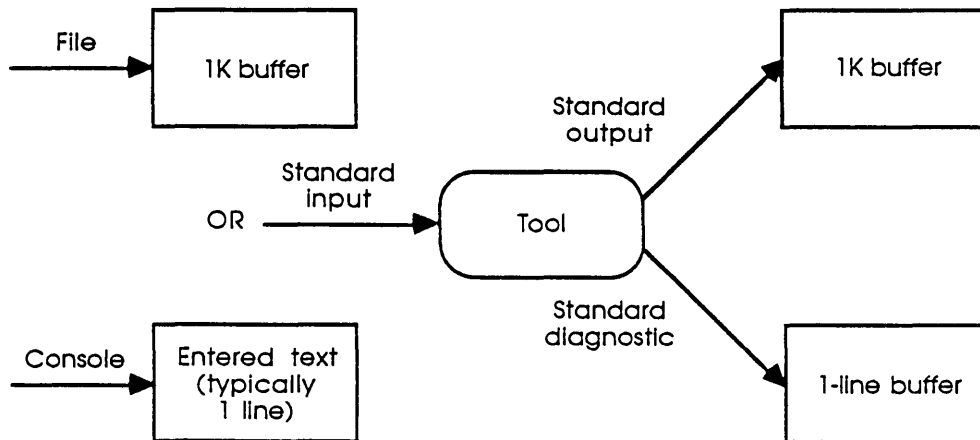


Figure 13-2  
I/O buffering

- C The standard I/O files are available for reading or writing in C, via the file descriptors 0, 1, and 2, or the StdIO stream descriptors `stdin`, `stdout`, `stderr`. These descriptors are fully documented in the *MPW C 2.0 Reference*.
- Pascal In Pascal, the program parameters *Input* and *Output* correspond to the standard input and output channels. A text file variable called `diagnostic`, which is connected to the standard diagnostic channel, is available from the IntEnv unit. The use of these parameters is documented in detail in the *MPW Pascal 2.0 Reference*.

## I/O to windows and selections

The MPW environment also provides to tools the ability to read and write to windows or to selections within windows. No special programming is required to use this feature. The MPW Shell monitors file system calls, and intercepts those that refer to a file that is currently open as a window. These calls are redirected automatically to the window rather than the file. (Thus, any modifications to the file do not become permanent until the window is saved.)

Accessing *selections* within windows is equally transparent to programs. All that is required is that the filename contain the selection suffix (`.$`). Reading from a selection is the same as reading from a file, and the beginning and end of the selection are treated as the bounds of the file. However, writing to a selection *replaces* the selection and has the interesting property that the data written is inserted into the file, rather than overwriting the data that follows.

Because window and selection I/O is handled automatically by the MPW Shell, tools should simply assume that they are always dealing with files.

---

## Signal handling

The MPW environment provides a set of routines to handle signals. A **signal** is an event that diverts program control from its normal execution sequence.

- ❖ *Note:* The only signal currently supported is Command-period, the standard Macintosh command for terminating the execution of an operation.

Signals can be caught, held and released, and ignored. The default action of any signal is to close all open files, execute any exit procedures (described below in "Exit Processing"), and terminate the program. If, however, your program requires special handling of a signal, or chooses to ignore it, you can use the procedure `sigset` to replace the default signal handling procedure with your own procedure. Your program can also temporarily suspend action on a signal (for instance, before entering a critical section of code) by calling `sighold`. You can restore the signal by calling the procedure `sigrelease`, whereupon the signal handling procedure will take affect if the signal was raised during the hold period. Your program may also pause until one or more signals are raised by calling the procedure `sigpause`. See the MPW language reference manuals for the details on how to use these routines.

---

## Exit processing

A program often requires some special processing before terminating. You can use the procedure `onexit` to register a procedure to be called at program termination or when the exit procedure is called. This procedure guarantees your program a chance to do any cleanup before terminating. Exit processing is especially useful for cleaning up after an uncaught signal.

---

## Status codes

Every tool is expected to return a status code to the Shell when it terminates. The Shell inspects this result—if the status code is nonzero and if the Shell variable `{Exit}` is nonzero (the default), the Shell terminates the execution of the current command file. The Shell also converts the result to string form and creates a Shell variable called `{Status}` with that value. The variable can then be tested with the Shell command language and action can be taken based on its value.

The following conventions are used for status codes:

- 0 Success
- 1 Command syntax error
- 2 Some error in processing
- 3 System error or insufficient resources

❖ *Note:* Only the bottom 24 bits of a tool's status code are returned to the Shell. The remaining codes (including negative numbers) are reserved for use by the Shell.

You may want to return error codes other than these. In that case, you should carefully document the numbers and their meanings.

|                   |                                                                                                                                                                                     |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| C                 | Result codes are returned from C tools as the function result of the procedure <code>main()</code> or by passing them as the parameter to the C Library <code>exit</code> function. |
| Pascal            | Pascal programmers must call the <code>IntEnv</code> procedure <code>IEexit</code> to return the status result.                                                                     |
| Assembly language | The Integrated Environment routine <code>_RTExit</code> is available to assembly-language programmers. <code>_RTExit</code> takes the status code as a parameter.                   |

---

---

## Restrictions

Tools are similar to desk accessories in that they coexist with another program (the MPW Shell), and many of the same restrictions apply to tools as to desk accessories. (See "Writing Your Own Desk Accessories" in the Desk Manager chapter of *Inside Macintosh*.) The following sections touch on some of the considerations in enabling tools to coexist with the Shell.

---

## Initialization

Because tools run with the Shell, most Macintosh Toolbox initialization calls are not necessary and should not be called. In particular, you should not make the following calls:

- InitFonts
- InitWindows
- InitMenus
- TEInit
- InitDialogs
- MaxApplZone
- SetApplLimit
- SetGrowZone
- InitResources
- RsrcZoneInit
- ExitToShell

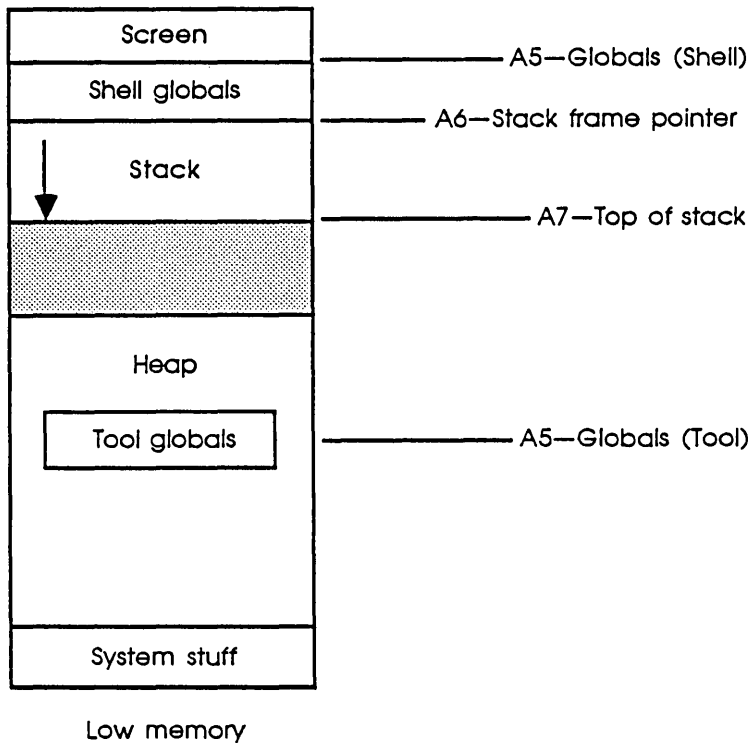
(Note that this is not an inclusive list.)

If your program uses QuickDraw or any routine that uses QuickDraw, be sure to call the InitGraf routine. This routine is necessary when using QuickDraw, because QuickDraw uses register A5-relative global variables, and tools have their own private A5 global area. Even a simple call to the QuickDraw function Random will not work properly unless InitGraf is called.

---

## Memory management

The Shell and tools execute out of the same heap and share the same stack. When a tool is started, the Shell allocates an area in the heap for the tool's globals and jump table, adjusts the global register A5 to point there, and then "calls" the tool. Any dynamic stack space required is allocated on the same stack, and any heap objects created go into the same heap.



**Figure 13-3**  
Memory map

When a tool terminates, the Shell restores the registers to their previous values and deallocates the tool's global area and any other pointers and handles in the heap that may have been left allocated. The tool's resources, however, are not deallocated immediately. They are unlocked and made purgeable so that the space can be used if needed. This practice allows for a quick restart of the tool if it is still in memory, but with no memory wastage penalty should the space be needed for other purposes.

---

### Warning

Although the Shell releases memory that has been allocated by the tool, sometimes the Shell has insufficient information to determine the owner of a master pointer. When a master pointer is NIL, it cannot be released by the Shell and cannot be reused.

NIL master pointers are produced as a result of calls to `EmptyHandle`, and by a number of Resource Manager actions. For example, a `GetResource` with `ResLoad` set to `FALSE` will create a NIL master pointer. If this is followed by a `DetachResource` or `RmveResource`, the handle remains as a NIL pointer.

It is always good programming practice to clean up handles after they have become obsolete. Use `DisposHandle` to get rid of such obsolete handles.

## Heap

Because the Shell and tools share the same heap, some cooperation is necessary to ensure efficient use of the heap. Before a tool is started, the Shell makes many of its heap objects unlocked and purgeable. The Shell's memory-resident code is kept as low in the heap as possible. The tool's code should be moved as high in the heap as possible. This is done automatically, if the locked bit is not set on the tool's code resources (the default from the Linker). When allocating heap space, tools should attempt to allocate no more space than is needed so that objects aren't needlessly purged from the heap.

When there is insufficient memory space to run a tool, you can make more space available in several ways:

- If you are using RAM caching, you can reduce the size of the cache.
- You can free up about 45K by running without the debugger (that is, name it something other than MacsBug and reboot, or hold down the mouse button while booting).
- You can minimize the number of windows open when the tool is run. (Certain memory-resident data structures are required for each window.) Directing program output to a file instead of a window will also provide the tool with more memory.
- You can also reduce the stack space by using the Shell resource described in the next section.

## Stack

When the Shell starts up, it immediately grows the heap to its maximum size based on the maximum stack size. The default maximum dynamic stack size is 10K bytes when less than 480K is available for the application heap; the default maximum dynamic stack size is 20K when more than 480K is available. Because some tools may require more stack space or more heap space, 'HEXA' resource number 128 is available in the Shell to adjust the maximum stack size.

- ❖ *Note:* Because the stack is shared between the Shell and the tool, executing tools from within nested command files results in less stack space for the tool. The Shell uses about 200 bytes of stack per nesting level.

---

---

## Windows, graphics, and events

The creation of windows, use of graphics, and event processing by tools is a largely unexplored area in the MPW environment. MPW aims to support these types of tools; however, little work has been done so far in this area, and unknown restrictions may exist.



---

---

## Conventions

MPW tools adhere to a certain style that allows them to work well together in an integrated fashion:

- Tools take their inputs as command-line parameters, rather than prompting for input. This input style allows their execution to be automated and allows them to take advantage of the Shell's command-line processing features such as variable substitution and filename generation.
- Deviations from a tool's standard behavior are specified with command options. Options may be specified anywhere on the command line and their order is not significant.
- Tools operate on a list of filename parameters, not just one, allowing the Shell's filename generation feature to be exploited.
- When no file parameters are given, tools take their input from standard input and write their output to standard output. The use of standard I/O allows the piping of the output of one program into the input of another. For example,

```
Files | Count -l
```

This command sends the output of the Files command into the input of the Count command, yielding the number of files and directories in the current directory.

- Most tools operate silently as they process their input. Visual feedback is provided by the spinning cursor. If more feedback is desired, a `-p` (progress) option is usually provided to send status and summary information to the diagnostic output.
- Error messages are in the form of Shell comments or are "executable" so that the error can be easily located. For example, the language translators report errors in the form

```
File "Test.c" ; line 25 ### expected: ';' got: name
```

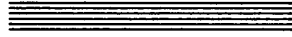
This message may be directly executed, to open the file and select the offending line. (See "Executable Error Messages" in Chapter 5.)

See the "Command Prototype" section at the beginning of Part II for more information on MPW command language conventions.





## **Chapter 14**



# **Creating a Commando Interface for Tools**

---

---

## About Commando

MPW 2.0 provides a dialog interface that makes it easier to use the MPW tools and scripts. A dialog is the programmed interaction between a user and a tool. A dialog box is the graphical vehicle used to display the various controls available for a tool or script. A dialog may employ several nested dialog boxes.

You implement dialogs for MPW tools by using Commando. This program looks in the resource fork of a tool or script for a resource of the type 'cmdo', that is, any dialogs to be used by the tool. Commando then loads the resource, builds a dialog list, handles events, and passes the command line back to the Shell for execution.

You can create a dialog interface for your own MPW tools and scripts. This chapter is a guide to creating the resources Commando requires to operate the dialog.

The MPW 2.0 Shell has several features that support Commando. The ellipsis character (three closely spaced periods obtained by pressing Option-semicolon) is reserved as a way to invoke Commando. When you enter an ellipsis at any point in a command line, the Shell will first carry out all alias and variable substitutions and then pass the entire command line to Commando. The output of Commando is then executed by the Shell.

Three new Shell variables are used by Commando:

- **Aliases:** This variable contains a list of all defined aliases, each name separated by a comma. The list contains only the names, not the definitions. Commando uses Aliases with the built-in command "alias". Without this variable Commando would have no way of knowing the names of the existing aliases. *The variable Aliases must be exported.*
- **Commando:** This variable tells the Shell which command to execute when the ellipsis character is present in a command line. To use the Commando tool, set the variable Commando to "Commando". You can use this variable to send the output of other tools to the Shell for execution. If the variable does not exist, then the ellipsis is removed from the command line and normal execution proceeds.
- **Windows:** This variable contains a list of the current windows, each name separated by a comma. Commando uses this list to redirect input or output to or from existing windows. Without this variable Commando would have no way of knowing the names of the current windows. *The variable Windows must be exported.*

Throughout this chapter, each type of Commando control is illustrated with an excerpt from Cmdo.r, found in the RIncludes folder.

---

## Using Commando

This procedure has proven to be the easiest:

1. Design your dialog boxes in ResEdit. It's a good idea to start with a template based on the examples shown later in this chapter.
2. DeRez the dialog box design and read the coordinates for your boxes and lines.
3. Use these coordinates when you create a Commando resource file for your tool or script. Create the help text in a separate pass, to make sure it will fit in the available space.
4. Add the Commando resource to your tool or script. For example,

```
Rez AddMenu.r -o "{MPW}MPW Shell" -a
Rez Rez.r -o {MPW}Tools:Rez -a
```
5. Adjust coordinates and repeat step 4.

---

---

## Resource description

Cmdo.r, the resource description file for Commando, is located in {RIncludes}cmdo.r.

---

### Resource ID and name

Any resource ID may be used for tools or scripts. Commando uses the first 'cmdo' resource it finds in the command's resource fork.

Commando draws an outlined button, the Do It button, in the lower-right corner of every dialog box. The Do It button is labeled with the name of the tool or script. (Normally Commando uses the name of the tool or script passed from the Shell.) Commando will capitalize the first character and force the rest of the characters to lowercase. For example, "StackSNiffer" would come out as "Stacksniffer."

Some people may prefer a different capitalization scheme. If you specify the 'cmdo' resource of a tool or script with a resource name, Commando will use that name "as is" as the label for the outlined button. This feature should be used sparingly; if you rename a tool, the previous name in the resource will still be displayed in the Do It button.

---

### Size of the dialog box

The width of Commando dialog boxes is fixed at 480 pixels. You are free to set the height to accommodate the controls in your tool's dialog box. The number specifying the height shouldn't exceed 295 to be compatible with the smaller Macintosh screens. Specifying this height in the 'cmdo' resource will result in the screen elements shown in Figure 14-1. See Table 14-1 for other recommended dimensions.

Please refer to the array declared under "type 'cmdo'" in the sample resource description file at the end of this chapter. The area labeled "Options" in Figure 14-1 is the area reserved for your controls and options.

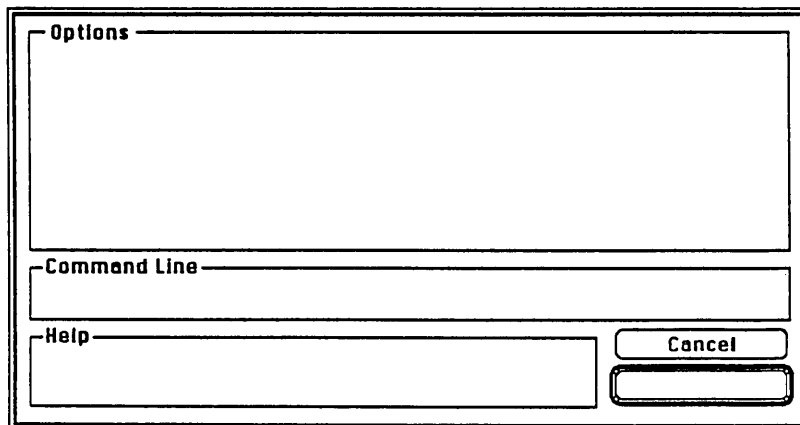


Figure 14-1  
Basic template for a Commando dialog box

The dimensions given below are not policy but recommendations. The sizes of the text-edit fields are important if you want to avoid text that shifts up and down slightly.

**Table 14-1**  
Summary of recommended sizes for Commando screen elements

---

|                        |                                                                                                                                             |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| Radio buttons          | 16 pixels high                                                                                                                              |
| Check boxes            | 16 pixels high                                                                                                                              |
| Pop-up menus           | 19 pixels high                                                                                                                              |
| Pop-up menu titles     | 16 pixels high. Top of title starts 1 pixel below top of pop-up menu (that is, top of title = top of pop-up menu + 1 pixel).                |
| Regular entries        | 16 pixels high                                                                                                                              |
| Multi-regular entries  | 16 pixels per line                                                                                                                          |
| Editable pop-up menus  | 22 pixels high                                                                                                                              |
| Editable pop-up titles | 16 pixels high. Top of title starts 3 pixels below top of editable pop-up menu (that is, top of title = top of editable pop-up + 3 pixels). |
| Icons                  | 32 pixels high; 32 pixels wide                                                                                                              |
| Pictures               | Same relative bounds as the rectangle stored in the PICT resource                                                                           |

---

---

## Tool description

At the bottom of the Commando dialog box is a three-line help box. The text in this box should be a brief, concise description of the tool, stating what it does. See the array declared under "type 'cmdo'" in the sample resource description file at the end of this chapter.

---

---

## Regular entry

The regular entry control is the most generic control available. The control behaves exactly like the text-edit fields in conventional Macintosh dialog boxes. In addition to strings and numbers, the regular entry control can be used for special options that have no specified standard control.

Here is the case declaration for regular entry controls:

```
case RegularEntry:
 key byte = RegularEntryID;
 cstring; /* title */
 align word;
 rect; /* bounds of title */
 rect; /* bounds of input box */
 cstring; /* default value */
 byte ignoreCase keepCase; /* the default value is never displayed in
 the Command window. If what the user
 types in the textedit window matches
 the default value, then that value
 isn't displayed. This flag tells
 Commando whether to ignore case when
 comparing the contents of the text edit
 window with the default value */
 cstring; /* option returned.*/
 cstring; /* help text for entry */
```

---

---

## Multiregular entry

Multiregular entry controls are similar to regular entry controls, except that multiregular entry controls accept values that can be entered more than once. For example, most compilers accept some type of **-define** option that can be specified more than once.

Here is the case declaration for multiregular entry. Note that the cstring for default values is the only control that passes its default values to the command line. This is an exception to the rule.

```
case MultiRegularEntry: /* for scrollable lists of an option */
 key byte = MultiRegularEntryID;
 cstring; /* title */
 align word;
 rect; /* bounds of title */
 rect; /* bounds of input list */
 byte = $$CountOf(DefEntryList);
 array DefEntryList {
 cstring; /* default values */
 };
 cstring; /* option returned. Each value will be preceded
 with this option.*/
 cstring; /* help text for entry */
```

Figure 14-2 shows the Defines window in the Rez dialog box with two defines entered. Here is the resource control for this function:

```
NotDependent (), MultiRegularEntry (
 "Defines:",
 {20, 35, 35, 125},
 {40, 30, 120, 225},
 {},
 "-d",
 "Type in multiple #defines here (such as LANGUAGE=French)"
),
```

The empty braces after the Defines window coordinates indicates that there are no default strings.

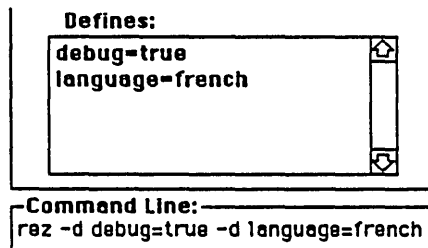


Figure 14-2  
Multiregular entry

---



---

## Check boxes

The check box control is likely to be the most often used because it corresponds to the on/off type options typical of MPW tools. Here is the case declaration for CheckOption:

```
case CheckOption:
 key byte = CheckOptionID;
 byte NotSet, Set; /* whether button is set or not */
 rect; /* bounds */
 cstring; /* title */
 cstring; /* option returned */
 cstring; /* help text for button */
```

The byte NotSet or Set is used to set the button's default state. The option is returned only when the button is not its default state.

This resource code produces the check boxes in Figure 14-3:

```
notDependent (), CheckOption (
 NotSet, {20, 10, 36, 235}, "Show macro expansions",
 "-print GEN",
 "Expand macros in the listing file."),
notDependent (), CheckOption (
 Set, {35, 10, 51, 235}, "Allow automatic page ejects",
 "-print NOPAGE",
 "Controls whether the Assembler sends automatic page ejects
 to the listing file"),
```



```

notDependent { }, CheckOption {
 Set, {50, 10, 66, 235}, "Show warning messages",
 "-print NOWARN",
 "Controls both the display and count of warning messages." },
notDependent { }, CheckOption {
 Set, {65, 10, 81, 235}, "Show macro call statements",
 "-print NOMCALL",
 "Controls the listing of macro call statements." },
notDependent { }, CheckOption {
 Set, {80, 10, 96, 235}, "Show generated object code",
 "-print NOOBJ",
 "List generated object code or data for each listed line." },
notDependent { }, CheckOption {
 NotSet, {95, 10, 111, 235}, "Show up to 255 bytes of data",
 "-print DATA",
 "Controls whether object data is shown in full
 (up to 18 lines) or limited to one line." },
notDependent { }, CheckOption {
 Set, {110, 10, 126, 235}, "Show macro directive lines",
 "-print NOMDIR",
 "Controls whether macro directives (including conditional
 and set directives) are shown in the listing." },
notDependent { }, CheckOption {
 Set, {125, 10, 141, 235}, "Show header lines", "-print NOHDR",
 "Controls whether header lines are printed in the listing." },
notDependent { }, CheckOption {
 Set, {140, 10, 156, 235}, "Show generated literals",
 "-print NOLITS",
 "Controls listing of literals produce by PEA and LEA machine
 instructions." },
notDependent { }, CheckOption {
 NotSet, {155, 10, 171, 235}, "Show assembly status",
 "-print STAT",
 "Controls display of assembly status in the listing." },

```

Figure 14-3 shows a set of check boxes in their default state and again after the two top buttons have been clicked.

Default state of buttons

- Show macro expansions
- Allow automatic page ejects
- Show warning messages
- Show macro call statements
- Show generated object code
- Show up to 255 bytes of data
- Show macro directive lines
- Show header lines
- Show generated literals
- Show assembly status

Command Line: \_\_\_\_\_

asm

State after top two buttons clicked

- Show macro expansions
- Allow automatic page ejects
- Show warning messages
- Show macro call statements
- Show generated object code
- Show up to 255 bytes of data
- Show macro directive lines
- Show header lines
- Show generated literals
- Show assembly status

Command Line: \_\_\_\_\_

asm -print GEN -print NOPAGE

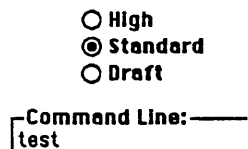
**Figure 14-3**  
Setting the CheckOption default state

---

---

## Radio buttons

The simplest set of radio buttons offers several mutually exclusive options. For example, the Print Option radio buttons in Figure 14-4 let you choose High or Standard or Draft. The Standard mode is the default.



**Figure 14-4**  
Radio buttons with default setting

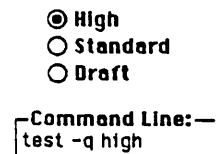
Here is the case declaration for radio buttons:

```
case RadioButtons:
 key byte = RadioButtonsID;
 byte = $$CountOf (radioArray); /* # of buttons */
 wide array radioArray (
 rect; /* bounds */
 cstring; /* title */
 cstring; /* option returned */
 byte NotSet, Set; /* whether button is set or not */
 cstring; /* help text for button */
 align word;
);
```

To make the middle radio button the default, as shown in Figure 14-4 above, declare the middle Standard button set:

```
notDependent, RadioButtons (
 (
 (115, 300, 130, 400), "High", "-q high", notset,
 "Print the selected files in the highest quality
 available from the printer.",
 (132, 300, 147, 400), "Standard", "-q standard", set,
 "Print the selected files in the normal quality mode.",
 (149, 300, 164, 400), "Draft", "-q draft", notset,
 "Print the selected files in the fastest way possible
 at the expense of quality."
```

No option is passed to the command line box because the middle button is explicitly declared the default. If a button other than the default is clicked, Commando passes the appropriate option to the command line, as shown in Figure 14-5.



**Figure 14-5**  
Clicking a button other than the default

Suppose that in the previous example you wanted the default radio button to display its option in the command line. You would simply change the order in which you declared the radio buttons, so that the middle button would be declared first. Be sure that all buttons are NotSet. The result is shown in Figure 14-6. Commando will set the first button that it encounters if no button is specified as set.

```
○ High
◎ Standard
○ Draft

└─Command Line:─
 test -q standard
```

**Figure 14-6**  
No button specified as set

- ❖ *Note:* A radio button can be either dependent upon or parent to another control. For purposes of establishing dependency relations, a cluster of radio buttons is considered a single item in the resource listing. See “Control Dependencies” in this chapter for more information.

---

---

## Boxes, lines, and text titles

It is recommended that you group dialog controls or functions within boxes. Commando supplies the facilities to draw a box (case Box), to draw a box with a title embedded in the upper-left corner (case TextBox), and to create titles in any font (case TextTitle).

- ❖ *Note:* When you draw a box around a set of controls, always list the box declaration after listing the other controls. Otherwise the Dialog Manager might confuse which control is clicked.

Box and TextBox cannot depend on other controls, nor can other controls depend on them. Commando would not complain if you set up such a dependency, but the line or box would not respond to the state of the determining item. TextTitles, on the other hand, can be dependent on another control.

---

### Box

Use the case Box to draw boxes around controls or to draw lines. To draw lines, make the rect one pixel wide. In other words, to draw a horizontal line, you might set the rect to {10, 10, 11, 100}. Here is the case declaration for the Box control:

```
case Box: /* Can be used to draw lines too */
 key byte = BoxID;
 byte black, gray; /* color of object */
 rect; /* bounds of box or line */
```

---

## TextBox

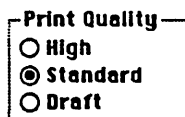
The case `TextBox` lets you draw a box with the title embedded in the line at the upper-left corner. This is a frequently used convention in the Commando dialogs. (See the sample dialog box template in Figure 14-1.) Here is the case `TextBox` declaration from the Command resource file:

```
case TextBox: /* Draws a box with title in upper-left */
 key byte = TextBoxID;
 byte black, gray; /* color of object */
 rect; /* bounds of box or line */
 cstring; /* title */
```

For example,

```
notDependent, TextBox {
 gray,
 {105, 295, 169, 405},
 "Print Quality"
},
```

This declaration gives you the results shown in Figure 14-7.



**Figure 14-7**  
TextBox example

---

## TextTitle

Use `TextTitle` to draw text in any font. Here is the case declaration:

```
case TextTitle:
 key byte = TextTitleID;
 byte plain; /* style of text */
 rect; /* bounds of title */
 int systemFont; /* font number to use */
 int systemSize; /* font size to use */
 cstring; /* the text to display */
```

For example, let's write "so cool" in a cool way:

```
notDependent, TextTitle {
 bold + italic, {20,20,40,100},
 systemFont, 12, "So Cool..."
},
```

***So Cool...***

---

---

## Pop-up menus

Pop-up menus are a convenient way to select an item from a list of related items. Commando manages the associated windows, aliases, fonts, and Shell variables. Here is the case declaration:

```
case PopUp:
 key byte = PopUpID;
 byte Window, Alias, Font, Set; /* popup type */
 rect; /* bounds of title */
 rect; /* bounds of popup line */
 cstring; /* title */
 cstring; /* option returned */
 cstring; /* help text for popup */
 byte noDefault, hasDefault; /* hasDefault if 1st item
 is "Default Value" */
```

The last field, "byte noDefault, hasDefault", tells Commando whether the pop-up menu has a default value or not. If the pop-up menu does not have a default value, the first value in the pop-up list is automatically selected and passed to the command line. If the pop-up menu does have a default value, then Commando adds a new item that says something like "Default Value" to the front of the list. When this value is selected, no value is displayed in the window that generates the pop-up menu.

Here is an example of the resource code for a pop-up menu with a default value. See Figure 14-8 for the resulting window and pop-up menu.

```
notDependent, PopUp {
 Font,
 {21, 20, 37, 60},
 {20, 60, 39, 160},
 "Font",
 "-f",
 "popup help message",
 hasDefault
},
```

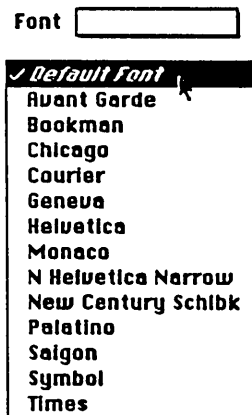


Figure 14-8  
Pop-up menu with default value

Here is the resource code for a pop-up menu with no default value. The results are shown in Figure 14-9.

```
notDependent, PopUp {
 Font,
 {46, 20, 62, 60},
 {45, 60, 64, 160},
 "Font",
 "-f",
 "popup help message",
 noDefault
},
```

Font **Avant Garde**

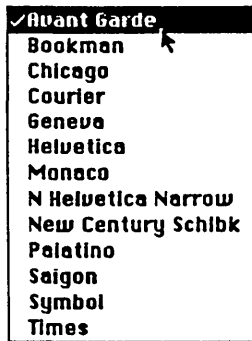


Figure 14-9  
Pop-up menu without default value

---

## Editable pop-up menus

Pop-up menus associated with a text-edit box can be edited. You can choose existing values from a list and still have the flexibility to enter completely new values.

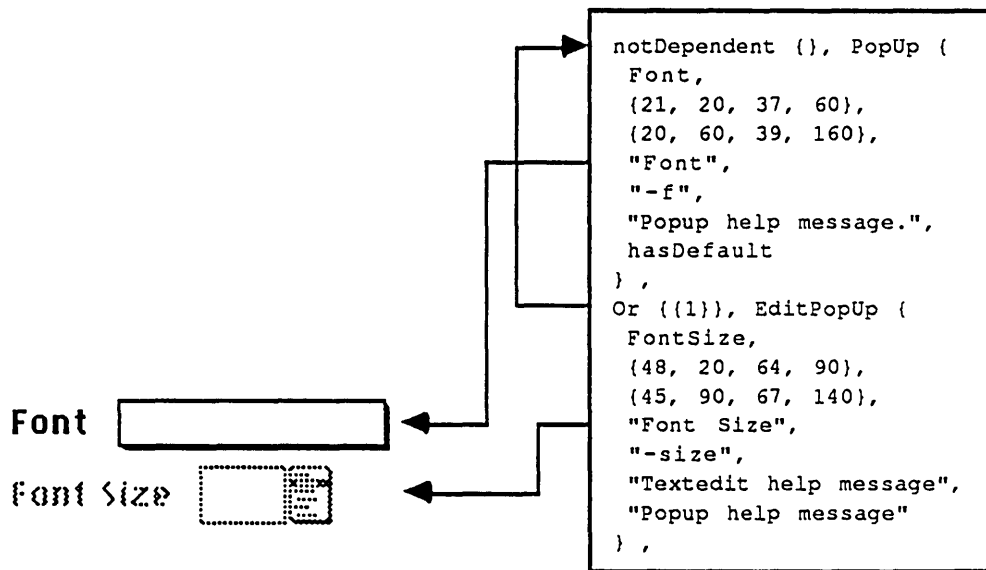
```
case EditPopUp:
 key byte = EditPopUpID;
 /* For MenuItem, this EditPopUp must be dependent on
 another EditPopUp of the MenuItem type so that
 the control recognizes which menu item to display.

 For FontSize, this EditPopUp must be dependent on
 a PopUp of the Font type so that the control
 recognizes which sizes of the font exist. */

 byte MenuItem, MenuItem /* Type of editable pop-up */
 FontSize, Alias, Set;

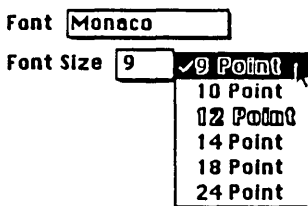
 rect; /* bounds of title */
 rect; /* bounds of text edit area */
 cstring; /* title */
 cstring; /* option to return */
 cstring; /* help text for text-edit */
 cstring; /* help text for pop-up */
```

The example in Figure 14-10 shows how the Font Size editable pop-up menu is made dependent on the current font.



**Figure 14-10**  
How Font Size dependency is handled

If a particular font is selected in the Font box, then the font sizes that actually exist are outlined. In the example in Figure 14-11 the Monaco font has been selected in the Font box. The 9-point item is outlined and has been selected with the mouse. Any font size could be typed in the Font Size box.



**Figure 14-11**  
Font Size pop-up menu with font selected

---

---

## Lists

Use the case List to enable users to make multiple selections from a list of items. Four types of things can be listed:

- volumes (Inserted disks will be recognized and added to the list.)
- shell variables
- windows
- aliases

Here is the case declaration for List:

```
case List: /* Puts up list of items & allows multiple
selections */
 key byte = ListID;
 byte Volumes, ShellVars, /* what to display in
 Windows, Aliases; the list */
 cstring; /* option to return before each item */
 cstring; /* title */
 align word;
 rect; /* bounds of title */
 rect; /* bounds of list selection box */
 cstring; /* help text for selection box */
```

Here is the resource code for the two examples shown in Figure 14-12. The second example shows that the user has already selected a window.

```
notDependent, List {
 Volumes,
 "",
 "Volumes",
 {20,30,35,120},
 {37,30,101,200},
 "Help message"
},
notDependent, List {
 Windows,
 "-w",
 "Window List",
 {20,220,35,3030},
 {37,220,101,400},
 "Help message"
},
```

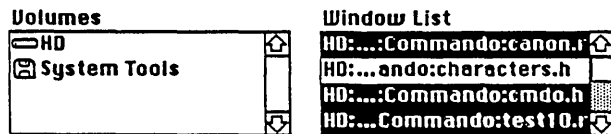


Figure 14-12  
List control



---



---

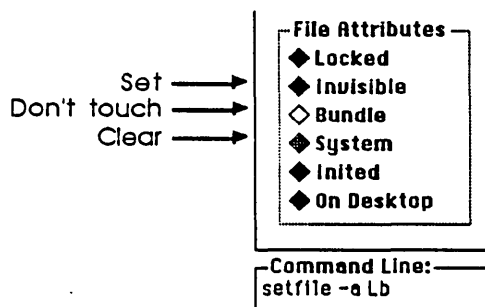
## Three-state buttons

Three-state buttons were invented to handle the SetFile and SetPrivilege commands. Both of these commands deal with the setting or clearing of flags. These commands also have another implicit state: "Don't touch." Therefore, these buttons have three states: Set, Clear, and Don't touch.

```
case TriStateButtons:
 key byte = TriStateButtonsID;
 byte = $$CountOf (threeStateArray); /* # of buttons */
 cstring; /* option returned */
 wide array threeStateArray (
 align word;
 rect; /* bounds */
 cstring; /* title */
 cstring; /* for Clear state */
 cstring; /* for DontTouch state */
 cstring; /* for Set state */
 cstring; /* help text for button */
);
```

Here is the resource code for the example shown in Figure 14-13.

```
notDependent, TriStateButtons (
 "-a",
 {
 {40, 25, 58, 135}, "Locked", "l", "", "L",
 "This button affects the file \"Locked\" attribute.",
 {58, 25, 76, 135}, "Invisible", "v", "", "V",
 "This button affects the file \"Invisible\" attribute.",
 {76, 25, 94, 135}, "Bundle", "b", "", "B",
 "This button affects the file \"Bundle\" attribute.",
 {94, 25, 112, 135}, "System", "s", "", "S",
 "This button affects the file \"System\" attribute.",
 {112, 25, 130, 135}, "Inited", "i", "", "I",
 "This button affects the file \"Inited\" attribute.",
 {130, 25, 148, 135}, "On Desktop", "d", "", "D",
 "This button affects the file \"On Desktop\" attribute.",
```



**Figure 14-13**  
Three-state buttons

---

---

## Icons and pictures

Use the case `PictOrIcon` to place icons, pictures, or both in the Commando windows. This item cannot be dependent on any other item, nor other items on it. Here is the case declaration for icons and pictures:

```
case PictOrIcon:
 key byte = PictOrIconID;
 byte Icon, Picture; /* display a picture or icon */
 int; /* resource ID of icon */
 rect; /* display bounds */
```

The icon in Figure 14-14 is produced by an `ICON` resource with an ID of 0, located in the system file.



**Figure 14-14**  
Icon in a Commando window

Here is the resource code that generates the example shown in Figure 14-14:

```
notDependent, PictOrIcon {
 Icon, 0, {20, 20, 52, 52}
},
```

---

---

## Control dependencies

Sometimes one control is dependent upon the value of another control. For example, a font size control might be dependent on a preceding font selection control. In this case the font size control is termed the **dependent**. The preceding font selection control is called the **parent** because it enables or disables the dependent.

Commando numbers each item sequentially in the order of its appearance in the resource description file. These numbers do not appear in the resource code; you must count them manually. The dependent/parent relationship in Commando is controlled by the sequential order of items entered into a Commando resource.

An item may be dependent only on other items within the same dialog box. In the case of nested dialog boxes, the items in the second dialog box must be renumbered starting from one.

---

## Direct dependency

Usually dependency on another control means that the dependent control is disabled if the parent control is disabled (or has no value).

Figure 14-15 shows two states of a directly dependent control. In the first case, nothing has been entered in the Type field, so the dependent Creator field is disabled and appears dimmed in the dialog box. In the second case, the Creator field is enabled as soon as something is typed in the Type field.

Figure 14-15 also illustrates how the ignoreCase/keepCase flag works. Because the flag is keepCase and 'appl' is not equal to 'APPL', the option is displayed in the Command Line box.

|         |                                   |
|---------|-----------------------------------|
| Type    | <input type="text"/>              |
| Creator | <input type="text" value="????"/> |

---

Command Line:  
test

|         |                                   |
|---------|-----------------------------------|
| Type    | <input type="text" value="appl"/> |
| Creator | <input type="text" value="????"/> |

---

Command Line:  
test -t appl

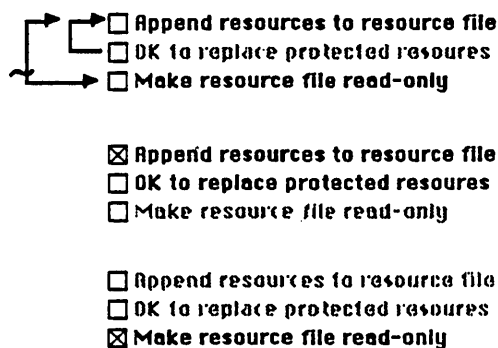
**Figure 14-15**  
Direct dependency

---

## Inverse dependency

A control can be inversely dependent on another control. In other words, if the parent is disabled, then the dependent is enabled. Or, if the parent is enabled, then the dependent is disabled.

It is also possible for two controls to be inversely dependent on each other. This means that both controls are enabled until one is selected; then the other is disabled. For example, there are two types of dependencies illustrated in Figure 14-16. The user can select either the top check box or the bottom one, but not both; that is, the user is allowed to append resources to a resource file *or* to make the resource map read-only. The middle box is enabled only when the top box is checked, because it makes sense to replace protected resources only when appending to a source file.



**Figure 14-16**  
Inverse dependencies

Here is the resource description of the three check boxes shown in Figure 14-16. To make a control inversely dependent on another control, make the value of the parent negative.

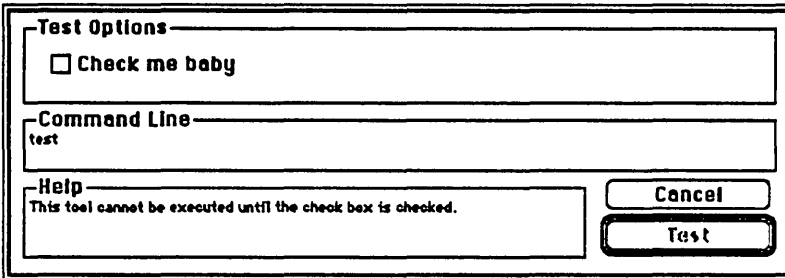
```
Or { {-3} }, CheckOption {
 NotSet,
 {20, 10, 40, 350},
 "Append resources to resource file",
 "-a",
 "some help text..."
},
Or { {1} }, CheckOption {
 NotSet,
 {40, 10, 60, 350},
 "OK to replace protected resources",
 "-ov",
 "some help text..."
},
Or { {-1} }, CheckOption {
 NotSet,
 {60, 10, 80, 350},
 "Make resources file read-only",
 "-ro",
 "some help text..."
},
```

The second CheckOption (the dependent) is enabled only if the first (the parent) is enabled. The third CheckOption is enabled only if the first is disabled.

---

## Dependency on the Do It button

To make the Do It button dependent on something, you must use the special DoIt Button item in the Commando resource type definition. This item can be specified only once per resource and can be specified only in the first dialog. In the example shown in Figure 14-17 the Do It button is dependent on the check box.



**Figure 14-17**  
Dependency on the Do It button

Here is the resource code for the above example:

```
NotDependent { }, CheckOption {
 NotSet,
 {20, 20, 40, 200},
 "Check me baby",
 "-c",
 "Help us to help you.",
},
Or { {1} }, DoItButton {
}
```

---

## Multiple dependencies

A Commando item can be dependent on one or more other items. For example, a control might be enabled only when two other controls are enabled. Such situations are considered multiple dependencies.

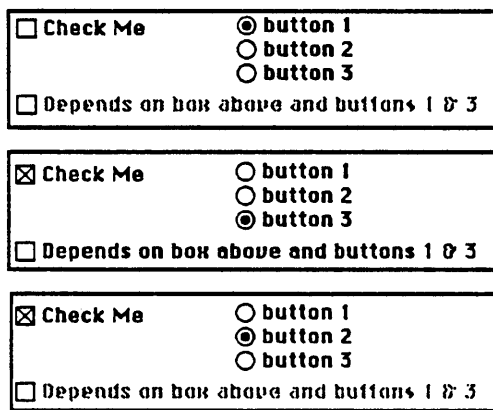
Multiple dependencies may be of two types: OR and AND. In an OR dependency, a dependent control is enabled if any of its parents is enabled. In an AND dependency, the dependent control is enabled only if all its parents are enabled. It is possible to mix ANDs and ORs. Include an item within an AND or OR list that is dependent on a dummy control (case Dummy)—and make the dummy control dependent on another list of controls. An example appears in the next section.

## Dependencies on radio buttons

Commando considers a cluster of radio buttons to be one item. Remember that Commando numbers each item sequentially in the order of its appearance in the resource description file. When an item is dependent on a specific radio button within a cluster of radio buttons, the number of the individual button is placed in the upper four bits of the item number that describes the entire cluster of radio buttons. For example, consider a radio button cluster that is item 5 and contains six radio buttons. To have a dependency on button #3 you would write, in Rez syntax,

```
(3<<12) + 5
```

Figure 14-18 shows three ways in which the check box at the bottom of the dialog box is dependent on the upper check box and radio buttons.



**Figure 14-18**  
Dependencies on radio buttons

Here is the resource description code describing the operation of the dialog box in Figure 14-18:

```
notDependent { }, CheckOption {
 NotSet, {15, 15, 31, 100}, "Check Me", "-root", ""
},
And ({1, 3}), CheckOption {
 NotSet, {65, 15, 81, 450}, "Depends on box above and
 buttons 1 & 3", "-above1", ""
},
Or ({ (1 << 12) + 4, (3 << 12) + 4 }), Dummy {
},
notDependent { }, RadioButtons ({
 {15, 150, 31, 450}, "button 1", "-b1", NotSet, "no help",
 {30, 150, 46, 450}, "button 2", "-b2", NotSet, "no help",
 {45, 150, 61, 450}, "button 3", "-b3", NotSet, "no help",
}) },
```

In Figure 14-18 the first CheckOption is Item #1 in the resource description file and the next CheckOption is Item #2 in the same file. The third item is a dummy item used to perform the complex dependency. Item #4 is the entire cluster of three radio buttons. Item #2 (the bottom check box in the sample dialog) is dependent on Item #1 (the top check box) AND radio button #1 OR radio button #3.

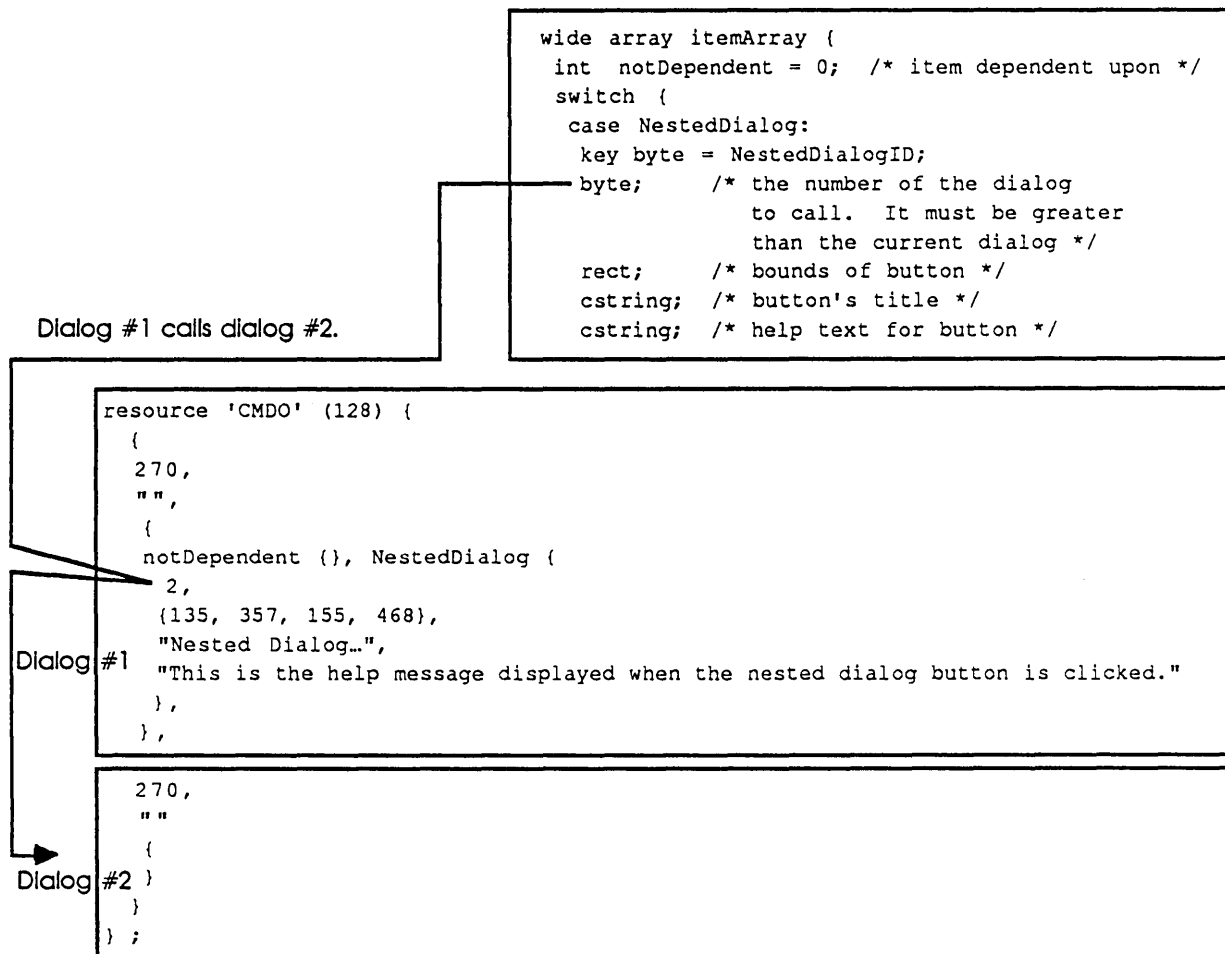
---

---

## Nested dialog boxes

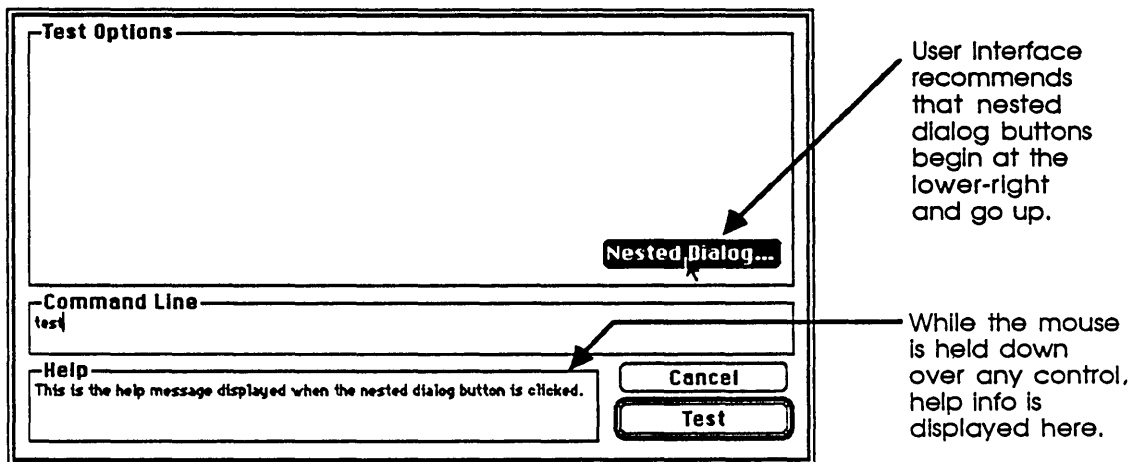
Complex tools may require more than one dialog box in order to display all the options. When there are several nested dialog boxes, all of them are called from buttons in the first dialog box. It's best to avoid calling nested dialog boxes from other nested dialog boxes.

Figure 14-19 shows how dialog box #2 can be called from dialog box #1.



**Figure 14-19**  
Setting up nested dialog boxes

Figure 14-20 shows the recommended placement of nested dialog call buttons.



**Figure 14-20**  
Placement of nested dialog buttons

Clicking the Cancel button in a nested dialog box reverts all its controls to their state before the nested dialog box was opened, returning the user to dialog box #1. Clicking the Do It button (typically labeled "Continue") saves the current state of all controls in the nested dialog box, returning the user to the first dialog box.

---

---

## Redirection

Redirection is the easiest control to add to a Commando resource description file. Simply specify the type of redirection and the point location of the upper-left corner. Commando takes care of the rest. Here is the case declaration for redirection:

```
case Redirection
 key byte = RedirectionID;
 byte StandardOutput, /* the type of redirection */
 DiagnosticOutput,
 StandardInput;
 point; /* upper-left point of the entire contraption */
```



Figure 14-21 shows the resource code for redirection along with its results.

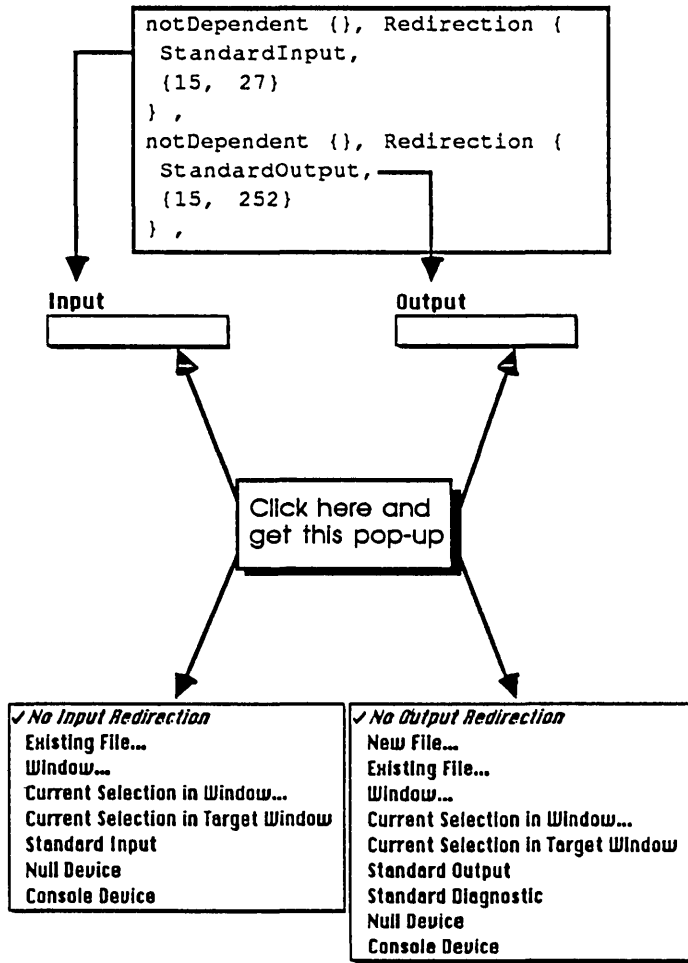


Figure 14-21  
How to obtain input and output redirection

---



---

## Files and directories

There are four types of Commando dialogs, offering four different ways to make files and directories available:

- as individual items for both input and output
- as multiple files for input only
- as multiple files and directories for input only
- as multiple new files for output

---

### Individual files and directories

The Files control enables users to select a single file or directory that can be used for input or output. This control supports seven combinations of files, as illustrated in Figure 14-22.

```

case Files:
key byte = FilesID:
byte InputFile, /* These types require the */
 InputFileOrDir, /* 'Additional' case below. */
 InputOrOutputFile,
 InputOrOutputOrDir,
 OutputFile, /* These types require the */
 OutputFileOrDir, /* 'NoMore' case below. */
 DirOnly:
switch (
case OptionalFile:
/* Using this case makes a popup with two or three items.
This first item is used to select a default file or to
select no file. The second (and third) item(s) are used
to select input or output files. */
key int = 0:
rect: /* bounds of title */
rect: /* bounds of display box */
cstring: /* title */
cstring: /* default file */
cstring: /* option to return before file */
cstring: /* If this item is dependent upon
another Files item, an extension
can be specified here to be added
to the dependents file. */
cstring: /* help text for popup */
byte dim, dontDim: /* Normally, dependent items are
dimmed if the parent is disabled.
if this field is "dontDim", then
this item won't be dimmed */
/* These next three strings are the strings displayed in the popup. Most
of the file types have only two strings but InputOrOutputFile and
InputOrOutputOrDir require three strings. If a string is set to "",
Commando will use a built-in default. */
cstring: /* popup item #1 */
cstring: /* popup item #2 */
cstring: /* popup item #3 */

case RequiredFile:
/* Using this case makes a button that goes directly to standard
file. Use this case when a file is required and the user doesn't
have the choice of a default file or no file. */
key int = 1:
rect: /* bounds of button */
cstring: /* title of button */
cstring: /* option to return before file */
cstring: /* help text for button */
};

/* Some file types take additional information. See the comment next to the
file types to find out which case to choose here. */
switch (
case Additional:
key byte = 0:
cstring: /* For InputOrOutputFile, an option
can be specified when a new
(or output) file is chosen. */
cstring FilterTypes = ".*"; /* preferred file extension (i.e. ".c")
If null, no radio buttons will be
displayed. If FilterTypes is used,
the radio buttons will toggle
between showing files only with
the types below and all files */
cstring: /* title of only files button */
cstring: /* title of all files button */
byte = $$CountOf(TypesArray); /* up to 4 types may be specified */
align word:
array TypesArray (
literal longint text = 'TEXT', /* desired input file type, specify */
obj = 'OBJ', /* no type for all types */
appl = 'APPL',
da = 'DFIL',
tool = 'MPST';
);

case NoMore:
key byte = 1:
};

```

Each group of files uses its own case

This case generates a popup.

This case generates a button.

Figure 14-22  
Resource description for "individual files and directories" controls

Here is the resource code for the "individual files and directories" controls that appear in Figure 14-23.

```
notDependent {} , Files {
 InputFile,
 OptionalFile (
 {20,20,40,130},
 {20,100,40,300},
 "C Input Files",
 "", "", "",
 "Help message here.",
 dim,
 "Read Standard Input",
 "Select a file to compile...",
 "",
),
 Additional (
 "",
 ".c",
 "Only files that end in .c",
 "All text files",
 {text}
),
},
},
Or {{1}}, Files {
 OutputFile,
 OptionalFile (
 {50,20,70,100},
 {50,100,70,300},
 "Object File",
 "c.o", "-o", ".o",
 "Help message here.",
 dontDim,
 "Send object code to c.o",
 "Select an object file...",
 "",
),
 NoMore {},
},
},
```

Figure 14-23 shows the control resulting from the resource code above. The control is shown first in its default state, then as the user selects an input file, and finally as Commando produces the object file associated with the input file selected by the user.

Default state

C Input File

Object File

Choose an input file

C Input File  Read Standard Input  
 Select a file to compile...

Object File

Object file dependent on Input

C Input File

Object File

**Figure 14-23**  
Examples of "Individual files and directories" controls

---

## Multiple files and directories for input and output

Use the case MultiFiles (shown here) to enable users to select multiple files and directories for input and output. Note the four cases representing subtypes within the case MultiFiles:

- case MultiInputFiles
- case MultiDirs
- case MultiInputFilesAndDirs
- case MultiOutputFiles

Here is the MultiFiles case:

```
case MultiFiles:
 key byte = MultiFilesID;
 cstring; /* button title */
 cstring; /* help text for button */
 align word;
 rect; /* bounds of button */
 cstring; /* message like "Source files
 to compile:" */ switch {

case MultiInputFiles:
 key byte = 0;
 byte = $$CountOf (MultiTypesArray); /* up to 4 types may be
 specified */

 align word;
 array MultiTypesArray {
 literal longinit text = 'TEXT', /* desired input file
 type, specify no type */
 /* for all types */
 obj = 'OBJ',
 appl = 'APPL',
 da = 'DFIL',
 tool = 'MPST';
 };
 cstring FilterTypes = ":"; /* preferred file extension
 (that is, ".c"). If null,
 no radio buttons will be
 displayed. If FilterTypes
 is used, the radio buttons
 will toggle between show-
 ing files with only the types
 below, and all files. */

 cstring; /* title of only files button */
 cstring; /* title of all files button */

case MultiDirs:
 key byte = 1;

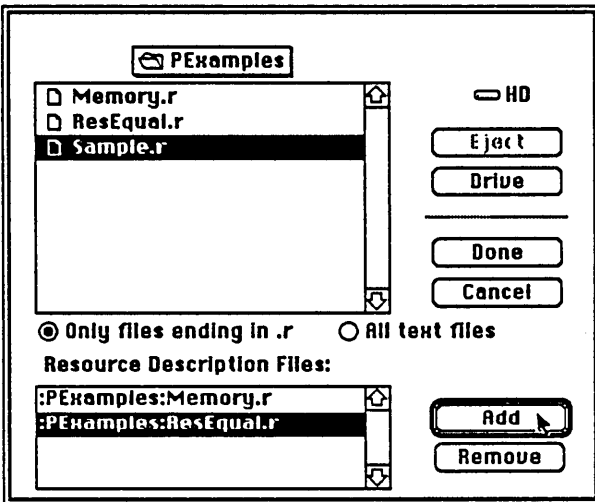
case MultiInputFilesAndDirs:
 key byte = 2;

case MultiOutputFiles:
 key byte = 3;
};
```

Figure 14-24 is a Files dialog box controlled by resource code using the MultiFiles case. Here is the resource code:

```
notDependent {}, MultiFiles {
 "Description Files...",
 "Select resource input files to compile",
 {60, 330, 80, 468},
 "Resource Description Files:", "" ,
 MultiInputFiles {
 {text}.
 ".r",
 "Files ending in .r",
 "All text files"
 },
},
```

"Description Files..." is the button in the Rez dialog box that displays the large standard file dialog box shown in Figure 14-24. The last two titles refer to the two radio buttons.



**Figure 14-24**  
Example of multiple input files

In the above example two resource files have just been added.

---

## Multiple files and directories for input only

Here's how you can use the case MultiFiles to enable the user to select multiple directories for input only.

```
NotDependent {}, MultiFiles (
 "Include Paths...",
 "Help message for directory button.",
 (110, 330, 130, 468),
 "Include Search Paths:",
 "-s",
 MultiDirs {},
),
```

The first item in the above code, "Include Paths...", is the button in a frontmost dialog box (in this example Rez was used) that generates the file dialog box shown in Figure 14-25. "Include Search Paths:" is the title of the scrollable window at the bottom of the dialog box. Two Includes folders have just been selected from the upper window and added to the Include Search Paths window just below.

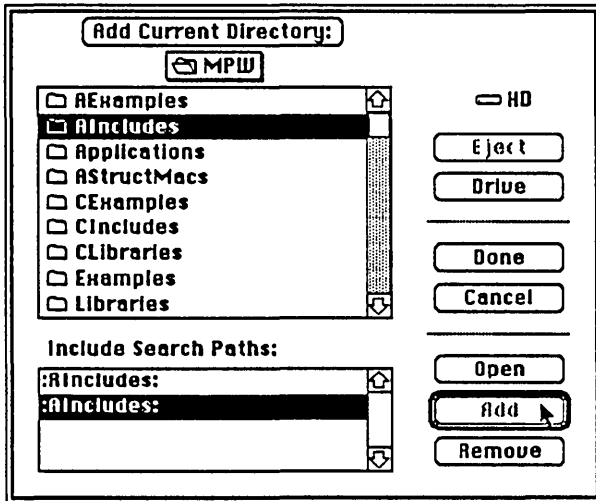


Figure 14-25  
Multiple directories for input

Another file dialog box is used to select multiple files and directories. This dialog box appears in Figure 14-26. Here is the resource code that produces this dialog box.

```

NotDependent {}, MultiFiles (
 "Files to duplicate...",
 "This button brings up a dialog allowing"
 "selection of files and directories to duplicate.",
 (25, 50, 45, 230),
 "Files and Directories to duplicate:",
 "",
 MultiInputFilesAndDirs ()
),

```

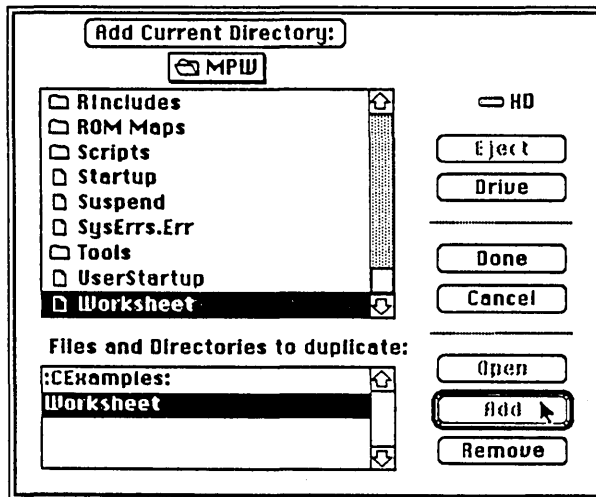


Figure 14-26  
Example of a "directories" control for multiple input files

---

## Multiple new files

The case MultiFiles also gives the user the ability to select multiple files for output.

Here is the resource code resulting in the example shown in Figure 14-27.

```
notDependent (), MultiFiles (
 "New Files...",
 "Help message for button",
 {110, 330, 130, 468},
 "New files to open:",
 "-n",
 MultiOutputFiles (),
);
```

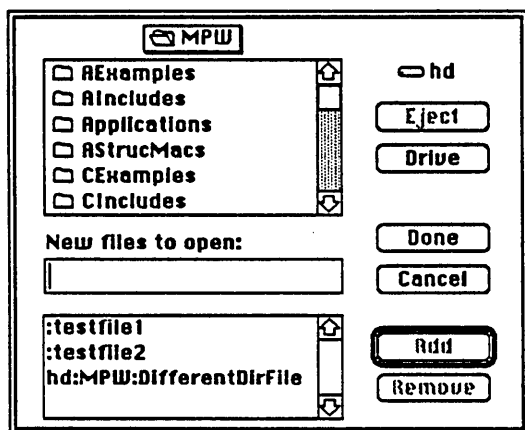


Figure 14-27

Using the MultiOutputFiles subcase of the case MultiFiles

---

---

## A Commando example

The best way to learn how to make a Commando interface is to study an actual Commando resource for an existing MPW tool. Choose a tool, explore the operation of the controls in its Commando dialog, and then use DeRez to generate a readable version the tool's Cmdo.r resource.

To obtain the Commando resource for a tool, use this syntax:

```
DeRez {MPW}Tools: toolname Cmdo.r -only cmdo
```

To obtain the Commando resource for a Shell command, use this syntax:

```
DeRez {MPW}"MPW Shell" Cmdo.r -only "'cmdo'(@"commandname @")"
```



For your convenience, the Commando resource for ResEqual, ResEqual.r, is shown here. You can find this file in the PExamples folder.

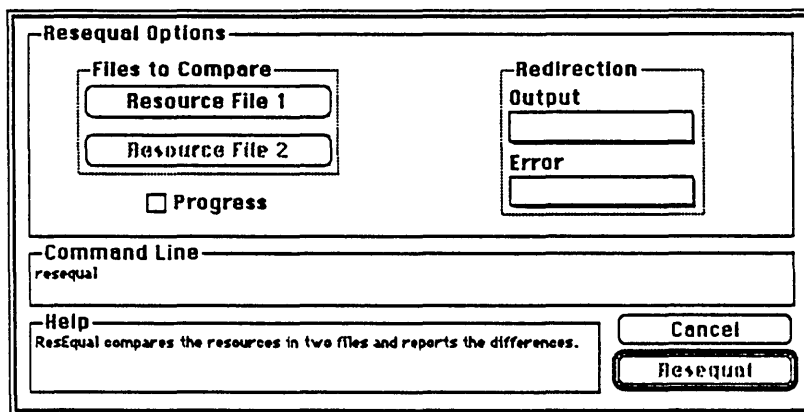
```
#include "cmdo.r"
resource 'cmdo' (355) {
 {
 240,
 "ResEqual compares the resources in two files and reports the differences.",
 {
 NotDependent {}, Files {
 InputFile,
 RequiredFile {
 {40, 40, 60, 190},
 "Resource File 1",
 "",
 "Select the first file to compare.",
 },
 Additional {
 "",
 FilterTypes,
 "Only applications, DA's, and tools",
 "All files",
 {
 appl,
 tool,
 da
 }
 }
 },
 Or {{1}}, Files {
 InputFile,
 RequiredFile {
 {70, 40, 90, 190},
 "Resource File 2",
 "",
 "Select the second file to compare.",
 },
 Additional {
 "",
 FilterTypes,
 "Only applications, DA's, and tools",
 "All files",
 {
 appl,
 tool,
 da
 }
 }
 },
 NotDependent {}, TextBox {
 gray,
 {30, 35, 95, 195},
 "Files to Compare"
 },
 NotDependent {}, CheckOption {
 NotSet,
 {105, 75, 121, 155},
 "Progress",
 "-p",
 "Write progress information to diagnostic output."
 },
 }
 }
}
```

```

 NotDependent {}, Redirection {
 StandardOutput,
 (40, 300)
 },
 NotDependent {}, Redirection {
 DiagnosticOutput,
 (80, 300)
 },
 NotDependent {}, TextBox {
 gray,
 (30, 295, 121, 420),
 "Redirection"
 },
 Or {{2}}, DoItButton {
 },
 }
};

```

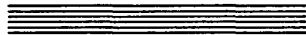
The above resource code generates the frontmost dialog box of ResEqual, which appears in Figure 14-28.



**Figure 14-28**  
A Commando example: frontmost ResEqual dialog box



## **Chapter 15**



### **Writing a Desk Accessory or Other Driver Resource**

This chapter documents the DRVRRuntime library and describes the specifics of writing a desk accessory or other driver by using MPW and MPW Pascal or MPW C. The material in this chapter is specific to the MPW programming languages. Because a desk accessory is a special case of a driver, all of the information in this chapter applies to both. You should already be familiar with the following:

- “Writing Your Own Desk Accessories” in the Desk Manager chapter of *Inside Macintosh*
- The Device Manager chapter of *Inside Macintosh* (for information about 'DRVVR' resources, and so on)
- “Building a Desk Accessory or Driver” in Chapter 9 of this reference

For information about the actual routines used in Pascal, C, or assembly language, see the appropriate MPW language reference manual.

---

---

## The DRVRRuntime library

Desk accessories have traditionally been written in assembly language, partly because of the peculiar 'DRVVR' resource format used for desk accessories. Setting up the 'DRVVR' layout header, passing register-based procedure parameters, and coping with the nonstandard exit conventions of the driver routines have made it difficult to implement desk accessories in higher-level languages. To overcome these difficulties and simplify the task of writing a desk accessory in Pascal or C, MPW provides the following:

- The library DRVRRuntime.o, which contains the “glue” for setting up your *open*, *prime*, *status*, *control*, and *close* routines
- The resource type 'DRVW', declared in {RIncludes}MPWTypes.r. The 'DRVW' resource is an intermediate form of the 'DRVVR' resource, and contains constants that point to the addresses of the driver routines in DRVRRuntime.o

The DRVRRuntime library contains a main entry point that overrides the main entry point in CRuntime.o or in your Pascal or assembly-language source. The DRVRRuntime entry point contains driver glue that sets up the parameters for you, calls your routines, and performs the special exit procedure required for a desk accessory to return control to the system. Your routines perform the actions of the desk accessory, such as opening a window and responding to mouse clicks in it.

The Resource Compiler input (resource description file) for your desk accessory includes the details of your desk accessory header, such as its driver flags, event mask, menu ID, and driver name. The driver is built by coercing the intermediate 'DRVW' resource to a resource of type 'DRVVR', which is the format required for desk accessories. Your resource description file also specifies resources for strings, windows, and menus, if used in your desk accessory. (For an example of such a resource description file, see “The Desk Accessory Resource File” in Chapter 9.)

Using DRVRRuntime.o has several advantages:

- No assembly-language source code is required.
- The Resource Compiler is an integral step in the build process, permitting the easy addition of a desk accessory menu or other owned resources.
- The programmer's interface to the *open*, *prime*, *status*, *control*, and *close* routines uses standard calling conventions. Each function returns a result code which is passed back to the system.

- The DRVRRuntime glue handles the proper exit conventions. (Drivers have peculiar exit conventions, requiring immediate calls to exit via an RTS instruction, but non-immediate calls to jump to the IODone routine—these exit procedures cannot be expressed in C or Pascal.)

Together, the DRVRRuntime library and the 'DRVW' resource form the dispatch mechanism to your driver routines. The next section describes the structure of your driver routines.

---

---

## What your routines need to do

To write a driver, you need to write five functions named DRVROpen, DRVPrime, DRVStatus, DRVControl, and DRVRClose.

- ❖ *Pascal note:* In Pascal, you'll need to write a unit that declares these five functions in your interface.

Each of these functions is declared to use Pascal calling conventions, so that the DRVRRuntime library is available for use by both C and Pascal programmers. (See the appropriate language reference manual for details.)

The calling sequence for all five driver routines is the same: The parameter `ioPB` is the pointer to the driver's I/O parameter block (passed from the system in register A0), and `dCtl` is the pointer to the driver's device control entry (from register A1). The function returns a result code, which DRVRRuntime puts in register D0. This result code is a Pascal integer (C short). Desk accessories always return a result code of 0.

For example, here is the Pascal declaration for your DRVROpen routine:

```
FUNCTION DRVROpen(ct1PB: ParmBlkPtr; dCtl: DCtlPtr): OSErr;
```

Types `ParmBlkPtr` and `DCtlPtr` are declared in the file `OSIntf.p`. Type `OSErr` is an `INTEGER`, and is also defined in `OSIntf.p`.

In C, you would need to write the routines as follows:

```
pascal OSErr
DRVROpen(ct1PB,dCtl)
 CntrlParam *ct1PB;
 DCtlPtr dCtl;
{
 ...
 return(resultCode);
}
```

Types `CntrlParam` and `DCtlPtr` are declared in the file `Devices.h`. Type `OSErr` is a short, and is defined in `Types.h`.

- ❖ *Desk accessories only:* The body of the desk accessory code will reside in your routines `DRVROpen`, `DRVControl`, and `DRVRClose`. Your routines `DRVPrime` and `DRVStatus` are never called by the system, but the DRVRRuntime library expects them to be present anyway—they cannot be omitted. It is sufficient to declare them and have them simply return 0.

---

---

## Programming hints

In the current release of MPW, global data is not available for use by desk accessories. That is, variables declared outside your functions cannot be used. In particular, the following language constructs reference the global data area and cannot be used:

Asm     DATA directives  
Pascal   UNIT variables  
C        static or extern variables; string constants

Also note that QuickDraw globals cannot be used directly. Further, you cannot call library functions that use any of these things. (Look for the Linker message "No global data was allocated.")

❖ *Note:* Apple is investigating the use of A5-based global variables in desk accessories. Currently several Macintosh applications contain trap-override or ROM hook routines that expect A5 to point to the application's globals, but without saving, setting, and restoring A5 to ensure that this is the case. Such applications are incompatible with desk accessories that use A5, because the desk accessory's calls to the ROM could end up in the application's trap-override or hook code.

Typically, C and Pascal programmers will allocate global storage from the heap and use 'STR#' resources for string constants. If you need to allocate global data from the heap, you can declare a record containing all of the global variables you need. In your DRVROpen routine, you should allocate memory from the heap with the size of this record, and store its handle in the dCtlStorage field of the device control entry. Then, to reference an element in the record, you can use this handle to reference the global variable that you want to use.

---

---

## Sample desk accessory

A sample desk accessory, Memory, is included in the Examples folders for assembly language, C, and Pascal. This desk accessory has the following features:

- It displays the current amount of free space in both the application heap and the system heap.
- It displays the number of free bytes on the default volume, along with the name of the default volume.
- It performs these operations every five seconds, so that you can see how your memory conditions change.

For instructions on building this desk accessory, see the Instructions file in the Examples folder, or refer to "Building a Desk Accessory or Driver" in Chapter 9.



# Appendix A



## Macintosh Workshop Files

This appendix lists all of the files provided with the Macintosh Programmer's Workshop 2.0.2, including MPW Pascal and MPW C. A list of MPW files as they are distributed on 800K disks appears below; a list of the files in their standard configuration on a hard disk begins on page A-7. (Volume names are shown in bold; directory names begin and end with a colon.)

---

---

### Distribution files

---

#### Distribution disk MPW1:

##### Files

MacsBug  
'MPW Shell'  
MPW.Help  
Quit  
Resume  
Startup  
Suspend  
SysErrs.Err  
UserStartup  
Worksheet

##### MPW1:Applications:

ResEdit

##### MPW1:Examples:

AddMenus  
Instructions  
Lookup  
'Startup, etc.'  
State  
'Unix Aliases'

**MPW1:Libraries:**

DRVRRuntime.o  
Interface.o  
ObjLib.o  
PerformLib.o  
Runtime.o  
SERD  
ToolLibs.o

**MPW1:RIncludes:**

Cmdo.r  
MPWTypes.r  
Size.R  
SysTypes.r  
Types.r

**MPW1:Scripts:**

BuildCommands  
BuildMenu  
BuildProgram  
CreateMake  
DirectoryMenu  
Line  
SetDirectory

---

**Distribution disk MPW2:****MPW2:Tools:**

Canon  
Commando  
Count  
CVTObj  
DeRez  
DumpCode  
DumpObj  
GetListItem  
Lib  
Link  
Make  
PerformReport  
Print  
ResEqual  
Rez  
RezDet  
Search  
Translate



---

## Distribution disk MPW3:

### 'MPW3:More Tools:'

Backup  
Canon.Dict  
Compare  
Entab  
FileDiv  
GetErrorText  
GetFileName  
MakeErrorFile  
ProcNames  
SetVersion

### 'MPW3:ROM Maps:'

MacIIROM.map  
MacPlusROM.map  
MacSEROM.map

---

---

## MPW Assembler files

---

## Distribution disk MPW Assembler1:

### 'MPW Assembler1'

Asm  
MDSCvt  
MDSCvt.Directives  
TLACvt  
TLACvt.Directives

---

## Distribution disk MPW Assembler2:

### 'MPW Assembler2:AEExamples:'

Count.a  
Count.r  
Instructions  
MakeFile  
Memory.a  
Sample.a  
Sample.r  
Stubs.a

### 'MPW Assembler2:AIncludes:'

ApplDeskBus.a  
Atalkequ.a  
FixMath.a  
FSEqu.a  
FSPrivate.a  
Graf3DEqu.a  
HardwareEqu.a  
IntEnv.a

ObjMacros.a  
PackMacs.a  
PaletteEqu.a  
PickerEqu.a  
PrEqu.a  
PrintCallsEqu.a  
PrintTrapsEqu.a  
Private.a  
QuickEqu.a  
ROMEQu.a  
SANEMacs.a  
ScriptEqu.a  
SCSIEQu.a  
ShutDownEqu.a  
Signal.a  
SlotEqu.a  
SonyEqu.a  
Sound.a  
SysEqu.a  
SysErr.a  
TimeEqu.a  
ToolEqu.a  
Traps.a  
VideoEqu.a

**'MPW Assembler2:AStructMacs:'**

FlowCtlMacs.a  
ProgStrucMacs.a  
Sample.a

---

---

## **MPW Pascal files**

Pascal  
PasMat  
PasRef

**'MPW Pascal:'PEXamples:**

Instructions  
MakeFile  
Memory.p  
Memory.r  
ResEd.p  
ResEd68K.a  
ResEqual.p  
ResEqual.r  
ResXXXXEd.p  
Sample.p  
Sample.r  
Stubs.a  
TestPerf.p

**'MPW Pascal:'Interfaces:**

AppleTalk.p  
CursorCtl.p  
ErrMgr.p  
FixMath.p  
Graf3D.p  
IntEnv.p  
MacPrint.p  
MemTypes.p  
ObjIntf.p  
OSIntf.p  
PackIntf.p  
PaletteMgr.p  
PasLibIntf.p  
Perf.p  
PickerIntf.p  
PrintTraps.p  
Quickdraw.p  
ROMDefs.p  
SANE.p  
Script.p  
SCSIIntf.p  
Signal.p  
Sound.p  
ToolIntf.p  
VideoIntf.p

**'MPW Pascal:'Libraries:**

PasLib.o  
SANELib.o  
SANELib881.o

---

---

## **MPW C files**

### **C**

**'MPW C:'CEXamples:**

Count.c  
Count.r  
Instructions  
MakeFile  
Memory.c  
Memory.r  
Sample.c  
Sample.r  
Stubs.c  
TestPerf.c

**'MPW C:'Includes:**

AppleTalk.h  
Controls.h  
CType.h  
CursorCtl.h  
Desk.h  
DeskBus.h  
Devices.h  
Dialogs.h  
DiskInit.h  
Disks.h  
ErrorMgr.h  
ErrNo.h  
Errors.h  
Events.h  
FCntl.h  
Files.h  
FixMath.h  
Fonts.h  
Graf3D.h  
IOCtl.h  
Lists.h  
Math.h  
Memory.h  
Menus.h  
OSEvents.h  
OSUtils.h  
Packages.h  
Palette.h  
Perf.h  
Picker.h  
Printing.h  
PrintTraps.h  
Quickdraw.h  
Resources.h  
Retrace.h  
ROMDefs.h  
SANE.h  
Scrap.h  
Script.h  
SCSI.h  
SegLoad.h  
Serial.h  
SetJump.h  
ShutDown.h  
Signal.h  
Slots.h  
Sound.h  
Start.h  
StdIO.h  
String.h  
Strings.h  
TextEdit.h  
Time.h  
ToolUtils.h

Traps.h  
Types.h  
Values.h  
VarArgs.h  
Video.h  
Windows.h

**'MPW C:'CLibraries:**

CInterface.o  
CLib881.o  
CRuntime.o  
CSANELib.o  
CSANELib881.o  
Math.o  
Math881.o  
StdCLib.o

---

---

## Hard disk configuration

**HardDisk:MPW:**

:AExamples:  
:AIncludes:  
:Applications:  
:AStructMacs:  
:CExamples:  
:CIncludes:  
:CLibraries:  
:Examples:  
:Libraries:  
:PEXamples:  
:PInterfaces:  
:PLibraries:  
:RIncludes:  
'ROM Maps:'  
:Scripts:  
:Tools:  
MacsBug  
'MPW Shell'  
MPW.Help  
Quit  
Resume  
Startup  
Suspend  
SysErrs.Err  
UserStartup  
Worksheet

**HardDisk:MPW:AExamples:**

Count.a  
Count.r  
Instructions  
MakeFile  
Memory.a  
Sample.a  
Sample.r  
Stubs.a

**HardDisk:MPW:AIncludes:**

ApplDeskBus.a  
ATalkEqu.a  
FixMath.a  
FSEqu.a  
FSPrivate.a  
Graf3DEqu.a  
HardwareEqu.a  
IntEnv.a  
ObjMacros.a  
PackMacs.a  
PaletteEqu.a  
PickerEqu.a  
PrEqu.a  
PrintCallsEqu.a  
PrintTrapsEqu.a  
Private.a  
QuickEqu.a  
ROMEQu.a  
SANEMacs.a  
ScriptEqu.a  
SCSIEqu.a  
ShutDownEqu.a  
Signal.a  
SlotEqu.a  
SonyEqu.a  
Sound.a  
SysEqu.a  
SysErr.a  
TimeEqu.a  
ToolEqu.a  
Traps.a  
VideoEqu.a

**HardDisk:MPW:Applications:**

ResEdit

**HardDisk:MPW:AStructMacs:**

FlowCtlMacs.a  
ProgStrucMacs.a  
Sample.a

**HardDisk:MPW:CExamples:**

Count.c  
Count.r  
Instructions  
MakeFile  
Memory.c  
Memory.r  
Sample.c  
Sample.r  
Stubs.c  
TestPerf.c

**HardDisk:MPW:CIncludes:**

AppleTalk.h  
Controls.h  
CType.h  
CursorCtl.h  
Desk.h  
DeskBus.h  
Devices.h  
Dialogs.h  
DiskInit.h  
Disks.h  
ErrMgr.h  
ErrNo.h  
Errors.h  
Events.h  
FCntl.h  
Files.h  
FixMath.h  
Fonts.h  
Graf3D.h  
IOCtl.h  
Lists.h  
Math.h  
Memory.h  
Menus.h  
OSEvents.h  
OSUtils.h  
Packages.h  
Palette.h  
Perf.h  
Picker.h  
Printing.h  
PrintTraps.h  
Quickdraw.h  
Resources.h  
Retrace.h  
ROMDefs.h  
SANE.h  
Scrap.h  
Script.h  
SCSI.h  
SegLoad.h  
Serial.h  
SetJump.h  
ShutDown.h  
Signal.h  
Slots.h  
Sound.h  
Start.h  
StdIO.h  
String.h  
Strings.h  
TextEdit.h  
Time.h  
ToolUtils.h  
Traps.h

Types.h  
Values.h  
VarArgs.h  
Video.h  
Windows.h

**HardDisk:MPW:CLibraries:**

CInterface.o  
CLib881.o  
CRuntime.o  
CSANELib.o  
CSANELib881.o  
Math.o  
Math881.o  
StdCLib.o

**HardDisk:MPW:Examples:**

AddMenus  
Instructions  
Lookup  
'Startup, etc.  
State  
'Unix Aliases'

**HardDisk:MPW:Libraries:**

DRVRRuntime.o  
Interface.o  
ObjLib.o  
PerformLib.o  
Runtime.o  
SERD  
ToolLibs.o

**HardDisk:MPW:PEexamples:**

Instructions  
MakeFile  
Memory.p  
Memory.r  
ResEd.p  
ResEd68K.a  
ResEqual.p  
ResEqual.r  
ResXXXXEd.p  
Sample.p  
Sample.r  
Stubs.a  
TestPerf.p



**HardDisk:MPW:PInterfaces:**

AppleTalk.p  
CursorCtl.p  
ErrMgr.p  
FixMath.p  
Graf3D.p  
IntEnv.p  
MacPrint.p  
MemTypes.p  
ObjIntf.p  
OSIntf.p  
PackIntf.p  
PaletteMgr.p  
PasLibIntf.p  
Perf.p  
PickerIntf.p  
PrintTraps.p  
Quickdraw.p  
ROMDefs.p  
SANE.p  
Script.p  
SCSIIntf.p  
Signal.p  
Sound.p  
ToolIntf.p  
VideoIntf.p

**HardDisk:MPW:PLibraries:**

PasLib.o  
SANELib.o  
SANELib881.o

**HardDisk:MPW:RIcludes:**

Cmdo.r  
MPWTypes.r  
Size.R  
SysTypes.r  
Types.r

**HardDisk:MPW':ROM Maps:'**

MacIIROM.map  
MacPlusROM.map  
MacSEROM.map

**HardDisk:MPW:Scripts:**

BuildCommands  
BuildMenu  
BuildProgram  
CreateMake  
DirectoryMenu  
Line  
SetDirectory

**HardDisk:MPW:Tools:**

Asm  
Backup  
C  
Canon  
Canon.Dict  
Commando  
Compare  
Count  
CVTObj  
DeRez  
DumpCode  
DumpObj  
Entab  
FileDiv  
GetErrorText  
GetFileName  
GetListItem  
Lib  
Link  
Make  
MakeErrorFile  
MDSCvt  
MDSCvt.Directives  
Pascal  
PasMat  
PasRef  
PerformReport  
Print  
ProcNames  
ResEqual  
Rez  
RezDet  
Search  
SetVersion  
TLACvt  
TLACvt.Directives  
Translate

---

---

## Appendix B

---

---

# Summary of Selections and Regular Expressions

This appendix formally defines the syntax of selections and regular expressions as used in the Shell command language. It also lists the Option-key characters used in selections and regular expressions. For examples of their use, see Chapter 6.

---

---

## Selections

Selections are passed as arguments to the editing commands. They're defined in Table B-1.

**Table B-1**  
Selections

---

|                                            |                                                                            |
|--------------------------------------------|----------------------------------------------------------------------------|
| <i>selection</i>                           | (specifies a selection or insertion point)                                 |
| <b>§</b>                                   | Current selection                                                          |
| <i>name</i>                                | Identifies marked text                                                     |
| <i>number</i>                              | Line number                                                                |
| <b>!</b> <i>number</i>                     | <i>number</i> lines after the end of the current selection                 |
| <b>;</b> <i>number</i>                     | <i>number</i> lines before the start of the current selection              |
| <i>position</i>                            | Position (defined below)                                                   |
| <i>pattern</i>                             | Pattern (defined below)                                                    |
| <b>(</b> <i>selection</i> <b>)</b>         | Selection grouping                                                         |
| <i>selection</i> <b>:</b> <i>selection</i> | Both selections and everything in between                                  |
| <i>posillon</i>                            | (specifies an insertion point)                                             |
| <b>•</b>                                   | Position before the first character in the file                            |
| <b>∞</b>                                   | Position after the last character in the file                              |
| <b>Δ</b> <i>selection</i>                  | Position before the first character of <i>selection</i>                    |
| <i>selection</i> <b>Δ</b>                  | Position after the last character of <i>selection</i>                      |
| <i>selection</i> <b>!</b> <i>number</i>    | Position <i>number</i> characters after the end of <i>selection</i>        |
| <i>selection</i> <b>;</b> <i>number</i>    | Position <i>number</i> characters before the beginning of <i>selection</i> |
| <i>pattern</i>                             | (specifies characters to be matched)                                       |
| <b>/</b> <i>entireRegularExpr</i> <b>/</b> | Regular expression—search forward (see Table B-2)                          |
| <b>\</b> <i>entireRegularExpr</i> <b>\</b> | Regular expression—search backward                                         |

This is the precedence of the selection operators, from highest to lowest:

/ and \  
 ( )  
 Δ  
 ! and ;  
 :

---



---

## Regular expressions

Regular expressions are used for pattern matching within /.../ and \...\ (See “*pattern*” in Table B-1.) Regular expressions are defined in Table B-2.

**Table B-2**  
 Regular expressions

|                                                                     |                                                                                                 |
|---------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| <b><i>enlireRegularExpr</i></b>                                     |                                                                                                 |
| • <i>regularExpr</i>                                                | Regular expression at beginning of line                                                         |
| <i>regularExpr</i> ∞                                                | Regular expression at end of line                                                               |
| <i>regularExpr</i>                                                  | Regular expression                                                                              |
| <b><i>regularExpr</i></b>                                           |                                                                                                 |
| <i>simpleExpr</i>                                                   | Untagged regular expression                                                                     |
| <i>taggedExpr</i>                                                   | Tagged regular expression                                                                       |
| <i>literal</i>                                                      | Quoted string literal                                                                           |
| <i>regularExpr</i> <sub>1</sub> <i>regularExpr</i> <sub>2</sub>     | <i>regularExpr</i> <sub>1</sub> followed by <i>regularExpr</i> <sub>2</sub>                     |
| <b><i>simpleExpr</i></b>                                            |                                                                                                 |
| ( <i>regularExpr</i> )                                              | Regular expression grouping                                                                     |
| <i>characterExpr</i>                                                | Single-character regular expression                                                             |
| <i>simpleExpr</i> *                                                 | Regular expression zero or more times                                                           |
| <i>simpleExpr</i> +                                                 | Regular expression one or more times                                                            |
| <i>simpleExpr</i> « <i>number</i> »                                 | Regular expression <i>number</i> times                                                          |
| <i>simpleExpr</i> « <i>number</i> ,»                                | Regular expression at least <i>number</i> times                                                 |
| <i>simpleExpr</i> « <i>n</i> <sub>1</sub> , <i>n</i> <sub>2</sub> » | Regular expression at least <i>n</i> <sub>1</sub> times and at most <i>n</i> <sub>2</sub> times |
| <b><i>taggedExpr</i></b>                                            |                                                                                                 |
| ( <i>regularExpr</i> )@ <i>digit</i>                                | The string matched by the <i>regularExpr</i> can be referred to as @ <i>digit</i>               |
| <b><i>literal</i></b>                                               |                                                                                                 |
| ' <i>string</i> '                                                   | Each character in <i>string</i> is taken literally                                              |
| " <i>string</i> "                                                   | Each character in <i>string</i> is taken literally, except for ∂ and { } substitutions          |
| <b><i>characterExpr</i></b>                                         |                                                                                                 |
| <i>character</i>                                                    | Character (unless it's listed as special following the table)                                   |
| ∂ <i>character</i>                                                  | ∂ defeats special meaning of following character                                                |
| ?                                                                   | Any character except Return                                                                     |
| =                                                                   | Any string not containing a Return, including the null string (this is the same as ?*)          |
| [ <i>characterList</i> ]                                            | Any character in the list                                                                       |
| [ ¬ <i>characterList</i> ]                                          | Any character not in the list                                                                   |

**Table B-2 (continued)**  
Regular expressions

---

|                                                      |                                                                                             |
|------------------------------------------------------|---------------------------------------------------------------------------------------------|
| <i>characterList</i>                                 |                                                                                             |
| ]                                                    | "]" first in list represents itself                                                         |
| -                                                    | "-" first in list represents itself                                                         |
| <i>character</i>                                     | Character                                                                                   |
| <i>characterList character</i>                       | List of characters                                                                          |
| <i>character<sub>1</sub> - character<sub>2</sub></i> | Character range from <i>character<sub>1</sub></i> to <i>character<sub>2</sub></i> inclusive |

*Note:* The regular expression operators ?, ~, [...], \*, +, and «...» are also used in filename generation.

The following characters have special meanings:

|                     |                                                          |
|---------------------|----------------------------------------------------------|
| ∂                   | Always special, except within '...'                      |
| ? ~ * + [ « ( ) ' " | Special everywhere except within [...], '...', and "..." |
| ®                   | Special only after a right parenthesis character, )      |
| •                   | Special as first character of entire regular expression  |
| ∞                   | Special as last character of entire regular expression   |
| / \                 | Special if used to delimit regular expression            |
| { }                 | Special everywhere except within '...'                   |

The operators are listed below beginning with the highest-precedence operators.

|                   |  |
|-------------------|--|
| ( )               |  |
| ? ~ * + [ ] « » ® |  |
| concatenation     |  |
| • ∞               |  |

---



---

## Option-key characters

The following Option-key characters are used in selections and regular expressions.

| Character | Key            | Meaning                         |
|-----------|----------------|---------------------------------|
| §         | Option-6       | Current selection character     |
| ∂         | Option-d       | Escape character                |
| =         | Option-x       | Any string                      |
| •         | Option-8       | Beginning of line or file       |
| ∞         | Option-5       | End of line or file             |
| ¡         | Option-1       | Minus number of lines or spaces |
| Δ         | Option-j       | Position                        |
| ®         | Option-r       | Tag operator                    |
| <<        | Option-\       | Encloses number of repetitions  |
| >>        | Shift-Option-\ | Encloses number of repetitions  |



# Appendix C

## MPW Character Reference

This appendix gives a brief summary of the special operators used in the Macintosh Programmer's Workshop. For characters that are part of the extended character set, Option-key combinations are also given. For details on the action of these operators, see Chapters 5 and 6.

**Table C-1**  
MPW operators

| Operator                | Meaning                                                                                                                             |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <b>Shell characters</b> |                                                                                                                                     |
| space                   | Separates words                                                                                                                     |
| tab                     | Separates words                                                                                                                     |
| return                  | Separates commands                                                                                                                  |
| ;                       | Separates commands                                                                                                                  |
|                         | Pipe—separates commands, piping output to input                                                                                     |
| &&                      | “And”—separates commands, executing second if first succeeds                                                                        |
|                         | “Or”—separates commands, executing second if first fails                                                                            |
| ( <i>commands</i> )     | Group commands                                                                                                                      |
| # <i>comment</i>        | Ignore <i>comment</i>                                                                                                               |
| <i>char</i>             | Escape—literalizes <i>char</i> ; <i>dn</i> , <i>dt</i> , and <i>df</i> are special ( <i>d</i> is Option-D)                          |
| ' <i>chars</i> '        | “Hard quotes”—literalize <i>chars</i>                                                                                               |
| " <i>chars</i> "        | “Soft quotes”—literalize <i>chars</i> except for {...} (variable substitution), `...` (command substitution), and <i>d</i> (escape) |
| / <i>chars</i> /        | Regular expression quotes—literalize <i>chars</i> except for {...}, `...`, and <i>d</i>                                             |
| ...                     | Ellipsis (Option-semicolon; not three periods) following a command invokes Commando                                                 |

(continued)

**Table C-1 (continued)**  
MPW operators

| Operator                                                 | Meaning                                                                                                                           |
|----------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>Shell characters</b>                                  |                                                                                                                                   |
| <code>\chars\</code>                                     | Regular expression quotes—literalize <code>\chars\</code> except for <code>{...}</code> , <code>^...^</code> , and <code>∂</code> |
| <code>{variable}</code>                                  | Substitute <i>variable</i>                                                                                                        |
| <code>`command`</code>                                   | Substitute output of <i>command</i>                                                                                               |
| <code>&lt; filename</code>                               | Redirect standard input                                                                                                           |
| <code>&gt; filename</code>                               | Redirect standard output, replacing contents of <i>filename</i>                                                                   |
| <code>&gt;&gt; filename</code>                           | Redirect standard output, appending to <i>filename</i>                                                                            |
| <code>≥ filename</code>                                  | Redirect diagnostics, replacing contents of <i>filename</i> (≥ is Option->)                                                       |
| <code>≥≥ filename</code>                                 | Redirect diagnostics, appending to <i>filename</i> (≥ is Option->)                                                                |
| <b>Selections (editing commands)</b>                     |                                                                                                                                   |
| <code>§</code>                                           | Current selection (§ is Option-6)                                                                                                 |
| <code>n</code>                                           | Line number <i>n</i>                                                                                                              |
| <code>!n</code>                                          | <i>n</i> lines after §                                                                                                            |
| <code>jn</code>                                          | <i>n</i> lines before § ( <i>j</i> is Option-1)                                                                                   |
| <code>•</code>                                           | Beginning of file (• is Option-8)                                                                                                 |
| <code>∞</code>                                           | End of file (∞ is Option-5)                                                                                                       |
| <code>Δselection</code>                                  | Beginning of <i>selection</i> (Δ is Option-j)                                                                                     |
| <code>selectionΔ</code>                                  | End of <i>selection</i> (Δ is Option-j)                                                                                           |
| <code>selection !n</code>                                | <i>n</i> characters before <i>selection</i>                                                                                       |
| <code>selection jn</code>                                | <i>n</i> characters after <i>selection</i> ( <i>j</i> is Option-1)                                                                |
| <code>/regExpr/</code>                                   | <i>regExpr</i> after current selection                                                                                            |
| <code>\regExpr\</code>                                   | <i>regExpr</i> before current selection                                                                                           |
| <code>selection<sub>1</sub>:selection<sub>2</sub></code> | <i>selection<sub>1</sub></i> , <i>selection<sub>2</sub></i> and in between                                                        |
| <b>Regular expressions (and filename generation)</b>     |                                                                                                                                   |
| <code>char</code>                                        | Match <i>char</i>                                                                                                                 |
| <code>∂char</code>                                       | Literal <i>char</i> (∂ is Option-D)                                                                                               |
| <code>'char...'</code>                                   | Literal <i>chars</i>                                                                                                              |
| <code>?</code>                                           | Any character                                                                                                                     |
| <code>=</code>                                           | Zero or more chars (short for <code>?*</code> ) (= is Option-X)                                                                   |
| <code>[characterList]</code>                             | Any character in <i>characterList</i>                                                                                             |
| <code>[¬characterList]</code>                            | Any character not in <i>characterList</i> (¬ is Option-L)                                                                         |
| <code>regExpr*</code>                                    | <i>regExpr</i> zero or more times                                                                                                 |
| <code>regExpr+</code>                                    | <i>regExpr</i> one or more times                                                                                                  |
| <code>regExpr«n»</code>                                  | <i>regExpr</i> <i>n</i> times (« and » are Option-\ and Option-Shift-\)                                                           |
| <code>regExpr«n,»</code>                                 | <i>regExpr</i> <i>n</i> or more times                                                                                             |
| <code>regExpr«n<sub>1</sub>,n<sub>2</sub>»</code>        | <i>regExpr</i> <i>n<sub>1</sub></i> to <i>n<sub>2</sub></i> times                                                                 |
| <code>(regExpr)</code>                                   | <i>regExpr</i> (grouping)                                                                                                         |
| <code>(regExpr)@n</code>                                 | Tagged <i>regExpr</i> , where $0 \leq n \leq 9$ (@ is Option-R)                                                                   |
| <code>regExpr<sub>1</sub>regExpr<sub>2</sub></code>      | <i>regExpr<sub>1</sub></i> followed by <i>regExpr<sub>2</sub></i>                                                                 |
| <code>regExpr</code>                                     | <i>regExpr</i> at beginning of line (• is Option-8)                                                                               |
| <code>regExpr∞</code>                                    | <i>regExpr</i> at end of line (∞ is Option-5)                                                                                     |



**Table C-1 (continued)**  
MPW operators

| Operator                               | Meaning                                         |
|----------------------------------------|-------------------------------------------------|
| <b>Shell numbers</b>                   |                                                 |
| $\$nnn$                                | Hexadecimal number                              |
| $0xnnn$                                | Hexadecimal number                              |
| <b>Shell operators (by precedence)</b> |                                                 |
| $(expr)$                               | Expression grouping                             |
| $-$                                    | (unary) arithmetic negation                     |
| $\sim$                                 | (unary) bitwise negation                        |
| $!$ NOT $\neg$                         | (unary) logical negation ( $\neg$ is Option-L)  |
| $*$                                    | Multiplication                                  |
| $+$ DIV                                | Division (+ is Option-/)                        |
| $\%$ MOD                               | Modulus                                         |
| $+$                                    | Addition                                        |
| $-$                                    | Subtraction                                     |
| $<<$                                   | Shift left                                      |
| $>>$                                   | Shift right (logical)                           |
| $<$                                    | Less than                                       |
| $<=$ $\leq$                            | Less than or equal ( $\geq$ is Option- $<$ )    |
| $>$                                    | Greater than                                    |
| $>=$ $\geq$                            | Greater than or equal ( $\geq$ is Option- $>$ ) |
| $==$                                   | Equal                                           |
| $!=$ $<>$ $\neq$                       | Not equal ( $\neq$ is Option- $=$ )             |
| $=~$                                   | Matches regular expression                      |
| $!~$                                   | Does not match regular expression               |
| $\&$                                   | Bitwise AND                                     |
| $\wedge$                               | Bitwise XOR                                     |
| $ $                                    | Bitwise OR                                      |
| $\&\&$ AND                             | Logical AND                                     |
| $  $ OR                                | Logical OR                                      |



---

---

# Appendix D

---

---

## Resource Description Syntax

This appendix defines the form of a resource description file, used by the Resource Compiler and Decompiler. See Chapter 8 for information on how to use these tools.

---

---

### Syntax notation

The following syntax notation is used in this appendix:

|                     |                                                                           |
|---------------------|---------------------------------------------------------------------------|
| <b>terminal</b>     | Must be entered as shown                                                  |
| <i>non-terminal</i> | May be replaced by anything matching its definition                       |
| A   B   C           | Either A or B or C (vertical stacking also indicates an either/or choice) |
| {...}?              | Enclosed element is optional, but may not be repeated                     |
| {...}+              | Enclosed element may be repeated one or more times (not optional)         |
| {...}*              | Enclosed element may be repeated zero or more times                       |
| {...}n              | Enclosed element must be repeated <i>n</i> times                          |

If one of the syntax elements must be included literally, it is shown enclosed in single quotes; for example,

```
{ '(data-string)' }?
```

indicates that a *data-string* is optional, and must be enclosed in braces, if included. Otherwise, all punctuation (; , ' " \$ =) must be entered as shown.

Note that the ellipsis (three closely spaced periods) within braces signifies only some unspecified element on which an operation is to be performed. An actual ellipsis in a command line (Option-semicolon) would invoke a command's Commando dialogs.

Note that the semicolon is a statement terminator; every statement must be terminated by a semicolon. In a resource type definition, semicolons can be liberally sprinkled without ill effect. In a resource specification (where the actual resource data is initialized), commas are used everywhere to separate items, including array elements.

The nonterminal symbols used are fully defined under "Syntax" at the end of this appendix.

---

---

## Structure of a resource description file

The Resource Compiler input file consists of any number of statements, where a statement may be any of the following:

|                       |                                                                         |
|-----------------------|-------------------------------------------------------------------------|
| <code>include</code>  | Include resources from another file.                                    |
| <code>read</code>     | Read the data fork of a file and include it as a resource.              |
| <code>data</code>     | Specify raw data.                                                       |
| <code>type</code>     | Declare resource type descriptions for subsequent resource statements.  |
| <code>resource</code> | Specify data for a resource type declared in a previous type statement. |

---

### Include—include resources from another file

```
include file { include-selector }? ;
include-selector ::= resource-type { '(' ID-specifier ')' }?
 not resource-type
 resource-type1 as resource-type2
 resource-type1 '(' ID-specifier ')'
 as resource-type2 '(' resource-specifier)'

file ::= string
ID-specifier ::= ID-range
 resource-name

ID-range ::= ID { : ID }?
resource-specifier ::= resource-ID { , resource-name }? { resource-attributes }?
resource-ID ::= word-expression
resource-name ::= string
resource-attributes ::= { resource-literal-attributes } * | resource-numeric-attributes
resource-numeric-attributes ::= , byte-expression
resource-literal-attributes ::= { , sysheap | , appheap }?
 { , purgeable | , nonpurgeable }?
 { , locked | , unlocked }?
 { , preload | , nonpreload }?
```

---

### Read—read data as a resource

```
read resource-type '(' resource-specifier)' file ;
```

---

## Data—specify raw data

*data resource-type* ('*resource-specifier* ') '{ *data-string* { ; }? }' ;

---

## Type—declare resource type

*type resource-type* ('(*ID-range* ')?) '{ { *type-statement* ; } \* '}' ;

*resource-type* ::= *long-expression*

*type-statement* ::= *data-type*  
*fill-type*  
*alignment*  
*switch-type*  
*array-type*

## Data-type

*data-type* ::= *data-type-specifier* { *symbolic-declaration* | = *declaration-constant* }?

*data-type-specifier* ::= *char*  
*string* { '[' *length* ' ] ' }?  
*pstring* { '[' *length* ' ] ' }?  
*cstring* { '[' *length* ' ] ' }?  
*wstring* { '[' *length* ' ] ' }?  
*numeric-type-specifier*  
*point*  
*rect*

*length* ::= *expression*

*numeric-type-specifier* ::= *boolean*  
{ *unsigned* }? { *radix* }? *numeric-type*

*radix* ::= *binary*  
*octal*  
*decimal*  
*hex*  
*literal*

*numeric-type* ::= *byte*  
*integer*  
*longint*  
*bitstring* '[' *length* ' ]'

*symbolic-declaration* ::= *range-block* { , *range-block* }\*

*range-block* ::= *identifier* { = *declaration-constant* }?

*declaration-constant* ::= *expression*  
*point-constant*  
*rect-constant*  
*string*

### Fill-type

*fill-type* ::= fill *fill-size* {'[' *expression* ']}?  
*fill-size* ::= bit | nibble | byte | word | long

### Alignment

*alignment* ::= align *align-size*  
*align-size* ::= nibble | byte | word | long

### Switch-type

*switch-type* ::= switch {'switch-body'}  
*switch-body* ::= { case *case-name* : *case-body* }+  
*case-name* ::= *identifier*  
*case-body* ::= { *type-statement* ; }\* *key-constant-statement* ; { *type-statement* ; }\*  
*key-constant-statement* ::= key *data-type-specifier* = *declaration-constant*

### Array-type

*array-type* ::= { wide }? array { *array-specifier* }? *type-body*  
*array-specifier* ::= *array-name*  
'[' *expression* ']'  
*array-name* ::= *identifier*  
*type-body* ::= '{' { *type-statement* ; }\* '{'

---

## Resource—specify resource data

resource *resource-type* '(' *resource-specifier* ')' *data-body* ;  
*data-body* ::= '{' { *data-statement* { , *data-statement* }\* }? }'  
*data-statement* ::= *expression*  
*point-constant*  
*rect-constant*  
*string*  
*identifier*  
*switch-data*  
*array-data*  
  
*switch-data* ::= *case-name* *data-body*  
*array-data* ::= '{' { *array-element* { , *array-element* }\* }? }'  
*array-element* ::= { *data-statement* { , *data-statement* }\* }?

---

---

## Preprocessor directives

These preprocessor directives are available:

```
#define identifier { define-string }?
#undef identifier
#if preprocessor-expr
#elif preprocessor-expr
#else
#endif
#ifdef identifier
#ifndef identifier
printf (string { , [expression | string] } *)
```

*Preprocessor-expr* is the same as *expression* with the following additional expressions:

```
defined ('identifier')
defined identifier
```

---

---

## Syntax

This section defines the nonterminal symbols used in the previous sections.

---

### Identifiers

An *identifier* may consist of letters (A–Z, a–z), digits (0–9), or the underscore character (`_`). Identifiers may not start with a digit; otherwise any mix of letters, digits, and underscores is acceptable. Identifiers are not case sensitive. An identifier may be of any length.

---

### Token delimiters

```
token-delimiter ::= { space | tab | newline | comment }+
comment ::= /* { printing-character }* */
```

---

### Compound types

```
point-constant ::= {'expression , expression '}
rect-constant ::= {'expression , expression , expression , expression '}
```

---

### Expressions

```
bit-expression ::= expression
byte-expression ::= expression
word-expression ::= expression
long-expression ::= expression
```

*expression* ::= *integer-constant*  
*literal-constant*  
*numeric-variable*  
*system-function*  
*expression*  
- *expression*  
~ *expression*  
! *expression*  
'(' *expression* ')'
  
*expression* >> *expression*  
*expression* << *expression*  
*expression* ^ *expression*  
*expression* '|' *expression*  
*expression* && *expression*  
*expression* '|' *expression*  
*expression* & *expression*  
*expression* != *expression*  
*expression* == *expression*  
*expression* >= *expression*  
*expression* <= *expression*  
*expression* > *expression*  
*expression* < *expression*  
*expression* - *expression*  
*expression* + *expression*  
*expression* \* *expression*  
*expression* / *expression*  
*expression* % *expression*

*system-function* ::= \$\$countof '(' *array-name* ')'

---

## Numbers

*integer-constant* ::= *decimal-constant*  
*octal-constant*  
*binary-constant*  
*hexadecimal-constant*

*decimal-constant* ::= *nonzero-digit* { *digit* }\*  
*octal-constant* ::= 0 { *octal-digit* }\*  
*hexadecimal-constant* ::= *hex-marker* { *hex-digit* }+  
*binary-constant* ::= *binary-marker* { *binary-digit* }+

*decimal-marker* ::= 0d | 0D  
*hex-marker* ::= 0x | 0X | \$  
*binary-marker* ::= 0b | 0B

*octal-digit* ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7  
*hex-digit* ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  
A | B | C | D | E | F |  
a | b | c | d | e | f

*binary-digit* ::= 0 | 1

*literal-constant* ::= ' { *character* }\* '



---

## Variables

*string-variable* ::=            \$\$Version  
                                  \$\$Date  
                                  \$\$Time  
                                  \$\$Name  
                                  \$\$Shell('"*Shell-variable-name*"')  
                                  \$\$Resource('file, resource-id, resourceName-or-ID')  
                                  \$\$Format ( string{ , [ expression | string ] } \* )  
  
                                  *resourceName-or-ID* ::=    *resource-id*  
                                                                  *resource-name*

*numeric-variable* ::=            \$\$Hour  
                                  \$\$Minute  
                                  \$\$Second  
                                  \$\$Year  
                                  \$\$Month  
                                  \$\$Day  
                                  \$\$Weekday  
                                  \$\$Type  
                                  \$\$ID  
                                  \$\$Attributes

---

## Strings

*string* ::=                        *simple-string*  
                                  *hex-string*  
                                  *string-variable*  
                                  *string string*

*simple-string* ::=                 " { *character* } \* "

*hex-string* ::=                   \$" { *hex-digit hex-digit* } \* "

*character* ::=                    *printing-character* | *escape-character*

*escape-character* ::=            \  
                                  *escape-code*

*escape-code* ::=                 *character-escape-code* | *numeric-escape-code*

*character-escape-code* ::=       n | t | b | r | f | v | ? | \ | ' | "

*numeric-escape-code* ::=         { *octal-digit* } 3  
                                  *decimal-marker* { *decimal-digit* } 3  
                                  *hex-marker* { *hex-digit* } 2  
                                  *binary-marker* { *binary-digit* } 8





## Appendix E



# File Types, Creators, and Suffixes

---

---

### File types and creators

Table E-1 lists MPW file types and creators.

**Table E-1**  
File types and creators

| File                | Type   | Creator                    |
|---------------------|--------|----------------------------|
| MPW Shell           | 'APPL' | 'MPS ' ( <i>MPSspace</i> ) |
| Tools               | 'MPST' | 'MPS '                     |
| Text files          | 'TEXT' | 'MPS '                     |
| Object files        | 'OBJ ' | 'MPS '                     |
| Pascal load/dump    | 'DMPP' | 'MPS '                     |
| Assembler load/dump | 'DMPA' | 'MPS '                     |

---

---

### File suffixes

File suffix conventions are as follows.

---

#### Text files

|                   |                                                     |
|-------------------|-----------------------------------------------------|
| <i>name.a</i>     | Assembly-language source file                       |
| <i>name.a.lst</i> | Assembler listing file                              |
| <i>name.c</i>     | C source file                                       |
| <i>name.h</i>     | C header file                                       |
| <i>name.map</i>   | Linker map                                          |
| <i>name.p</i>     | Pascal source file                                  |
| <i>name.r</i>     | Resource description file (Resource Compiler input) |

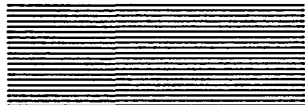
Text files are identified by their file type (TEXT) rather than by a special suffix. Several applications (including MacWrite, MDS Edit, and the MPW Shell) can create and edit files of type TEXT. The creator 'MPS ' indicates to the Finder that the MPW Shell is the application to launch when a text file is opened.

---

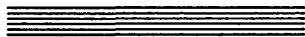
## Object files

*name.a.o*     Object file created by the Assembler  
*name.p.o*     Object file created by the Pascal Compiler  
*name.c.o*     Object file created by the C Compiler  
*name.o*       Object file (library) created by Lib; object files shipped with MPW

Compilers add the suffix ".o" to the source file name to construct the object file name. The language suffix is left in the name in order to prevent name conflicts for programs whose components are written in several languages. (For example, a program might have source files MacGismo.a and MacGismo.c and object files MacGismo.a.o and MacGismo.c.o.)



# Appendix F



## Object File Format

This appendix is addressed to programmers who are writing compilers or assemblers to run under MPW.

---

---

### Object file format

An object file consists of a sequence of object file records. These records are in the data fork of the file. There are 11 types of object file records:

- The first record in the file must be a *first record*.
- One-byte *pad records* are used to maintain word alignment.
- *Comment records* allow comments to be included in the file.
- *Dictionary records* associate names with unique IDs.
- *Module records* define code and data modules.
- *Entry-point records* define entry points in code and data modules.
- *Size records* specify the size of a module.
- *Contents records* specify the contents of a module.
- *Reference records* and *computed-reference records* specify locations in modules that contain references to other modules or entry points.
- The last record in the file must be a *last record*.

A **module** is a contiguous region of memory that contains code or static data. (The jump-table is considered to be code.) A module is the smallest unit of memory that is included or removed by the Linker. An **entry point** is a location (offset) within a module. (The module itself is treated as an entry point with offset zero.) A **segment** is a named collection of modules.

All modules, entries, and segments are given a unique, positive, 16-bit ID. An **ID** is a file-relative number for a module, an entry point, or a segment, identifying the module, entry point, or segment within a single object file.

Modules and entry points may be local or external. A **local** (module, entry point, or segment) can be referenced only from within the file where it is defined. An **external** (module, entry point, or segment) can be referenced from different files. In addition to an ID, each external module or entry point defined or referenced in an object file must also have a unique name (a string identifier) that identifies it across files. A module, entry point, or segment without a name is said to be anonymous.

Names and IDs are specified in dictionary records. Local IDs may be anonymous. (If no dictionary entry is found for it, an ID is considered anonymous.) Local modules and entries need not have unique names, and an external segment may have the same name as an external module or entry point.

At any given point in an object file, there can be one current code module and one current data module. The beginning of a new code or data module is indicated by a module record. The current code and data modules are further defined by entry point, size, contents, reference, and computed-reference records—these records can occur in any order after the module record. In each of these records, a flag bit indicates whether the record refers to the code or the data module.

The structure and semantics of each of the record types is defined below.

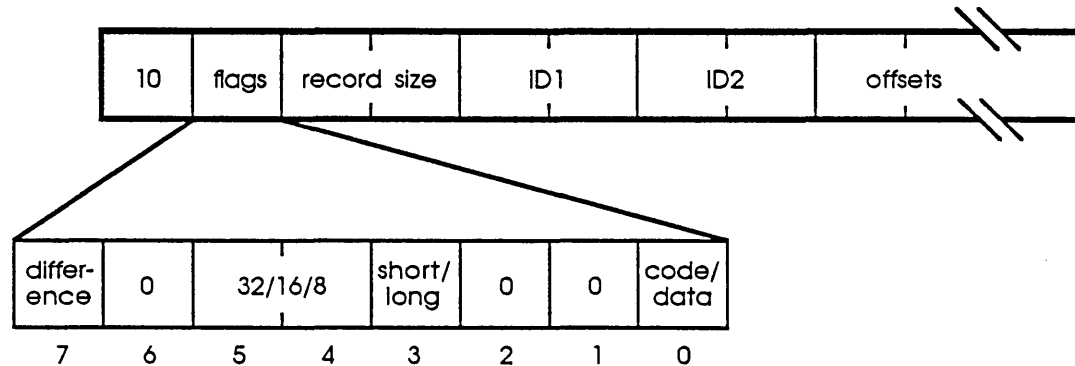
---



---

## Notation used in this appendix

Each record type is represented by a diagram such as the following:



**Figure F-1**  
Record type prototype

The first box illustrates the record. Each block represents a byte. The first byte indicates the record type, in this case, 10. The *flags* byte is expanded in the second box. The *record size* is a signed, 16-bit integer that indicates the total length of the record (including the record type byte, flags byte, and record size field). Hence, any one object file record is limited to 32K bytes. (This is not a limit on the size of the module, because partial contents can be placed in several records.)

The second box represents the flag bits. In this example, they are interpreted as follows:

| Bit  | Meaning                                                          |
|------|------------------------------------------------------------------|
| 0    | 0 indicates code, and 1 indicates data                           |
| 1, 2 | Must always be 0                                                 |
| 3    | 0 indicates short and 1 indicates long                           |
| 4-5  | 0 indicates 32 bits, 1 indicates 16 bits, and 2 indicates 8 bits |
| 6    | Always 0                                                         |
| 7    | 1 indicates a difference computation                             |

❖ *Note:* All unspecified bits must be zero.

---

---

## Object file records

This section defines each of the object file record types.

---

### Pad record

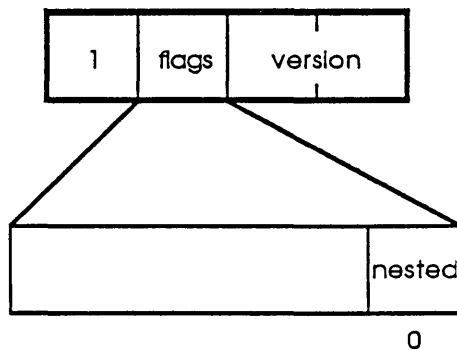


**Figure F-2**  
Pad record

A pad record is a single byte that is always zero. A pad record follows any record whose length is an odd number of bytes, in order to maintain word alignment. (Other than pad records, all records are word-aligned.)

---

### First record



**Figure F-3**  
First record

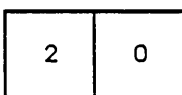
The first record in an object file must be a first record.

If the *nested* bit in the *flags* field is one, then the Linker interprets all references to undefined ID-name pairs as external references. If the nested bit is zero, the Linker will try to match the name of an undefined symbol with a local name before treating the undefined symbol as external.

The *version* field contains a version number that is 1 for the current definition of the object file format.

---

### Last record

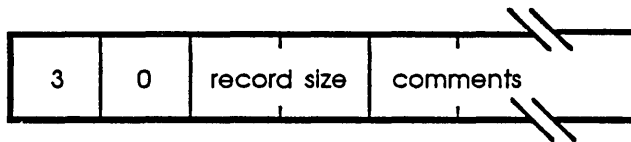


**Figure F-4**  
Last record

The last record in an object file must be a last record.

---

## Comment record



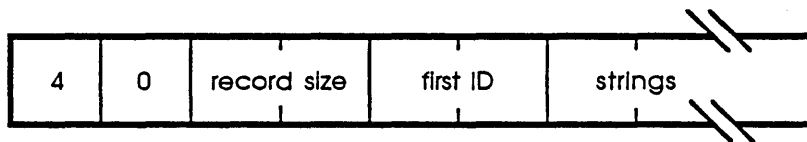
**Figure F-5**  
Comment record

A comment record allows comments to be included in an object file. It has no effect on the semantics of the object file.

The *record size* field specifies the total number of bytes in the record.

---

## Dictionary record



**Figure F-6**  
Dictionary record

A dictionary record associates a name with an ID (or several names with several IDs). At most one dictionary record may appear for a given ID in a single object file.

The *record size* field specifies the total number of bytes in the record.

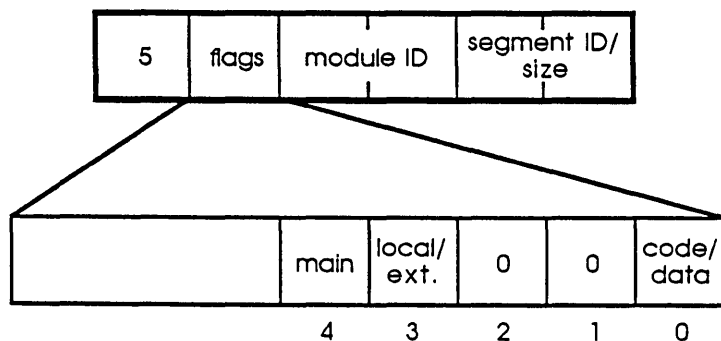
The *strings* field contains one or more names, each of which is preceded by a length byte.

The first name in the strings field is associated with the ID given in the *first ID* field. The second name is associated with *first ID*+1, and so on.

The dictionary record for an ID must appear before the module or entry-point record that defines the ID, but need not appear before reference or computed-reference records that refer to the ID. If an ID has no dictionary record or has a name with a length of zero, it's considered anonymous.

---

## Module record



**Figure F-7**  
Module record

❖ *Note:* Bit 7 now means "Force Active."



A module record associates an ID with a module, and makes that module the current code or data module. All entry-point, size, contents, reference and computed-reference records help define the current code or data module.

Modules may contain either *code* or *data*:

- For code modules, the *segment ID* field specifies the segment in which the code is placed. Segments may be named or anonymous. Named segments are treated as external; anonymous segments are local. (If the segment is named, the dictionary record specifying the name must appear before the segment ID can be used in a module record.)
- For data modules, a nonzero *size* field specifies the size of the module. In this case size or contents records are unnecessary. (The size of a module can also be specified by a size record, or implicitly by the offset of the last byte in a contents record.)

Modules may be either *local* or *external*. (Local modules may be anonymous.)

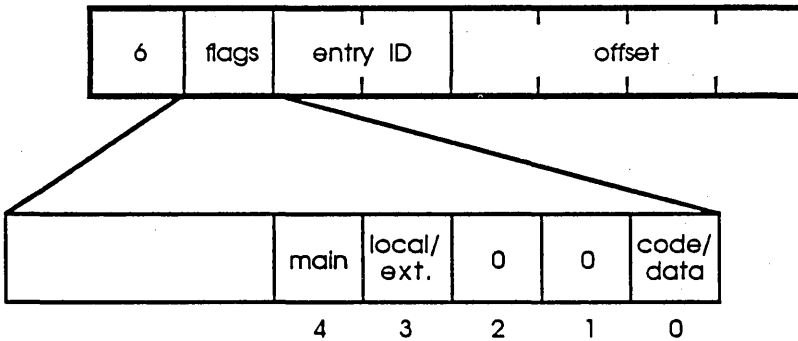
A code module flagged as *main* becomes the execution starting point of the program. A data module flagged as *main* becomes the main program data area, just below the location pointed to by A5. At most one main code module or entry point and one main data module may appear in an object file.

If a code or data module has the *force active* bit set, then the Linker will not strip that module, even though it is not referenced by any other module and even though it is not the main module.

References to a module are considered to be references to the first byte of the module.

---

### Entry-point record



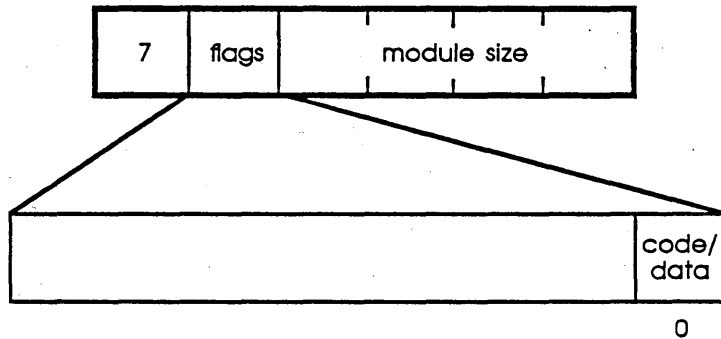
**Figure F-8**  
Entry-point record

An entry-point record declares an entry-point ID. The entry point is in the current code or data module, as indicated by bit 0 of the flags field.

The *offset* field gives the byte offset of the entry point relative to the beginning of the module. The offset of an entry point may be outside the module (for example, a virtual base for an array).

*Flags:* An entry point may be defined for either a code or a data module. Entry points may be either local or external. (Local entry points may be anonymous.) A code entry point flagged as *main* becomes the execution starting point of the program. At most one main code module or entry point may appear in an object file.

## Size record

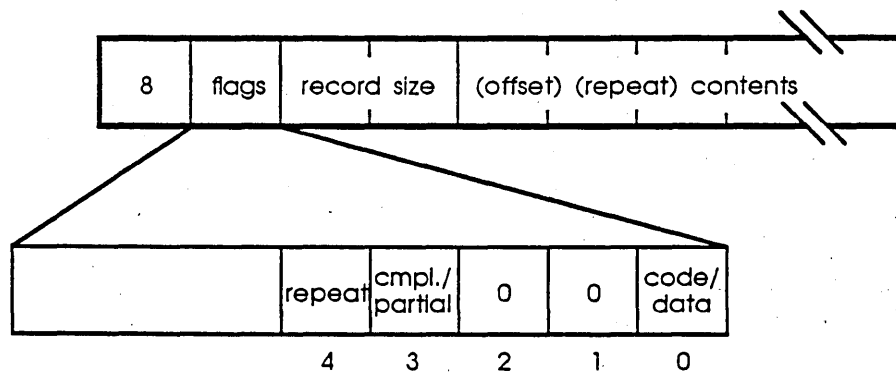


**Figure F-9**  
Size record

A size record specifies the size of the current code or data module. The size is in bytes. The bytes within a module of size  $N$  are numbered  $0, 1, \dots, N-1$ . The size of a module may also be specified in a contents record, or (for data modules) in the module record. If more than one size is specified, the largest size given is taken as the size of the module.

❖ *Note:* In allocating records, the Linker rounds the size of a module up to a multiple of two, to ensure that modules are word-aligned in memory.

## Contents record



**Figure F-10**  
Contents record

Contents records specify the contents of the current code or data module.

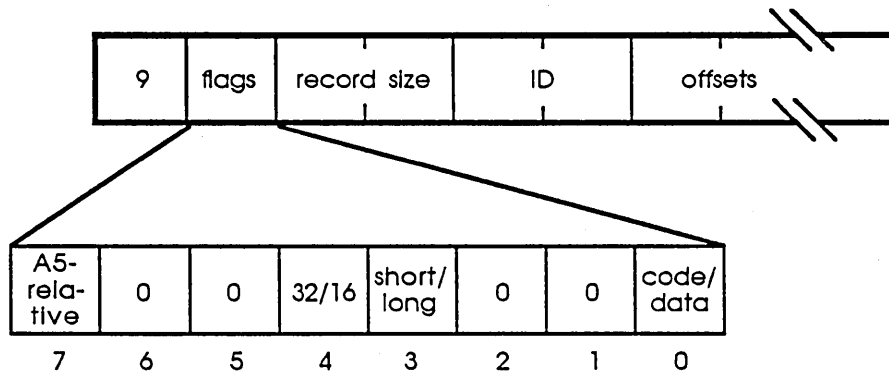
The *record size* field specifies the total number of bytes in the record.

Either *complete* or *partial* contents may be specified. If partial contents are specified, the first four bytes of the contents field specify the *byte offset* of the contents from the beginning of the module.

The *contents* may be either the bytes to be placed in the module, or a 2-byte *repeat count* followed by the bytes to be repeated. (If both an offset and a repeat count are specified, the offset comes first.)

Multiple contents records per module are permitted, in any order. The offset of the last byte for which contents are specified determines the module's total size. (Size specifications may also appear in the module record, and in size records—if more than one size is specified, the largest size given is taken as the size of the module.)

## Reference record



**Figure F-11**  
Reference record

A reference record specifies a list of references to an ID. The references are from the current code or data module, and may be to either code or data.

The *record size* field specifies the total number of bytes in the record.

The *ID* field specifies the module or entry point being referenced.

The *offsets* field specifies a list of byte offsets from the beginning of the current code or data module. These offsets may be either *short* (16 bits) or *long* (32 bits). The location modified may be either 32 or 16 bits. Multiple references to the same or overlapping locations are permitted. References from code may indicate instruction editing (that is, whether an offset is A5- or PC-relative).

References fall into four categories: from code to code, from code to data, from data to code, and from data to data.

- **Code-to-code references:** If the *A5-relative* flag is 1, the A5-relative offset of a jump-table entry associated with the specified module or entry is added to the specified location. No instruction editing is performed.

If the A5-relative flag is 0, the Linker selects either PC-relative or A5-relative addressing. The immediately preceding 16-bit word is assumed to contain a JSR, JMP, LEA, or PEA instruction, and is modified to indicate either PC-relative or A5-relative addressing. If the referenced module or entry point and the current code module are in the same segment, the PC-relative offset of the module or entry point is added to the contents of the specified location. If they are in different segments, the A5-relative offset of a jump-table entry associated with the specified module or entry is added to the specified location.

In either case, the location may be 32 or 16 bits. (32-bit PC-relative and A5-relative address modes are available for the MC68020, but not for the MC68000.)

*Note:* To indicate a PC-relative code-to-code reference, a compiler or assembler should use a computed-reference record.

- **Code-to-data references:** The A5-relative flag must be 1 for code-to-data references. The A5-relative offset of the specified data module or entry is added to the contents of the specified location. No instruction editing is performed. The location may be either 32 or 16 bits. (32-bit A5-relative addressing is available for the MC68020, but not for the MC68000.)

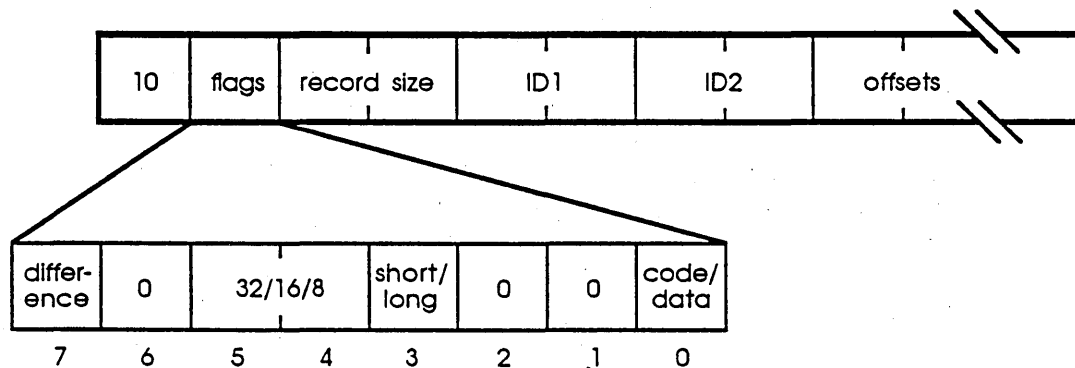
- **Data-to-code references:** If the A5-relative flag is 1, the A5-relative offset of a jump-table entry is added to the specified location, which may be either 32 or 16 bits.

If the A5-relative flag is 0, the memory address of a jump-table entry associated with the specified module or entry is added to the contents of the specified location, which must be 32 bits. (Note that this requires a runtime operation that adds the actual value of A5 to the A5-relative offset.)

- **Data-to-data references:** If the A5-relative flag is 1, the A5-relative offset of the module or entry is added to the specified location, which may be either 32 or 16 bits.

If the A5-relative flag is 0, the memory address of the specified module or entry is added to the contents of the specified location, which must be 32 bits. (Note that this requires a runtime operation that adds the actual value of A5 to the A5-relative offset.)

## Computed-reference record



**Figure F-12**  
Computed-reference record

A computed-reference record specifies a list of computed references based on two specified IDs.

The *record size* field specifies the total number of bytes in the record. The references are from the current code or data module, and may be to either code or data.

The *ID1* and *ID2* fields specify the modules or entry points being referenced. If *ID1* specifies a code reference, *ID2* must also be a code reference in the same segment—if *ID1* is a data reference, *ID2* must also be a data reference.

The only computation provided is *difference*.

The *offsets* field specifies a list of byte offsets from the beginning of the current code or data module. These offsets may be either *short* (16 bits) or *long* (32 bits). The location modified may be either 32, 16, or 8 bits. A 0 in bits 4 and 5 indicates 32; 1 indicates 16; and 2 indicates 8.

The value of the address of *ID1* minus the address of *ID2* is added to the contents of the specified location. Multiple references to the same or overlapping locations are permitted.







## Appendix G



# In Case of Emergency

This appendix contains some information that may be useful when serious system errors occur.

---

---

### Crashes

If you end up in the debugger (MacsBug) while running MPW, it may be possible to recover without rebooting and losing your recent changes. The debugger displays the register contents followed by a ">" prompt. Type `G SYSRECOVER`. The Shell will attempt to recover by aborting the current command, saving the contents of all the windows, and/or returning to the Finder. If this fails, type `ES` to return to the Finder, then shut down the system immediately.

---

---

### Stack space

The MPW Shell and tools that run integrated with the Shell share a single stack. The stack size is determined by the Shell at initialization time. Complex command files, large links, and other tools may require more stack space than is available. System errors 28, 2, and 3 are possible indications of this problem. You can increase the stack size by using ResEdit to modify 'HEXA' resource number 128 in the file MPW Shell. The default size is \$2710 (10,000 bytes) when less than 480,000 bytes are available for the application heap and \$4E20 (20,000 bytes) when more than 480K are available.







## Glossary

**active window:** The frontmost window. The Shell variable {Active} always contains the name of the current active window.

**alias:** An alternate name for a command, defined with the Alias command.

**application:** A program that runs stand-alone, outside of the Shell environment. An application's file type is APPL.

**blank:** A space or a tab character (in the context of separating words in the command language).

**build commands:** Shell commands that are output by the Make tool, used to build a program.

**build command line:** In the dependency rule of a makefile, the lines beginning with a space or tab that follow the dependency line.

**built-in commands:** Editing commands, structured commands, and other Shell commands that are part of the MPW Shell application (as opposed to *MPW tools*, which are separate files on the disk.)

**code resource:** A **resource** that contains a program's code—most commonly a resource of type 'CODE' (for applications and MPW tools), but other resource types such as 'DRVr' and 'PDEF' also contain code.

**command file:** See **script**.

**command name:** The first word of a command, identifying the name of a built-in command or the name of a file (tool, command file, or application) to execute.

**command script:** See **script**.

**command substitution:** The replacement of a command by its output. Command substitution takes place within back quotes (``...``).

**console:** The window where a command is entered and executed (standard input). Also, the window to which the command's output is returned (standard output).

**current selection:** The currently selected text in a window. In editing commands, the current selection in the target window is represented by the `$` metacharacter.

**data fork:** The part of a file that contains data accessed via the Macintosh File Manager.

**data initialization interpreter:** The module `_DATAINIT` in the libraries `RunTime.o` and `CRuntime.o`.

**dependency file:** A makefile.

**dependency line:** In Make, the first line of a **dependency rule**.

**dependency rule:** In Make, a rule that specifies the prerequisite files of a given target file, along with a list of the commands needed to build the target file.

**dependent:** In a Commando dialog, a control that is enabled or disabled depending on the state of its parent control.

**desk accessory:** A "mini-application," implemented as a device driver, that can be run at the same time as an application. Desk accessories are files of type DFIL and creator DMOV, and are installed by using the Font/DA Mover.

**device driver:** A program that controls the exchange of information between an application and a device.

**diagnostic output:** Commands and tools send error and progress output to diagnostic output (by default, the window where the command was executed). You can redirect diagnostic output to another file, window, or selection with the `>` and `>>` operators.

**dialog:** In Commando, the programmed interaction between a user and a tool or script. A dialog may utilize more than one dialog box.

**dialog box:** A window that appears when a command is invoked, offering options and parameters.

**Editor:** When appearing with initial capitals, the built-in commands appearing in MPW's Edit menu, a part of the MPW Shell.

**entry point:** A location (offset) within a **module**.

**escape character:** The Shell escape character is  $\partial$  (Option-D). It is used to disable (or "escape") the special meaning of the character following it, to continue commands over more than one line ( $\partial$ Return), and to insert invisible characters into command text.

**external:** A *module*, *entry point*, or *segment* that can be referenced from different object files.

**external reference:** A reference to a routine or variable defined in a separate compilation or assembly.

**filename:** A sequence of up to 31 printing characters (excluding colons), that identifies a file. See also **pathname**.

**file type:** A four-character sequence, specified when a file is created, that identifies the type of file. (Examples: TEXT, APPL, MPST.)

**Finder information:** Information that the Finder provides to an application upon starting it, telling it which documents to open or print.

**Font/DA Mover:** An application, available on the *System Tools* disk, used for installing desk accessories in the System file.

**full pathname:** A pathname that contains embedded colons but no leading colon.

**HFS:** "Hierarchical File System," used on 800K disks and the Apple Hard Disks.

**ID:** A file-relative number for a module, an entry point, or a segment, identifying the module, entry point, or segment within a single object file.

**insertion point:** An empty selection range; that is, the character position where text will be inserted (marked with a blinking vertical bar).

**Integrated Environment:** A set of routines, modeled on the C language, that provide parameter passing, access to variables, and other functions to MPW tools. (See Chapter 13.)

**interface routine:** A routine called from Pascal whose purpose is to trap to a certain ROM or library routine.

**jump table:** A table that contains one entry for every routine in an application or MPW tool, and is the means by which the loading and unloading of segments is implemented.

**leafname:** A partial pathname that contains no colons.

**literal:** In the Resource Compiler, a value within single quotation marks. (See Chapter 8.)

**local:** A *module*, *entry point*, or *segment* that can be referenced only from within the file where it is defined.

**location map:** The Linker can write to standard output a map of memory segments sorted by *segNum* and *segOffset*. (See Chapter 10.)

**main segment:** The segment containing the main program.

**makefile:** A file used by the Make command, which describes dependencies between the various pieces of a program, and contains a set of commands for building up-to-date files. The default makefile is named MakeFile.

**module:** A contiguous region of memory that contains code or static data. A module is the smallest unit of memory that is included or removed by the Linker.

**MPW Shell:** The application that provides the environment within which the other parts of the Macintosh Programmer's Workshop operate. The Shell combines an editor, command interpreter, and built-in commands.

**MPW tool:** An executable program (type MPST) that is integrated with the MPW Shell environment (contrasted with an application, which runs stand-alone). Like applications, tools exist as separate programs on the disk.

**non-HFS:** The nonhierarchical file system, used on 400K disks and Macintosh XL hard disks.

**option:** A command-line switch, specifying some variation from a command's default behavior. Options always begin with a hyphen (-).

**parameter:** The words following the keyword in a simple command. There are two types of parameters: options and files. Note that certain parameters, such as I/O redirection, are interpreted by the Shell and never seen by the command itself.

**parent:** In Commando, an option or control whose status determines whether a dependent option or control is enabled or disabled.

**partial pathname:** A pathname that either contains no colons or has a leading colon.

**pathname:** A sequence of up to 255 characters that identifies a file or directory. A *full pathname* is a pathname that contains embedded colons but no leading colon. A *partial pathname* either contains no colons or has a leading colon. A *leafname* is a partial pathname that contains no colons.

**pattern:** A literal text pattern (such as /ABCDEFGG/), or a regular expression. Patterns are a case of selection, and always appear between the pattern delimiters /.../ or \...\.

**pipe:** The command terminator | is the pipe (or pipeline) symbol. It causes the output of the preceding command to be used as the input for the subsequent command. (See Chapter 5, Table 5-1.)

**position:** In editing commands, position refers to the location of the insertion point.

**prefix:** The directory portion of a filename.

**prerequisite file:** In Make, the files that must exist or be up-to-date before the target file can be built.

**pseudo-filename:** Any device name that you can use in place of a filename, but that has no disk file associated with it. Any command can open a pseudo-filename. These are most often used for I/O redirection.

**quotes:** A set of characters that literalize the enclosed characters, used for disabling special characters. The quote symbols are '...', "...", \...\, and /.../. The escape character, \, quotes the character that follows it.

**reference:** The location within one module that contains the address of another module or entry.

**regular expressions:** A language for specifying text patterns, using a special set of metacharacters. (See Appendix B, Table B-2.)

**regular expression operators:** A special set of metacharacters used in regular expressions and filename generation. (See "Pattern Matching" in Chapter 6.)

**resource:** Data or code stored in a resource file and managed by the Macintosh Resource Manager.

**resource attribute:** One of several characteristics, specified by bits in a resource reference, that determine how the resource should be dealt with.

**resource compiler:** A program that creates resources from a textual description. The MPW Resource Compiler is named Rez.

**resource description file:** A text file that can be read by the Resource Compiler and compiled into a resource file. The Resource Decompiler disassembles a resource file, producing a resource description file as output.

**resource file:** Common usage for the resource fork of a Macintosh file.

**resource fork:** The part of a file that contains data used by an application, such as menus, fonts, and icons. An executable file's code is also stored in the resource fork.

**root:** In a makefile, a top-level target that is not a prerequisite of any other target.

**script:** An ordinary text file (type TEXT) containing a series of commands. The entire file can be executed by entering the filename. A script is also referred to as a *command file* or *command script*.

**segment:** One of several parts into which the code of an application may be divided. Not all segments need to be in memory at the same time.

**selection:** A series of characters, or a character position, at which the next editing operation will occur. Selected characters are inversely highlighted in the active window, and outlined in other windows. A *selection* is used as an argument to most editing commands, and can be specified by using a special set of selection operators. (See Appendix B, Table B-1.)

**signal:** An event that diverts program control from its normal sequence. (See Chapter 13.)

**signature:** Each Macintosh application has its own unique signature (or creator). For example, creating a file with the type DFIL and signature DMOV tells the Font/DA Mover that this file contains desk accessories. See the Finder Interface chapter of *Inside Macintosh*.

**simple command:** Any command consisting of a single keyword followed by zero or more parameters.

**standard error:** Diagnostic output.

**standard input:** Input that is typed directly into a window (the console).

**standard output:** Output that is returned to the window where its command or program was typed.

**Startup file:** A special command file containing commands that are executed each time the Shell is launched. Startup executes a second command file called UserStartup.

**status panel:** The panel in the lower left corner of the Worksheet window. The status panel shows what command MPW is executing. Clicking in the status panel is equivalent to pressing the Enter key.

**status value:** A code returned by commands in the Shell variable {Status}. Zero indicates successful completion of the previous command, and other values usually indicate an error.

**structured command:** Any command that controls the order in which other commands are executed. For and If are examples of structured commands. All structured commands are built into MPW and usually have more than one keyword. See also **simple command**.

**target selection:** The current selection in the target window, represented by the § character.

**target file:** In Make, a file that is to be rebuilt and that depends on one or more prerequisite files.

**target window:** The second window from the front—this is the default target for editing commands that are entered in the active window. The Shell variable {Target} always contains the name of the current target window.

**tool:** See **MPW Tool**.

**type declaration:** In the Resource Compiler, a statement that specifies the pattern for any associated resource data by indicating data types, alignment, size, and placement of strings.

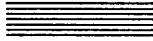
**user interface:** The system or set of conventions by which the user interacts with software. In addition to the standard Macintosh mouse-and-menu interface, MPW includes both a command language and a dialog user interface (Commando).

**word:** A single, blank-separated element in a command. A command name and each of its parameters are separate words in the command language.

**Worksheet window:** The main work area in MPW; the window usually used as the console.



# Index



## A

- (Active) 5-10
- active window 4-2
- AddMenu 2-4, 3-16, 4-16, 4-22, 5-17, 5-32-33
- address operator (&) 11-8, 11-20
- Address Register 11-10
- Adjust 6-2
- {AIncludes} 5-12
- Alert 5-27
- aliases 4-12, 5-8-9
- {Aliases} 5-10
- Align 3-7, 6-2
- align 8-10, 8-13
- align types 8-13
- appheap 8-8
- Apple menu 3-3
- Application A-trap Trace 11-15
- applications 1-9, 5-2
  - Macintosh 9-4
  - running outside Shell 5-6
- array 8-10, 8-14
- array types 8-14
- Asm 2-2, 5-2
- Assembler. *See* MPW Assembler
- assembly language, using
  - MacBug 11-5
- asterisk (\*) 6-12, 11-8, 11-17
- A-trap Break 11-15
- A-trap Clear 11-16
- A-trap commands, MacsBug 11-14-16
- A-trap Heap Zone Check 11-15
- A-trap Record 11-15
- A-trap Spy 11-16
- A-trap Trace 11-15
- at sign (@) 11-8
- \$\$Attributes 8-23
- Auto Indent 3-7

## B

- back quotes (``) 5-2, 5-15
- backslash character (\) 8-19, 8-24, 8-25
- Backup 4-4, 4-7, 10-13

- Begin...End 5-19
- bitstring 8-10, 8-11
- blanks 5-4
- boolean 8-10, 8-11
- Boolean type 8-11
- {Boot} 5-10, 10-13
- braces ({} ) 8-15
- Break 5-20, 5-21, 5-23, 5-24, 11-12
- break commands, MacsBug 11-12-13
- Break in Application 11-15
- Bring to Front 7-9
- Build 2-10
- Build... 3-16
- build command lines 10-3
- BuildCommands 9-11
- Build item 3-16
- BuildMenu 9-11
- Build menu 2-4, 2-6-10, 3-15-16
  - modifying 9-11-12
- BuildProgram 9-11
- built-in commands 1-2, 1-4, 2-4, 5-2
- byte 8-10, 8-11

## C

- Canon 1-8, 6-2
- Canon.dict 1-8
- case Box 14-9
- case List 14-14
- Case Sensitive 3-9
- {CaseSensitive} 5-12
- case TextBox 14-10
- Catenate 4-7, 5-28
- C Compiler 1-7, 5-12
- changed 8-8
- char 8-10, 8-11
- character editing panel 7-10
- character selection panel 7-10
- character types 8-11
- check boxes 4-16
  - Commando 14-6-7
- CheckOption 14-6-7
- Checksum 11-11
- Chooser Desk Accessory 3-5
- {CIncludes} 5-12
- Clear 3-6, 6-2, 7-5, 7-7, 11-12
- {CLibraries} 5-12
- Clipboard 3-6
- Close 3-4, 7-3, 7-5, 7-6
- Cmdo.r 14-3
- code resources 12-11
- code-to-code references F-7
- code-to-date references F-7
- colon (:) 4-8, 4-9, 6-7, 10-6
- comma (,) 8-6
- {Command} 5-10
- Command-[ 3-7
- Command-] 3-7
- Command-A 3-6
- Command-C 3-6
- Command-D 3-4, 4-2
- Command-Enter, terminating input 5-28
- Command-F 3-8
- command files. *See* scripts
- Command-G 3-8
- Command-H 3-8
- command interpreter 1-2, 3-2
  - negative status values 5-3-4
- command language
  - editing with 5-31
  - features 5-2
  - MacsBug 11-7-8
- Command Line 4-14
- Command-N 3-4
- command name 5-4
- Command-O 3-4
- Commando 1-8, 4-2, 9-11
  - boxes, lines, and text titles 14-9-10
  - check boxes 14-6-7
  - control dependencies 14-16-20
  - files and directories 14-23-30
  - icons and pictures 14-16
  - lists 14-14
  - multiregular entry 14-5-6
  - pop-up menus 14-11-13
  - radio buttons 14-7-8
  - redirection 14-22-23

- regular entry 14-5
- resource description 14-3-4
- screen elements 14-4
- three-state buttons 14-15
- using 14-2
- Commando 4-13
- {Commando} 5-11
- Commando dialogs 4-13-23
  - box controls 4-15-20
  - creating 14-1
  - invoking 4-13
  - using 4-14
- Command-P 2-6, 3-16
- Command-period 4-5, 4-14
- Command-Q 3-5
- Command-R 3-8
- Command-Return 4-3
- Command-S 3-4
- commands. *See also specific command*
  - aliases 5-8-9
  - built-in 1-2, 1-4, 2-4
  - comments 5-6
  - editing 1-3, 6-2-9
  - entering and executing 5-3-4
  - executing 4-4
  - file management 1-3, 4-7-8
  - interpretation 5-18
  - MacsBug 11-9-20
  - MPW Shell 3-3-16
  - options 4-3
  - parameters 5-4, 5-22
  - simple 5-6
  - status values 5-3-4
  - structure 5-4-6
  - structured 1-4, 5-6, 5-19-24
  - terminating 4-5
  - terminators 5-5
  - types 5-2-3
  - window 1-2
- {Commands} 5-11
- command scripts. *See scripts*
- command search path 4-11
- command substitution 5-15
- Command-T 3-8
- Command-U 5-32
- Command-V 3-6
- Command-W 3-4
- Command-X 3-6
- Command-Y 3-7
- Command-Z 3-6
- comment record F-2, F-4
- comments 8-4
  - command 5-6
  - makefile 10-9
- compound types D-5
- computed-reference record F-1, F-8
- conditional execution operators (&& and ||) 5-5
- console 4-3
- contents record F-2, F-6
- Continue 5-20, 5-23, 5-24
- control dependencies 14-16-20
  - direct 14-17
  - Do It button 14-19
  - inverse 14-18
  - multiple 14-19
  - radio buttons 14-20
- control loops 5-21
- Convert 11-10
- Copy 3-6, 6-2, 7-5, 7-7
- Count 4-9, 5-5
- \$\$countof 8-14
- crashes G-1
- Create Build Commands 2-8, 2-9, 2-10, 9-12
- Create Build Commands... 3-15-16, 9-11
- CreateMake 9-11, 9-12
- CRuntime.o 9-6
- cstring 8-10, 8-12
- current selection 4-9
- current selection character (Ⓢ) 4-9, 6-5-6
- cursor 7-8
- Cursor menu 7-8
- CURS' resources 7-8
- Cut 3-6, 6-2, 7-5, 7-7

**D**

- data 8-4, 8-9, D-2, D-3
- data fork 9-4
- data initialization interpreter 10-15
- Data → Mask 7-8
- Data Register 11-10
- data-to-code references F-8
- data-to-data references F-8
- data-type specifications 8-10-13
- \$\$Date 8-23
- \$\$Day 8-23
- debugger 1-9
- Debugger Exchange 11-13
- debugging. *See also MacsBug*
  - makefiles 10-10-11
- declaration files, standard type 8-3
- Default Input 4-19
- Default Output 4-19
- default rules 10-5-6
- #define 8-19, D-5
- Defines window 14-6
- Delete 4-7
- DeleteMenu 5-32
- dependency line 10-3
- dependency rules 10-3-7, 10-13
  - default 10-5-6
  - directory 10-6-7
  - double-f 10-4
- DeRez 1-8, 7-2, 8-2
  - Rez and 8-2
  - using 8-3-4
- DeRez.Out 8-5
- desk accessory
  - building 9-7-10
  - linking 9-9
  - resource file 9-10
- DeskTop 7-11
- diagnostic output 5-29
- dialog 1-2, 4-13
- dialog box 1-2, 4-2
  - Commando 14-3
  - CreateMake 2-9, 3-15
  - Date 4-14
  - Find 3-8
  - Find-and-Replace 3-8
  - Format 3-7
  - Mark 3-12
  - nested 4-20-21, 14-21-22
  - New 3-4
  - Open 3-4
  - Page Setup 3-5
  - Print 3-5
  - Program Name? 2-7, 3-16
  - Rez 4-20-21
  - Save As 3-4
  - Set Directory... 3-14
  - standard file 2-9, 3-2
  - Unmark 3-13
- dialog box controls
  - Commando 4-15-20
  - special 4-20-23
- Dialog Manager 14-9
- dictionary record F-1, F-4
- directories 4-9-11
  - changing 4-12
  - Commando 14-23-30
  - dependency rules 10-6-7
  - multiple 4-18
  - new 4-20
- Directory 4-7, 4-12
- Directory... 4-19
- DirectoryMenu 3-13, 9-11
- Directory menu 2-4, 2-5-6, 2-8-9, 3-13-14, 9-11
- DirectoryName 3-14
- Disassemble Hex 11-19
- disassembler commands, MacsBug 11-18-20
- Disk info window 7-4
- Disk volume window 7-3
- Display Byte 11-10
- Display Memory 11-10
- Display Selection 3-8
- Display Version 11-9

'DITL' editor 7-7  
 'DITL' resources 7-8-9  
 Do It button dependency 14-19  
 dollar sign (\$) 11-7  
 dot symbol (.) 11-10, 11-17, 11-19  
 double colons (::) 4-9  
 double quotation marks (") 8-24  
 driver  
   building 9-7-10  
   linking 9-9  
   writing 15-3  
 DRVRRuntime 9-7  
 DRVRRuntime library 15-2-3  
 DRVRRuntime.o 9-9, 15-2  
 DumpObj 6-14, 10-20  
 Duplicate 4-7, 5-2, 7-5, 7-7

## E

Echo 5-22, 5-25  
 {Echo} 5-11  
 editing commands 1-3, 6-2-3  
   ^ selections 6-3-9  
 Edit menu 2-4, 3-6-7, 7-7  
 editor 1-2, 3-8  
 #elif 8-20, D-5  
 ellipsis (...) 4-5, 4-13, 14-2  
 #endif D-5  
 Entab 1-8, 6-2  
 Entire Word 3-9  
 entry point 10-17, F-1  
 entry-point record F-2, F-5  
 error messages, executable 5-9  
 Error Output 4-21  
 escape character (ð) 6-11, 6-13, 8-25  
 escape conventions 5-17  
 Evaluate 5-24, 6-9  
 Event Manager 11-3  
 events, MPW Shell 13-10  
 Execute 5-14  
 Exists 4-7  
 Exit 5-20, 5-21, 5-23, 5-24  
 {Exit} 5-11  
 Exit to Application 11-9  
 ExitToShell 11-9  
 Exit to Shell 11-9  
 Export 5-14  
 expression operators 5-23-24  
 expressions 8-22, D-5-6  
   MacBug 11-8  
 external module 10-17, F-1

## F

FatBits 7-10  
 file creators 9-7, E-1  
 file management commands 1-3, 4-7-8

File menu 2-4, 3-3-5, 7-3, 7-5, 7-7  
 filename generation operators 5-25  
 filenames 4-8-13  
   pseudo- 5-30  
 files 5-4. *See also* input files;  
   output files  
     Commando 14-23-30  
     suffixes E-1-2  
     types 9-7, E-1  
 Files 2-10, 3-15, 4-7, 4-11, 5-2, 5-5, 14-23  
 fill 8-10, 8-13  
 fill types 8-13  
 Find 3-10, 3-11, 6-2, 6-3, 6-7, 6-14, 11-19  
 Find... 3-8  
 Find-and-Replace 3-8  
 Finder 1-9, 2-2, 2-3, 3-2, 7-3, 11-9  
 Find menu 2-4, 3-8-11  
 Find Same 3-8  
 Find Selection 3-8  
 first record F-1, F-3  
 Floating Control Register 11-11  
 Floating Data Register 11-11  
 Floating Instruction Address Register 11-11  
 Floating Status Register 11-11  
 Font 6-2  
 Font/DA Mover 1-9, 2-8, 9-7, 9-10  
 'FONT' editor window 7-9  
 Font Manager 11-3  
 'FONT' resources 7-9-10  
 font size control, dependent 14-16  
 Font Size pop-up menu 14-13  
 For 5-21  
 For... 5-19  
 For...End 5-21  
 Format... 3-7  
 \$\$Format 8-20, 8-23  
 Full Build 2-7, 2-8, 2-10  
 Full Build... 3-16  
 full pathname 4-9

## G

general data editor 7-7  
 GetFileName 4-7  
 Get Info 7-4, 7-6, 7-10  
 Get Info window 7-6  
 Go 11-12  
 Go Till 11-12  
 graphics, MPW Shell 13-10  
 graphic tools panel 7-10  
 Grid 7-9

## H

hard disk  
   configuration files A-7-12  
   installing MPW 2-2  
 heap, MPW Shell 13-10  
 Heap Check 11-16  
 Heap Dump 11-17  
 Heap Exchange 11-16  
 Heap Scramble 11-16  
 Heap Total 11-17  
 heap zone commands,  
   MacBug 11-16-17  
 Help 4-5-6  
 hex string 8-24  
 hierarchical directory structure 4-10  
 hierarchical file system 4-9, 5-11  
 \$\$Hour 8-23

## I

IAZNotify 11-9  
 'ICN#' resources 7-11  
 ID F-1  
 \$\$ID 8-23  
 identifiers D-5  
 If 5-23, 5-24, 6-9  
 If... 5-19  
 #if 8-20, D-5  
 #ifdef D-5  
 #ifndef D-5  
 include 8-4, 8-7-8, 10-5, D-2  
 #include 8-16, 8-19  
 infinity character (∞) 5-31  
 InitApplZone 11-9  
 InitGraf 13-8  
 initialization, MPW Shell 13-8  
 InitPerf 12-3  
 input  
   redirecting 5-26-30  
     standard 4-3, 5-26, 5-27-28  
 Input File... 4-19  
 input files  
   multiple 4-17  
   single 4-19  
 insertion point 6-7  
 Instruction Disassemble 11-19  
 Instruction List 11-19  
 integer 8-10, 8-11, 8-15  
 Integrated Environment 13-2  
 interrupt handler, performance tools  
   and 12-10  
 invisible characters 6-13  
 I/O buffering, MPW Shell 13-5-6  
 I/O channels, MPW Shell 13-4-6  
 IOInterfaces 10-13  
 itemName 4-23

## L

last record F-1, F-3  
leafname 4-9  
left brace (()) 8-15  
less-than symbol (<) 11-7, 11-8  
{Libraries} 5-12  
library construction 10-19-21  
{Libs} 10-13  
Lib tool  
  -**df** option 10-14, 10-20  
  library construction 10-19-21  
Line 6-2  
line continuation character 10-8  
line number, selection by 6-6  
Link 4-19, 5-2, 9-2  
  -**sg** option 10-14, 10-15  
  -**sn** option 10-14, 10-15  
Linker 1-7, 9-5-6, 10-13, F-3  
  controlling code resources 10-17  
  functions, 10-14-15  
  -**lf** option 10-18  
  location map 10-18  
  -**l** option 10-18  
  -**ma** option 10-17  
  optimizing 10-19  
  RAM caches 10-19  
  -**ra** option 10-16  
  resolving symbol definitions  
    10-17  
  -**rt** option 9-9  
  segmentation 10-15-16  
  setting resource attributes 10-16  
  -**sg** option 9-9  
  -**sn** option 9-9  
  -**uf** option 10-20  
  -**w** option 9-6  
linking 9-5-6, 10-14-19  
  desk accessory 9-9  
  driver 9-9  
  MPW tool 9-13  
{LinkOpts} 10-13  
Listings 10-13, 10-14  
Literal 3-9  
literals 8-21  
local module 10-17, F-1  
locked 8-8  
logical operators 5-24  
longint 8-10, 8-11  
Loop 5-21  
Loop...End 5-20

## M

MacDraw 1-9  
MacInterfaces 10-13  
Macintosh application, structure 9-4  
Macintosh Programmer's Workshop.  
  See MPW  
MacPaint 5-2, 5-6, 7-10

## MacsBug 1-9

A-trap commands 11-14-16  
boot code 11-3  
break commands 11-12-13  
command language 11-7-8  
command summary 11-20-21  
disassembler commands  
  11-18-20  
exceptions 11-4-5  
general commands 11-9  
heap zone commands 11-16-17  
installing 11-2-3  
memory commands 11-10-11  
memory usage 11-4  
using 11-5-6

## MacWrite 3-2

## Magic Return 11-13

## makefile 2-4-5

comments 10-9  
debugging 10-10-11  
format 10-2-3  
modifying 9-12  
quoting 10-8  
variables 10-7-8

## MakeLoad 10-13

{MakeObjs} 10-13, 10-14

## MakeOut 10-9

## Make-p-o 10-13

## Make tool 1-7, 10-2-14

-**d** option 10-7  
-**r** option 10-9  
-**s** option 10-10  
-**u** option 10-10  
-**v** option 10-10

{MakeUses} 10-13

## MakeX 10-13

## Mark 3-2, 6-2, 6-8

## Mark... 3-12

## markers 3-12-13, 6-8-9

## Markers 6-2

## Mark menu 2-4, 3-2, 3-12-13

## MDSCvt 2-2

## MDSCvt.Directives 2-2

## MemMgr 10-13

memory commands, MacsBug  
  11-10-11

memory management, MPW  
  Shell 13-9-10

## Memory Manager 11-3, 12-10

## menu bar 2-4

menu commands, defining 5-32

menuName 4-23

menus. *See specific menu*

metacharacters 6-9, 6-13, 6-14

minus sign (-) 6-11, 11-8

ssMinute 8-23

module record F-2, F-4-5

modules 10-14, F-1

  external 10-17, F-1

  local 10-17, F-1

\$\$Month 8-23

Mount 4-7

Move 4-7

## MPW

  building a program 2-4-10

  commands 2-4

  command scripts 1-5

  distribution files A-1-3

  editing functions 4-2

  entering commands 4-2-6

  files and directories 1-10

  installation 2-2

  operators C-1-3

  RAM caches 2-3

  starting 2-3

{MPW} 4-12, 5-11

MPW Assembler 1-6, 2-2

  files A-3-4

  -**print** option 4-16

## MPW C

  files A-5-7

  using MacsBug 11-6

MPW Pascal 1-6

  files A-4-5

MPW Shell 1-2-4

  built-in commands 1-2, 1-4

  conventions 13-11

  editing commands 1-3

  editing features 3-2

  exit processing 13-7

  file format 3-2

  file management commands 1-3,  
    4-7-8

  icon 2-3

  initialization 13-8

  memory management 13-9-10

  menu commands 3-3-16

  parameters 13-2-3

  running applications outside 5-6

  signal handling 13-6

  standard I/O channels 13-4-6

  status codes 13-7

  structured commands 1-4,  
    5-19-24

  variables 4-12, 5-9-14, 13-4

  window commands 1-2

MPW tools 1-5-8

  building 9-13

  linking 9-13

MPWTypes.r 8-3, 9-10

MultiDirs 14-26

MultiFiles 14-26, 14-27, 14-28,  
  14-30

MultiInputFiles 14-26

MultiInputFilesAndDirs 14-26

MultiOutputFiles 14-26



## N

- \$\$Name 8-23
- negation symbol (~) 6-11
- New 4-7, 7-3, 7-5, 7-6
- New... 3-4
- Newer 4-7
- {NewerDeps} 10-14
- NewFolder 4-7, 4-20
- No Input 4-19
- nonpreload 8-8
- nonpurgeable 8-8
- No Output 4-19
- Notepad desk accessory 4-15
- numbers 5-24, 8-21, D-6
  - MacBug 11-7
- number sign (\*) 4-5, 5-6, 10-9, 11-7
- numeric types 8-11

## O

- object files
  - format F-1-2
  - records F-3-8
  - suffixes E-2
- onexit 13-7
- Open 3-2, 4-2, 4-7, 7-3, 7-5, 7-6
- Open... 3-4
- Open a... 7-6
- Open General 7-5, 7-6
- Open Selection 3-4, 4-2
- Option-5 5-31
- Option-6 5-31, 6-5
- Option-8 6-13
- Option-D 5-5
- Option-F 10-2, 10-3
- Option-J 6-7
- Option key 4-13
- Option-key characters B-3
- Option-L 6-11
- options 5-4
- Option-semicolon 4-13, 4-14, 14-2
- Option-X 3-10, 4-13, 6-11
- output
  - diagnostic 5-29
  - redirecting 5-26-30
  - standard 4-3, 5-26, 5-28
- Output File... 4-19
- output files, single 4-19

## P

- pad record F-1, F-3
- Page Setup... 3-5
- parameters
  - command 5-4, 5-22
  - MPW Shell 13-2-3
- Parameters 5-22, 5-25
- parent 14-16

- parentheses (()) 5-5, 5-26
- Pascal
  - strings 8-24
  - using MacBug 11-6
- Pascal Compiler 5-12
- PasMat 2-2, 4-20
- PasRef 2-2
- Paste 3-6, 6-2, 7-5, 7-7
- pathnames 4-9-11
  - full 4-9
  - variables 4-12
- pattern 6-9
  - matching 6-9-14
- Perf.c 12-2
- PerfControl 12-3, 12-6
- PerfDump 12-2, 12-3, 12-6
- performance measurement tools
  - 1-8
  - using 12-5-6
- performance output file 12-6-8
- performance reports 12-6-10
  - interpreting 12-10
- performance tools
  - bucket counts 12-4
  - components 12-2-3
  - implementation issues 12-10-11
  - using 12-3
- PerformLib.o 12-2
- PerformReport 12-2, 12-3, 12-6
  - analyzing results 12-9
  - e option 12-9
  - interpreting 12-10
- Perf.p 12-2
- PictOrIcon 14-16
- {PInterfaces} 5-12
- pipe symbol (|) 5-5
- {PLibraries} 5-12
- plus sign (+) 6-12, 11-8
- point 8-10
- point types 8-12
- pop-up menus 4-16-17, 14-11-13
  - output file 4-19
- position 6-7
- preload 8-8
- preprocessor directives 8-4, 8-19-20, D-5
- prerequisite files 10-2
- #printf 8-20, D-5
- {PrintOptions} 5-12
- Print Selection 3-5, 5-12
- print tool, header options 4-22
- Print Window 3-5, 5-12
- program
  - building 9-2-4
  - segmenting 10-15-16
- Program Counter 11-11, 12-2
- sampling 12-3-4
- protected 8-8

- pseudo-filenames 5-30
- pstring 8-10, 8-12
- purgeable 8-8

## Q

- question mark (?) 4-13, 11-9
- QuickDraw 13-8
- Quit 1-9, 3-5, 5-8, 7-4, 7-5
- quotation marks (' and ") 5-17, 8-6.
  - See also back quotes
- quoting, makefile 10-8
- quoting special characters 4-8, 5-2, 5-15-17

## R

- radio buttons 2-10, 3-9, 4-15, 14-7-8
  - dependencies 14-20
- Random 13-8
- read 8-4, 8-8, D-2
- Reboot 11-9
- rect 8-10
- rectangle types 8-12
- reference 10-17
- reference record F-1, F-6-7
- regular expressions 1-3, 3-2, 6-9, B-2-3
  - operators 3-10, 5-25, 6-10
  - repeated instances 6-12
  - tagging 6-12-13
- Rename 4-7
- Replace 3-11, 6-2, 6-12
- Replace... 3-8
- Replace Same 3-8
- ResEdit 1-9, 5-2, 9-2
  - customizing 7-12-14
  - extensibility 7-2
  - file window 7-5
  - uses 7-2
  - using 7-3-11
- ResEqual 8-4
- resource, editing 7-7-11
- resource 8-4, 8-5, 8-13, 8-16-18, D-2, D-4
- \$\$resource 8-23, 9-9, 9-10
- resource attributes 8-8
  - setting 10-16
- Resource Compiler. See Rez
- Resource Decompiler. See DeRez
- resource description expression
  - operators 8-22
- resource description file 8-2
  - Commando 14-3-4
  - structure 8-4-5, D-2-4
  - syntax notation D-1
- resource description statements
  - 8-6-18

resource description syntax 8-21-25  
 resource fork 9-4  
 Resource Manager 11-3  
 resource template, creating 7-12-14  
 resource type 7-6-7  
   window 7-6  
 Restore 10-13  
 Resume 1-9, 5-8  
 Revert 4-7, 6-2, 7-5, 7-6  
 Revert to Saved 3-5  
 Rez 1-8, 4-19, 5-12, 7-2, 9-2  
   DeRez and 8-2  
   escape sequences 8-25  
   first dialog box 4-20  
   input file 9-10  
   nested Preprocessor dialog box 4-21  
   nested Redirection dialog box 4-21  
   using 8-3-4  
   variables 8-23  
 Rez command line, **-undef** option 8-19  
 RezDet 1-8  
 right brace character (}) 5-9, 8-15  
 (RIncludes) 5-12  
 roots 10-2, 10-13  
 Runtime.o 9-6

## S

sample text panel 7-10  
 Save 3-4, 4-7  
 Save a Copy... 3-5  
 Save as... 3-4  
 scripts 1-5, 1-9, 5-2, 5-7  
   parameters 5-13  
 scrolling 4-15  
 Search 6-9  
 Search Backward 3-9  
 searching, forward and backward 6-14  
 \$\$\$Second 8-23  
 segmentation  
   Linker 10-15-16  
   performance tools 12-11  
 segments 10-14, F-1  
 Select All 3-6  
 selection 6-3-9, B-1-2  
   extending 6-7  
   by line number 6-6  
   marker 6-8-9  
   operators 6-4  
   position 6-7  
 Selection Expression 3-9, 3-10

semicolon (;) 4-13, 5-20, 8-6, 8-15  
 Send to Back 7-9  
 Set 4-12  
 Set Byte 11-10  
 SetDirectory 3-13, 4-8, 9-11  
 Set Directory 2-6, 2-9  
 Set Directory... 3-14  
 SetFile 4-7, 14-15  
   -a option 4-23  
 Set Memory 11-10  
 SetPrivileges 4-8, 14-15  
 SetVersion 4-8  
 SFGetFile 4-19  
 SFPutFile 4-19, 4-20  
 \$\$Shell 8-23  
 (ShellDirectory) 5-10  
 Shift 5-21, 5-22  
 Shift Left 3-7  
 Shift Right 3-7  
 Show Build Commands 3-16  
 Show Clipboard 3-6  
 Show Directory 2-5, 2-6, 3-14  
 Show Full Build Commands 3-16  
 Show Invisibles 3-2, 3-7  
 signal 13-6  
 signal handling, MPW Shell 13-6  
 signature 9-7  
 sigpause 13-6  
 sigrelease 13-6  
 simple commands 5-6  
 simple command terminator (;) 5-5  
 single quotes (') 11-7  
 size record F-1, F-6  
 slashes 3-10  
 {SourceFiles} 10-13  
 square brackets ([]) 6-11, 8-12  
 stack, MPW Shell 13-10  
 Stack Crawl 11-17  
 stack space G-1  
 Stack Windows 3-11  
 standard input 4-3, 5-26, 5-27-28  
 standard output 4-3, 5-26, 5-28  
 Startup 1-9, 2-3, 4-4, 5-2, 5-8  
   variables 5-11-12  
 static 10-20  
 (Status) 5-10  
 status codes, MPW Shell 13-7  
 Status Panel 2-3, 5-3  
 Status Register 11-11  
 status value 5-3-4  
 StdErr 5-30  
 StdIn 5-30  
 StdOut 5-30  
 Step 11-12  
 Step Spy 11-13  
 Step Till 11-13  
 string 8-10, 8-12

strings 8-24-25, D-7  
   MacBug and 11-7  
   operators 5-24  
   types 8-12  
 structured commands 1-4, 5-6, 5-19-24  
 Suspend 1-9, 5-8  
 switch 8-10, 8-14  
 switch types 8-14  
 Symbol Dump 11-18  
 Symbol Exchange 11-18  
 symbols, MacBug 11-8  
 SymMgr 10-13  
 sysheap 8-8  
 System Folder 11-2  
 (SystemFolder) 5-10  
 SysTypes.r 1-8, 8-3

## T

Tab 3-7, 6-2  
 (Tab) 5-12  
 tagging operator (@) 5-24  
 Target 4-8, 5-9, 6-2  
 (Target) 5-10  
 TargetFile 10-4  
 target file 10-2  
 target window 4-2, 4-9  
 terminal D-1  
 TermPerf 12-3  
 (Test) 5-11  
 text files, suffixes E-1  
 Text Only 3-2  
 text string 8-24  
 TextTitle 14-10  
 three-state buttons 14-15  
 Tile Windows 3-11  
 \$\$Time 8-23  
 Time Manager 12-3  
 TLACvt 2-2  
 TLACvt.Directives 2-2  
 token delimiters D-5  
 (ToolDir) 10-13  
 tools 5-2  
 Tools folder 2-2  
 Total Display 11-11  
 Total Floating Point 11-11  
 Trace 11-12  
 Transfer 7-4  
 Translate 6-2  
 Try Cursor 7-8  
 type 8-4, 8-5, 8-9-13, 8-15, D-2, D-3-4  
 \$\$Type 8-23  
 type declaration 8-4  
 Types.r 1-8, 8-3, 8-5, 8-15

## U

- unchanged 8-8
- #undef 8-19, D-5
- Undo 3-6, 6-2, 7-7
- unlocked 8-8
- Unmark 3-2, 6-2
- Unmark... 3-13
- unprotected 8-8
- unsigned 8-11
- Use Full Window 7-9
- user interface 1-2
- Use RSRC Rect 7-9
- UserStartup 1-9, 2-4, 3-14, 4-4,  
5-8, 9-11
- Utilities 10-13

## V

- variables D-7
  - defining and modifying 5-13-14
  - exporting 5-14
  - makefile 10-7-8
  - MPW Shell 5-9-14, 13-4
  - pathname 4-12
  - predefined 5-10
  - Rez 8-23
- \$\$Version 8-23
- Volumes 4-8

## W

- \$\$Weekday 8-23
- Where 11-20
- Which 4-8
- wildcard characters 3-10, 4-13, 5-2,  
5-25, 6-11
- Window menu 2-4, 3-11
- windows
  - commands 1-2
  - MPW Shell 13-10
  - names 4-8-13
  - typing commands in 4-3
- Windows 4-8
- {Windows} 5-10
- window template 7-12
- {WordSet} 5-12
- {Worksheet} 5-10
- Worksheet window 2-3, 2-7, 3-11,  
5-28
- Wrap-Around Search 3-9
- wstring 8-10, 8-12

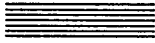
## Y

- \$\$Year 8-23





## **Part II**



# **Command Reference**

Part II is a command dictionary that describes each of the Macintosh Programmer's Workshop 2.0 commands. When you have become sufficiently familiar with the material in Part I, you can remove Part II to a smaller separate binder for convenient desktop reference. 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.

---

---

## Command prototype

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

### Syntax

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

*Note:* Filenames, command names, and options are not sensitive to case. The syntax notation itself is described in the preface in Part I.

### 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 of the options apply to the processing of all of the files, regardless of the ordering of options and files.

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

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>name</i>   | Named window or file                                                |
| §             | The selection in the target window (the second window from the top) |
| <i>name.§</i> | 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 maybe the file is missing or that there is 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 Assembler writes listings to standard output. The 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 values 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 by the Shell only.

**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 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 Menu Manager chapter 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: *style* can be any of the following:  
B            Bold  
I            Italic  
U            Underline  
O            Outline  
S            Shadow

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 values:

- 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 quotes—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 off before the item is actually added. The double quotes 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.

**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 the Menu Manager chapter of *Inside Macintosh*.

---

---

## Adjust—adjust lines

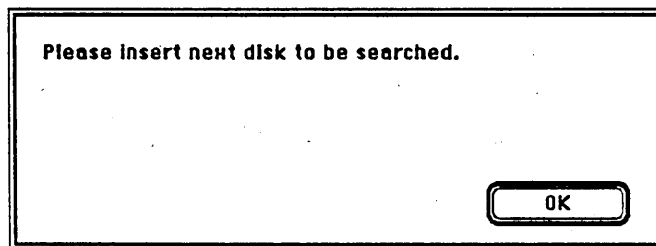
|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Adjust [-c <i>count</i> ] [-l <i>spaces</i> ] <i>selection</i> [ <i>window</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Description</b> | <p>Finds and selects the given selection, and shifts all lines within the selection to the right by one tab, without changing the indentation.</p> <p>If a count is specified, <i>count</i> instances of selection are affected. The -l option lets you move lines by any number of spaces to the left or right.</p> <p>If you specify the <i>window</i> parameter, the command operates on <i>window</i>. 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).</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Status</b>      | <p>Adjust may return the following status values:</p> <ul style="list-style-type: none"><li>0 At least one instance of the selection was found</li><li>1 Syntax error</li><li>2 Another error</li></ul>                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Options</b>     | <p>-c <i>count</i> Repeat the select-and-adjust operation <i>count</i> times.</p> <p>-l <i>spaces</i> Every line within the selection will be shifted <i>spaces</i> to the right. You can shift a selection left by specifying a negative value for <i>spaces</i>.</p>                                                                                                                                                                                                                                                                                                        |
| <b>Examples</b>    | <p>Adjust -l 4 S<br/>Shifts the lines containing the target selection to the right by four spaces.</p> <p>Adjust -l -8 /if/Δ:Δ/else/<br/>Selects everything after the next "if" and before the following "else", and shifts all lines within the selection to the left by eight spaces.</p>                                                                                                                                                                                                                                                                                   |
| <b>See also</b>    | <p>Align command.</p> <p>"Selections" in Chapter 6.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

---

---

## 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 values:
- 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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Alias [ <i>name</i> [ <i>word...</i> ] ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Description</b> | <p><i>Name</i> becomes an alias for the list of words. Subsequently, when <i>name</i> is used as a command name, <i>word...</i> will be substituted in its place.</p> <p>If only <i>name</i> is specified, any alias definition associated with <i>name</i> is written to standard output. If <i>name</i> and <i>word</i> 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.)</p> <p>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.</p> <p>You can remove aliases with the Unalias command.</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Output</b>      | When parameters are omitted, the Alias command writes aliases and their values to standard output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Status</b>      | Alias may return the following status values:<br>0 No errors<br>1 The specified alias could not be found                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Examples</b>    | <pre>Alias Dir Directory Creates an alias "Dir" for the Directory command.</pre><br><pre>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:</pre> <pre>Top                # find top of target window Top Sample.a      # find top of window Sample.a                   # (equivalent to "Find • Sample.a")</pre>                                                                                                                                                                                                                                                                                   |
| <b>See also</b>    | Unalias command.<br>"Command Aliases" in Chapter 5.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

---

---

## Align—align text to left margin

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Align [ <i>-c count</i> ] <i>selection</i> [ <i>window</i> ]                                                                                                                                                                                                                                                                                                                                                       |
| <b>Description</b> | <p>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.</p> <p>If you specify the <i>window</i> parameter, the Align command will act on <i>window</i>. 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).</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Status</b>      | <p>Align may return the following status values:</p> <ul style="list-style-type: none"><li>0 At least one instance of the selection was found</li><li>1 Syntax error</li><li>2 Any other error</li></ul>                                                                                                                                                                                                           |
| <b>Option</b>      | <i>-c count</i> Repeat the select-and-align operation <i>count</i> times.                                                                                                                                                                                                                                                                                                                                          |
| <b>Examples</b>    | <p>Align \$</p> <p>Same as the Align menu item; that is, aligns all lines in the default selection with the first line of the selection.</p> <p>Align /Begin/:/End/</p> <p>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".</p>                                                  |
| <b>See also</b>    | <p>Adjust command.</p> <p>"Selections" in Chapter 6.</p>                                                                                                                                                                                                                                                                                                                                                           |

---

---

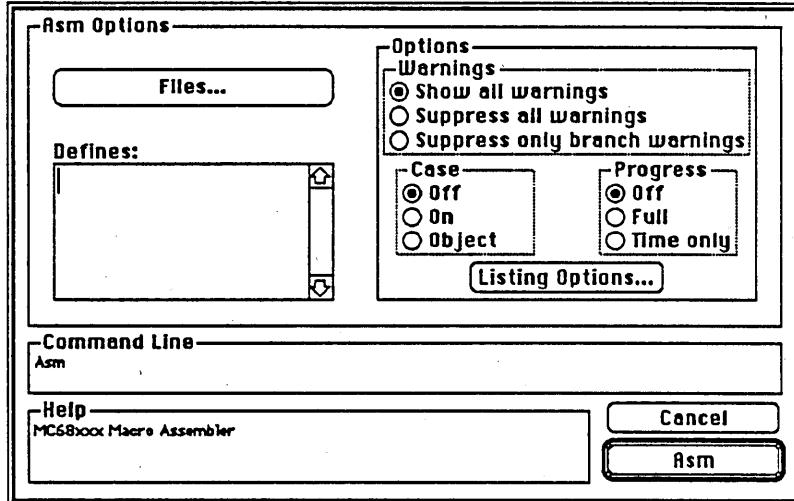
## Asm—MC68xxx Macro Assembler

- 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 file names end in the suffix ".a". Each file is assembled separately—assembling file *name.a* creates object file *name.a.o*. The object file name can be changed with the `-o` option.
- See the *MPW 2.0 Assembler Reference* for details of 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.)
- 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 file name can be changed with the `-lo` option.
- 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 values:
- 0    No errors detected in any of the files assembled
  - 1    Parameter or option errors
  - 2    Errors detected
  - 3    Execution terminated

## 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 to 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 to 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 non-macro 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, then 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. Note that in order to test whether or not the name is defined, you should use the `&Type` function. 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:

```
&name SETA value
```

```
&name SETC value
```

Note that in order to test whether the name is defined, you should use the `&Type` function. 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'`).

**-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; both may not be omitted. The default listing font is Monaco 7. Note that listings will be 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 will be searched. The search order is as follows:

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

If the file wasn't found, and the *pathname* used to specify the file was 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 **-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 Assembler.)

**-pagesize** [*l*] [*,w*] Set the listing page size. (This option is meaningful only if the **-s** or **-l** option is specified; both may not be omitted.) 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 2.0 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.
- t** Display the assembly time and the number of lines to the diagnostic file even if progress information (**-p**) is not being 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 Assembler 2.0 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 do the copies. Instead, a script of MPW Shell duplicate commands is generated.

Backup’s default operation is built 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 file has the same file type and creator, 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 folders 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
```

The 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 will be generated at the end of the list of duplicates. These commands eject the source and/or destination disks if either or both disks are specified as disk drive numbers 1 or 2 as the **-from** and **-to** option parameters.

If you use the **-compare** option, a Compare command will be written to the standard output file if the files are of type TEXT. Note that only the Compare will be generated if you specify **-compare only**. You can also specify 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 will be written to the diagnostic output file. The summary will show the modification dates of both the source and destination files. If you use the **-check** option, then a report of 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, is written to the diagnostic output file. You can redirect this report to a file by using the **-co** option.

## Status

Backup may return the following status values:

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

*Note:* Backup will return status value 3 when no files need copying. If there is no copying because *none* of the files in the source folder exists in the destination folder, Backup will also report 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 will not report this as an error if you use the **-l**, **-a**, or **-since** option.

## Options

- a** Normally, if a file in the destination ("to") does not exist in the source ("from"), then it is ignored. 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** ("multi-disk") 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).

The screenshot shows a graphical user interface for the Backup utility. It is titled "Backup Options" and is divided into several sections:

- Source/Destination Folders:** Contains two buttons: "Select 'From' Directory..." and "Select 'To' Directory...".
- Search Criteria:** Includes a checkbox for "Recursive", a "Type" field, and two text input fields for "Since File" and "Since Date".
- Drive Options:** Contains three checkboxes: "Eject disks", "Multi-disk", and "Alternate drives".
- Check Report Options:** Contains four checkboxes: "Files not in 'to'", "Files not in 'from'", "'to's newer than 'from's", and "Folders not in 'to'".
- Output Files...:** A button to specify the output file name.
- More Options...:** A button to access additional settings.
- Command Line:** A text field containing the command "Backup".
- Help:** A text box containing the text: "Files in a source ('from') folder are copied to a destination ('to') folder based on the modification date. Backup does not actually do the copies. Instead a script of MPV Shell duplicate commands is generated." Below this are "Cancel" and "Backup" buttons.

- 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:* only the outermost folder names are reported.
- newer** Report all files in the destination ("to") that are *newer* than the source ("from").

Note that 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 will be generated and not the duplicates. Additional Compare command options and output redirection may 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 have to be enclosed in quotes 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 then the options **-check to**, **-check alltos**, **-m**, 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 will be generated and 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 as follows:  
*command... fromFilename toFilename*  
If **-sync** is specified then 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 operation of copying the source to the destination) then the command string is generated, as follows:  
*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, as follows:  
*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 will be 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, then 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 in order 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 files listing of all files in the source ("from") folder. The **-to** option cannot be specified when **-l** is used. If the **-do** option is specified, then the **-do** command string, rather than a file listing, is generated for each file.

**-m** "Multi-disk" 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** specified 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 may 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 will be ejected. If not, the appropriate duplicate commands will be generated to update the out-of-date files on the disk in drive 1. An **eject 1** command will be generated to eject the disk after the duplicate commands are processed.

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

Recursively (**-r**) process 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 will be 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 will be written to the diagnostic file.

## Limitations

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

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

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

---

---

## Beep—generate tones

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Beep [ <i>note</i> [, <i>duration</i> [, <i>level</i> ] ] ]...                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Description</b> | <p>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.</p> <p><i>Note</i> is one of the following:</p> <ul style="list-style-type: none"><li>□ A number indicating the count field for the square wave generator, as described in the Summary of the Sound Driver chapter of <i>Inside Macintosh</i>.</li><li>□ A string in the following format:<br/>[ <i>n</i> ] <i>letter</i> [ #   b ]</li></ul> <p><i>n</i> 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.</p> <p>The optional <i>duration</i> is given in sixtieths of a second. The default duration is 15 (one-quarter second).</p> <p>The optional sound <i>level</i> is given as a number from 0 to 255. The default level is 128.</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Diagnostics</b> | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Status</b>      | A status value of 0 is always returned.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Examples</b>    | <p>Beep</p> <p>Produce a simple beep on the speaker.</p> <pre>Beep 2C, 20 '2C#, 40' 2D, 60</pre> <p>Play the 3 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.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

---

---

## Begin...End—group commands

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Begin<br><i>command...</i><br>End                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Description</b> | <p>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 (&amp;&amp; and   ), or input/output specifications (&lt;, &gt;, &gt;&gt;, ≥, ≥≥) are used, the operator must appear after the End command, and applies to all of the enclosed commands.</p> <p><i>Note:</i> 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.</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Diagnostics</b> | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Status</b>      | The status value of the last command executed is returned. (If no commands appear between Begin and End, 0 is returned.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

## Examples

The following commands save the current variables, exports, aliases, and menus in 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. Therefore, after executing the above commands, the command

```
Execute SavedState
```

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

*Note:* This technique is used in the Suspend command file 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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Break [ If <i>expression</i> ]                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Description</b> | If <i>expression</i> is nonzero, Break terminates execution of the immediately enclosing For or Loop command. (Null strings are considered zero.) If the "If <i>expression</i> " is omitted, the break is unconditional. (For a definition of <i>expression</i> , see the Evaluate command.)                                                                                                                                        |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Status</b>      | Break may return the following status values:<br>0 No errors detected<br>-3 Break is found outside a For...End or Loop...End, or the parameters to Break are incorrect<br>-5 Invalid expression                                                                                                                                                                                                                                     |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Example</b>     | <pre>Set Exit 0 For file in Startup UserStartup Suspend Resume Quit     EnTab "{file}" &gt; temp     Break If (Status) != 0     Rename -y temp "{file}"     Print -h "{file}"     Echo "{file}" End</pre> <p>This For loop entabs and prints each of the special MPW command files; the Break command terminates the loop if a nonzero status value is returned. (See the For command for further explanation of this example.)</p> |
| <b>See also</b>    | For, Loop, and If commands.<br>Evaluate command (for a description of expressions).<br>"Structured Commands" in Chapter 5.                                                                                                                                                                                                                                                                                                          |

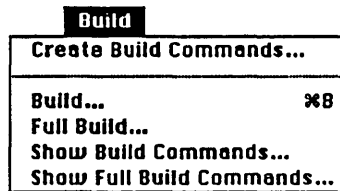
---

---

## 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 value 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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | BuildProgram program [ <i>options...</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Description</b> | <p>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.</p> <p>Make is used to determine the commands needed to do the build. If file <i>program.make</i> exists, it is used as the makefile. If not, file MakeFile is used.</p> <p>The options specified are passed directly to Make, and control the generation of the build commands. BuildProgram is used to implement the Build and Full Build menu items in the Build menu.</p> |
| <b>Type</b>        | Script.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Output</b>      | A transcript of the build, including timing information and the commands used to do the build, is written to standard output.                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Diagnostics</b> | Errors that occur during the generation of the build commands or during the build are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Status</b>      | Status value 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, Link) is returned.                                                                                                                                                                                                                    |
| <b>Option</b>      | <p>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 <b>-e</b>.</p> <p><b>-e</b> 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.</p>                                                                                                                                                                 |
| <b>Example</b>     | <pre>Open "{Worksheet}" BuildProgram -e Count &gt;&gt; "{Worksheet}" &gt;&gt; Dev:StdOut</pre> <p>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 item is implemented using similar commands.</p>                                                                                                                                                                                                            |
| <b>See also</b>    | "Building a Program: An Introduction" in Chapter 2.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

---

---

## C—C Compiler

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | C [ <i>option ...</i> ] [ <i>file</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Description</b> | Compiles the specified C source file. Compiling file <i>Name.c</i> creates object file <i>Name.c.o</i> . (By convention, C source file names end in a “.c” suffix.) If no filenames are specified, standard input is compiled and the object file “c.o” is created.<br>See the <i>MPW C 2.0 Reference</i> for details of the C language definition.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Type</b>        | Tool.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Input</b>       | If no filenames are specified, standard input is compiled. You can terminate input by pressing Command-Enter.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Output</b>      | If you specify the <b>-e</b> option, preprocessor output is written to standard output, and no object file is produced.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Diagnostics</b> | Errors and warnings are written to diagnostic output. If the <b>-p</b> option is specified, progress and summary information is also written to the diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Status</b>      | The following status values may be returned:<br>0 Successful completion<br>1 Errors occurred                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Options</b>     | <b>-c</b> Include comments with the preprocessor output. This option has no effect unless the <b>-e</b> option is also specified. By default, comments are not written to the preprocessor output.<br><b>-d name</b> Define <i>name</i> to the preprocessor with the value 1. This is the same as writing<br>#define <i>name</i> 1<br>at the beginning of the source file. (The <b>-d</b> option does not override #define statements in the source file.)<br><b>-d name=string</b> Define <i>name</i> to the preprocessor with the value <i>string</i> . This is the same as writing<br>#define <i>name</i> <i>string</i><br>at the beginning of the source file.<br><b>-e</b> Do not compile the program. Instead, write the output of the preprocessor to standard output. This option is useful for debugging preprocessor macros. |



- elems 881** Use MC68881 floating-point processor instructions instead of SANE routines for transcendental functions. This option generates code that can run only on a Macintosh II. The MC68881 floating-point processor uses less accurate but faster algorithms for transcendental functions. A program compiled with this option will run faster, but the results may not be the same as those obtained with the SANE routines. This option has no effect unless the **-mc68881** option is also used.
- g** Generate stack frame pointers in A6 (that is, LINK A6,x ... UNLK A6) for all functions. Insert the procedure name into the object code that follows the procedure's RTS instruction. Use this option if you plan to debug the program with MacsBug.
- ga** Generate stack frame pointers in A6 (that is, LINK A6,x ... UNLK A6) for all functions.
- 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 is as follows:
1. The include file name is used as specified. If a *full pathname* is given, then no other searching is applied.  
If the file wasn't found, and the pathname used to specify the file was 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 **-i** options, in the order listed.
  4. The directories specified in the Shell variable {CIncludes}.
- mc68020** Use the full instruction set of the MC68020 processor. This option generates object code that runs only on a Macintosh II.
- mc68881** Use the MC68881 floating-point processor for all arithmetic operations, conversions between different binary formats, and comparisons. This option generates code that can be run only on a Macintosh II.
- The **-mc68881** option has the following effects:
- establishes the extended data type to the 96-bit format used by the MC68881 floating-point processor
  - causes expression evaluation to be performed in floating-point registers
  - allows the Compiler to allocate extended variables to floating-point registers
- Even when this option is invoked, the Compiler generates SANE instructions instead of MC68881 instructions for transcendental functions. To override this feature, use both the **-elems881** option and the **-mc68881** option.
- Note:* Use of the **-mc68881** option does not set the **-mc68020** option. To get the best results on the Macintosh II, set both the **-mc68020** and **-mc68881** options.

- 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, compilation time, and compilation memory requirements) to diagnostic output.
  
- q** Optimize the code for speed, even if it's necessary to make the object code larger. By default, the Compiler performs optimizations that make the code both smaller and quicker—the **-q** option will perform further optimizations that may make the code faster, but also larger. The **-q** option should be specified only for those parts of the program that are executed frequently—it's counterproductive to specify **-q** on code that's rarely executed.
  
- q2** Allow the optimizer to assume that memory locations do not change except by explicit stores—that is, the optimizer is guaranteed (1) that no memory locations are I/O registers that can be changed by external hardware, and (2) that no memory locations are shared with other processes that can change them asynchronously with respect to the current process. This option must be used with extreme caution in device drivers, operating systems, and shared-memory environments, and when interrupts are present.
  
- s *name*** Name the object code segment. (The default segment name is "Main".) Because a segment may not exceed 32K bytes, large programs require multiple segments with different names. This option is overridden if the following statement appears in the source code:  

```
#define __SEG__ name
```
  
- 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.)
  
- x6** Use MOVE #0,x instructions rather than CLR x instructions for nonstack addresses. This option may be useful when writing device drivers.
  
- x55** Make bit fields of types int, short, and char be signed. (The default is for all fields to be unsigned.)

- z127** Always allocate 32 bits for enumerated data types, to maintain compatibility with Standard C. The default is to allocate 8, 16, or 32 bits.
- Caution:** This option is not compatible with the Macintosh interface libraries.
- x149** Allow *float*, *double*, and *comp* variables to be allocated in floating-point registers. This option has no effect unless the **-mc68881** option is also specified. By default, only extended variables are allocated in floating-point registers.
- z84** Enable language anachronisms. Warning messages are provided when anachronisms are encountered, and the constructs are compiled. (See the *MPW C 2.0 Reference* for information.)

**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 the `CExamples` folder.)

**Limitation**

1 megabyte of RAM is recommended; on a Macintosh 512K, even small C programs may not compile.

**Availability**

The C Compiler is available as part of a separate Apple product, Macintosh Programmer's Workshop C.

**See also**

*MPW C 2.0 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 be case insensitive or restricted to a fixed number of characters. (See the “Options” section below.) 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 non-identifier 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 values may be returned:

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

Canon Options

DictionaryFile Case sensitive

Assembler identifiers

Number of significant characters

Output Error

Command Line

Canon

Help
Copies the specified files to standard output, replacing identifiers with the canonical spellings given in dictionaryFile.

Options

- s Use case-sensitive matching. (Pattern matching is normally case insensitive.)
- a Causes the characters \$, %, and @ to be considered letters (for defining identifiers). This option is useful when processing 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 C 2.0 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 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 don't mind using 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 [<i>file...</i>] |
| 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 values 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 | <pre>Catenate Makefile.a</pre> <p>Writes Makefile.a to the active window, immediately following the command.</p> <pre>Catenate File1 File2 > CombinedFile</pre> <p>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.</p> <pre>Set selection ``Catenate \$``</pre> <p>Captures the selection from the target window in the Shell variable {selection}.</p> <pre>Catenate >> {Worksheet}</pre> <p>Appends all subsequently entered text to the Worksheet window (until you indicate end-of-file by pressing Command-Enter).</p> |
| Warning | Beware of commands such as <pre>Catenate File1 File2 > File1</pre> <p>This command will cause the original data in File1 to be lost. To append one file to another, use the form <pre>Catenate File2 >> File1</pre></p> |
| See also | Duplicate command. “Redirecting Input and Output” in Chapter 5. |

Clear—clear the selection

| | |
|--------------------|--|
| Syntax | Clear [-c <i>count</i>] <i>selection</i> [<i>window</i>] |
| Description | <p>Finds <i>selection</i> and deletes its contents. The selection is not copied to the Clipboard. (For a definition of <i>selection</i>, see Chapter 6.)</p> <p>If <i>window</i> 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).</p> |
| Type | Built-in. |
| Input | None. |
| Output | None. |
| Diagnostics | Errors are written to diagnostic output. |
| Status | <p>Clear may return the following status values:</p> <ul style="list-style-type: none">0 At least one instance of <i>selection</i> was found1 Syntax error2 Any other errors |
| Option | -c <i>count</i> Repeat the select-and-delete operation <i>count</i> times. |
| Examples | <p>Clear \$</p> <p>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.</p> <p>Clear /BEGIN/:/END/</p> <p>Selects everything from the next BEGIN through the following END, and deletes the selection.</p> |
| See also | <p>Cut and Replace commands.</p> <p>"Selections" in Chapter 6 (see Appendix B for a summary).</p> |

Close—close specified windows

| | |
|--------------------|--|
| Syntax | Close [-y -n] [-a <i>windows...</i>] |
| Description | Close the window or windows specified by <i>windows</i> . 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. In scripts you can use the -y or -n option to avoid this interaction. Use the -a option instead of <i>windows</i> 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 values: 0 No errors 1 Syntax error 2 Any other error, such as "Window not found" |
| Options | -a Close all open Shell windows (except for the Worksheet, which cannot be closed). This option cannot be specified when any <i>windows</i> 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. |
| Examples | <pre>Close</pre> <p>Closes the target window, prompting the user with a confirmation dialog box if needed.</p> <pre>Close -a -y</pre> <p>Saves and closes all open windows.</p> <pre>Close -n Test.a Test.r</pre> <p>Closes the windows Test.a and Test.r without saving any of the changes.</p> |

Commando—display dialog box for a command

| | |
|--------------------|---|
| Syntax | Commando [<i>commandname</i>] [<i>Commandname</i>] ... |
| Description | <p>The Commando interface lets you operate any properly configured MPW tool or script by 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.</p> <p>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 14 describes the structure of the Commando resource and shows how to create Commando dialogs for your own tools and scripts.</p> |
| Type | Tool. |
| Input | None. |
| Output | If <code>commando</code> is invoked by typing <code>Commando [<i>commandname</i>]</code> , then the command line is simply written to standard output. If <code>commando</code> is invoked by typing <code>[<i>Commandname</i>] ...</code> (the ellipsis generated by Command-semicolon), then the command line is intercepted by the Shell and executed. |
| Diagnostics | Errors are written to diagnostic output. |
| Status | Commando may return the following status values: 0 The Do It button was selected 1 The Cancel button was selected 2 Error occurred while parsing the <code>cmdo</code> resource 3 I/O or program error |
| Options | None. |
| Examples | <p><code>Commando Rez</code> Displays the frontmost Rez dialog box shown under “Rez” in Part II.</p> <p><code>Rez...</code> Displays the frontmost Rez dialog box shown under “Rez” in Part II, exactly as in the previous example.</p> |
| See also | “Invoking Commando” in Chapter 4. Chapter 14. |

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 (**-I**).

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

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

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, to how far out of synch 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 may 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, then 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, then the following message will appear if *any* mismatches occurred:

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

If both files match, then 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
- 3 Execution terminated

Options

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

The screenshot shows the 'Compare Options' dialog box. It features a title bar 'Compare Options' and a main area with several sections. At the top, there are two text boxes labeled 'File 1' and 'File 2' separated by a double colon '::'. Below them is a button labeled 'I/O Redirection...'. The main area is titled 'Options' and contains a grid of checkboxes: 'Ignore blanks', 'Ignore trailing blanks', 'Ignore case', 'Show mismatched lines' (checked), 'Quiet', 'Progress', 'Fixed grouping', 'No tabs', 'No line numbers', 'Grouping', 'Depth', 'Width', and 'Context'. The 'Columns' field is a text box containing '1-255'. Below the options is a 'Command Line' section with a text box containing 'Compare'. At the bottom left is a 'Help' section with the text 'Compare the lines of two TEXT files and writes their difference to the standard output file.' At the bottom right are two buttons: 'Cancel' and 'Compare'.

-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, 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 synch 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 being used, 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 to 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 is a number from 70 to 255 that controls the number of characters displayed in each portion by specifying the total display line 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, except when the **-b** option is used, tabs are expanded into spaces. 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.S File.new.S
```

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).

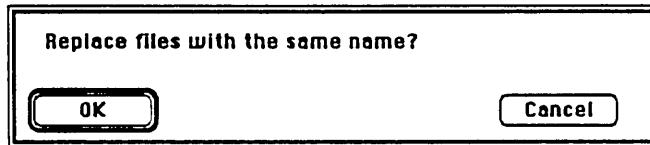
Confirm—display confirmation dialog box

| | |
|--------------------|---|
| Syntax | Confirm [-t] [<i>message...</i>] |
| Description | <p>Displays a confirmation dialog box with OK and Cancel buttons and the prompt <i>message</i>. There is no output to this command—the result of the dialog is returned in the {Status} variable.</p> <p><i>Note:</i> 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.)</p> |
| 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 | <p>The Confirm command may return the following status values:</p> <ul style="list-style-type: none">0 The OK button was selected1 Syntax error4 The Cancel button was selected5 The Cancel button was selected in a three-way dialog box—see the -t option <p><i>Note:</i> In the context of a two-button dialog, Cancel means the same thing as No; OK means Yes.</p> |
| Option | <p>-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.</p> |

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 <i>expression</i>] |
| Description | If <i>expression</i> 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 <i>expression</i> ” clause is omitted, the Continue is unconditional. If no further iterations are possible, the For or Loop is terminated. (For a definition of <i>expression</i> , 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 values: 0 No errors -3 Error in parameters, or Continue not within For...End or Loop...End -5 Invalid expression |
| Options | None. |
| Example | <pre>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 Print (PrintOptions) {list} Set Exit 1</pre> <p>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.</p> <p>(For a full explanation of this example, refer to the Confirm command.)</p> |
| 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 | <code>Copy [-c <i>count</i>] <i>selection</i> [<i>window</i>]</code> |
| Description | <p>Finds <i>selection</i> 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.</p> <p>For a definition of <i>selection</i>, see "Selections" in Chapter 6; a summary of the selection syntax is contained in Appendix B.</p> <p><i>Note:</i> To copy files, use the Duplicate command.</p> |
| Type | Built-in. |
| Input | None. |
| Output | None. |
| Diagnostics | Errors are written to diagnostic output. |
| Status | <p>Copy may return the following status values:</p> <ul style="list-style-type: none">0 At least one instance of the selection was found1 Syntax error2 Any other error |
| Option | <code>-c <i>count</i></code> For a count of <i>n</i> , find and copy the <i>n</i> th instance of <i>selection</i> . |
| Examples | <p><code>Copy s</code> 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.</p> <p><code>Copy /BEGIN/:/END/</code> Selects everything from the next BEGIN through the following END, and copies this selection to the Clipboard.</p> |
| See also | <p>Cut and Paste commands.</p> <p>"Selections" in Chapter 6 and Appendix B.</p> |

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, preceded by the filename, and 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 values:
- 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
- | | | |
|-------------------------|------------------|-------------------|
| <code>MakeFile.c</code> | <code>43</code> | <code>981</code> |
| <code>Count.c</code> | <code>153</code> | <code>3327</code> |
| <code>Total</code> | <code>196</code> | <code>4303</code> |
- `Files | Count -l`
- Display the total number of files and directories in the current directory.
- `Count -l $`
- Display 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.

CreateMake—create a simple makefile

Syntax

CreateMake [-Application | -Tool | -DA] *program file...*

Description

Create a simple makefile for building the specified program. The *program* parameter 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”), Pascal (“.p”), and/or Rez (“.r”).

Library files (suffix “.o”) may also be specified. The program will be linked with these files. CreateMake automatically links with the library files listed below. It is not necessary to specify these files as parameters to CreateMake.

Makefiles for building applications (the default), tools, and desk accessories may be created.

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}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
- 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 values may be returned:

0 Successful completion
1 Parameter or option error

Options

- application** Create build commands for building an application. This is the default.
- tool** Create build commands for building a tool.
- da** Create build commands for building a desk accessory.

Example

```
CreateMake -Tool Count Count.c Stubs.a Count.r
```

Creates the makefile Count.make containing commands for building tool Count from source files Count.c, Stubs.a, and Count.r. The makefile will be similar to the following:

```
# File:          Count.make
# Target:       Count
# Sources:     Count.c Stubs.a Count.r
# Created:     Monday, February 9, 1987 3:04:44 PM

Count.c.o f Count.make Count.c
      C Count.c
Stubs.a.o f Count.make Stubs.a
      Asm Stubs.a
Count ff Count.make Count.r
      Rez -append Count.r -o Count
Count ff Count.make Count.c.o Stubs.a.o
      Link -w -t MPST -c 'MPS ' @
          Count.c.o @
          Stubs.a.o @
          "{Libraries}"Interface.o @
          "{CLibraries}"CRuntime.o @
          "{CLibraries}"StdCLib.o @
          "{CLibraries}"CSANELib.o @
          "{CLibraries}"Math.o @
          "{CLibraries}"CInterface.o @
          "{Libraries}"ToolLibs.o @
      -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 | <code>Cut [-c <i>count</i>] <i>selection</i> [<i>window</i>]</code> |
| Description | <p>Finds <i>selection</i> 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.</p> <p>For a definition of <i>selection</i>, see "Selections" in Chapter 6; a summary of the selection syntax is contained in Appendix B.</p> |
| Type | Built-in. |
| Input | None. |
| Output | None. |
| Diagnostics | Errors are written to diagnostic output. |
| Status | <p>Cut may return the following status values:</p> <ul style="list-style-type: none">0 At least one instance of the selection was found1 Syntax error2 Any other error |
| Option | <code>-c <i>count</i></code> Find and cut <i>count</i> instances of <i>selection</i> . |
| Examples | <p><code>Cut \$</code></p> <p>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.)</p> <p><code>Cut /BEGIN/:/END/</code></p> <p>Selects everything from the next BEGIN through the following END, copies the contents of the selection to the Clipboard, and then deletes the selection.</p> |
| See also | <p>Clear, Copy, and Paste commands.</p> <p>"Selections" in Chapter 6 and in Appendix B.</p> |

CvtObj—convert Lisa Workshop object files to MPW object files

Syntax CvtObj [*-n namesFile*] [*-o outputFile*] [*-p*] *LisaObjFile*

Description Converts a Lisa object file (.OBJ file) to the Macintosh object format (.o file). This command is for Lisa Workshop users who have old object files but no source files that can be ported to the MPW system.

CvtObj supports object files produced by the Lisa Pascal Compiler, the Green Hills C Compiler, and the TLA Assembler that were targeted to the Macintosh runtime environment. Object files produced by other compilers have not been tested, but should work. The program should not be used to convert object files targeted for execution on Lisa.

Object files produced by the Lisa Pascal Compiler must have been compiled with the Macintosh code generation option, **\$M+**. Object files produced by the Green Hills C Compiler must have been compiled with the default code generation option, that is, the **-lisa** option must not have been specified. Assembler code produced by the TLA Assembler should conform to the guidelines outlined in the Using Assembly Language chapter of *Inside Macintosh*.

CvtObj detects and rejects a number of Lisa object record types. If this happens, CvtObj generates a fatal error message (“Can’t handle ...”), and terminates without producing an output file. However, CvtObj cannot detect and reject all object files targeted for execution on the Lisa, especially Pascal and TLA Assembler files.

The Lisa Workshop tools support only 8-character case-insensitive (shifted to uppercase) external identifiers. The MPW Compilers support variable-length, case-sensitive external identifiers. (The MPW Pascal Compiler still defaults to upshifting Pascal identifiers, primarily for language compatibility, portability of sources, and ease in providing both C and Pascal interfaces to the Macintosh ROM routines.) CvtObj provides the **-n** option for substituting names, so that old object files can be properly linked with new object files. The **-n** option specifies a “names” file, which controls name substitution.

Data Initialization: In general, CvtObj automatically matches the Lisa object file semantics with those of the Macintosh. However, data initialization records are more difficult to handle. With the Lisa tools, data areas were often defined with differing lengths, partial contents in different files, and so on. The underlying model was Fortran-named common areas, with multiple initialization sources. On the Macintosh, the default is to use only the first definition of a data module. In order to match the Macintosh default as closely as possible, CvtObj does not emit a defining instance of a data area unless initialization values are seen.

For C data areas that need to be initialized to zero, this behavior can result in Linker error messages reporting that the data area names are “unresolved external references.” If the references come from a file produced by CvtObj, then the **define** directive can be used in a “names” (**-n**) file to request CvtObj to emit a defining instance—this should result in a proper size definition for the data area, unless the data area was defined elsewhere as larger.

Note: The DumpObj command can be useful in tracking down and fixing anomalies in external names and data area definitions when using CvtObj.

| | |
|--------------------|--|
| Type | Tool. |
| Input | None. |
| Output | If no -o option is specified, output is written to the file <code>CvtObj.out.o</code> . |
| Diagnostics | Errors and warnings are written to the diagnostic file. Progress information is also written to the diagnostic file (with the -p option). |
| Status | The following status values may be returned: <ul style="list-style-type: none"> 0 No problem 2 Fatal error 3 User interrupt |
| Options | <p>-n <i>namesFile</i> Create name-conversion file. In this text file, lines that begin with a space or tab are interpreted as name substitution lines; the first name is the old name, the second name is the new name. (See “Examples” below.) All occurrences of the old name are replaced with the new name. Lines that begin with the word <code>define</code>, followed by an entry name, create a global data module for that name.</p> <p>-o <i>outputFile</i> Direct output to <i>outputFile</i>. The default output filename is <code>CvtObj.out.o</code>.</p> <p>-p Write progress information to diagnostic output.</p> |
| Examples | <pre>CvtObj -o MyFile.o MyLisaFile.OBJ</pre> <p>Converts file <code>MyLisaFile.OBJ</code>, placing the output in <code>MyFile.o</code>.</p> <pre>CvtObj -n NewNames -o MyFile2.o MyLisaFile2.OBJ</pre> <p>Converts file <code>MyLisaFile2.OBJ</code>, placing the output in <code>MyFile2.o</code>, and applying the name translations specified in <code>NewNames</code>. The <code>NewNames</code> file might contain the following:</p> <pre>ΔCLOSEOUT CloseOutput ΔDRAWROUN DrawRoundFigure ΔFOO2 Foo2 define FOO</pre> <p>where <code>Δ</code> indicates a leading space or tab character.</p> |
| See also | TLACvt, Link, and DumpObj commands. Appendix F. |

Date—write the date and time

| | |
|--------------------|---|
| Syntax | Date [-a -s] [-d -t] |
| Description | Writes the current date and time to standard output. |
| 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 values: 0 No error 1 Syntax error |
| Options | <p>-a Abbreviated date. Three-character abbreviations are used for the month and day of the week. For example, Thu, Aug 29, 1987.</p> <p>-d Write the date only.</p> <p>-s Short date form. Numeric values are used for the date. The day of the week is not given. For example, 8/29/87 (month/day/year).</p> <p>-t Write the time only.</p> |
| Examples | <pre>Date returns the date in the form Friday, February 14, 1986 10:34:25 PM Date -a returns Fri, Feb 14, 1986 10:34:25 PM Date -s -d returns 2/14/86</pre> |

Delete—delete files and directories

| | |
|--------------------|--|
| Syntax | Delete [-y -n -c] [-i] [-p] <i>name...</i> |
| Description | <p>Deletes file or directory <i>name</i>. If <i>name</i> is a directory, then <i>name</i> and its contents (including all subdirectories) are deleted.</p> <p>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.</p> |
| 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 | <p>The following status values may be returned:</p> <ul style="list-style-type: none">0 All specified objects were deleted (except for any directories skipped with the -n option)1 Syntax error2 An error occurred during the delete4 Cancel was selected or implied by the -c option |
| Options | <ul style="list-style-type: none">-i Ignore errors (that is, do not print messages, and return a status value 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 | <pre>Delete HD:MPW:*.c</pre> <p>Deletes <i>all</i> 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.)</p> |

Warning

Beware of potentially disastrous typing 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 would delete *all files* in the current directory, and also attempt 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 [<i>menuName</i> [<i>itemName</i>]] |
| Description | Deletes the user-defined item <i>itemName</i> , in the menu <i>menuName</i> . If <i>itemName</i> is omitted, all user-defined items for <i>menuName</i> are deleted. |
| Caution: | If <i>itemName</i> and <i>menuName</i> 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 values: 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. |

DeRez—Resource Decompiler

| | | | | | | | | | |
|--------------------|--|---|-----------|---|---------------------|---|----------------------|---|----------------------|
| Syntax | DeRez [<i>option...</i>] <i>resourceFile</i> [<i>resourceDescriptionFile...</i>] | | | | | | | | |
| Description | <p>Creates a text representation (resource description) of the resource fork of <i>resourceFile</i>, according to the resource type declarations in the resource description file(s). The resource description is written to standard output.</p> <p>A resource description file is a file of type declarations in the same format as that used by the Resource Compiler, Rez. The type declarations for standard Macintosh resources are contained in the files <i>Types.r</i> and <i>SysTypes.r</i>, 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.</p> <p>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.</p> <p>DeRez ignores all <code>include</code> (but not <code>#include</code>), <code>read</code>, <code>data</code>, and <code>resource</code> statements found in the <i>resourceDescriptionFile</i>. (It still parses these statements for correct syntax.)</p> <p>For the format of resource type declarations, see Chapter 6 and Appendix D.</p> | | | | | | | | |
| Type | Tool. | | | | | | | | |
| Input | <p>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.</p> <p>For all input files on the command line, the following search rules are applied:</p> <ol style="list-style-type: none">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 8.) | | | | | | | | |
| Diagnostics | If no errors or warnings are detected, DeRez runs silently. Errors and warnings are written to diagnostic output. | | | | | | | | |
| Status | <p>DeRez may return the following status values:</p> <table><tr><td>0</td><td>No errors</td></tr><tr><td>1</td><td>Error in parameters</td></tr><tr><td>2</td><td>Syntax error in file</td></tr><tr><td>3</td><td>I/O or program error</td></tr></table> | 0 | No errors | 1 | Error in parameters | 2 | Syntax error in file | 3 | I/O or program error |
| 0 | No errors | | | | | | | | |
| 1 | Error in parameters | | | | | | | | |
| 2 | Syntax error in file | | | | | | | | |
| 3 | I/O or program error | | | | | | | | |

Options

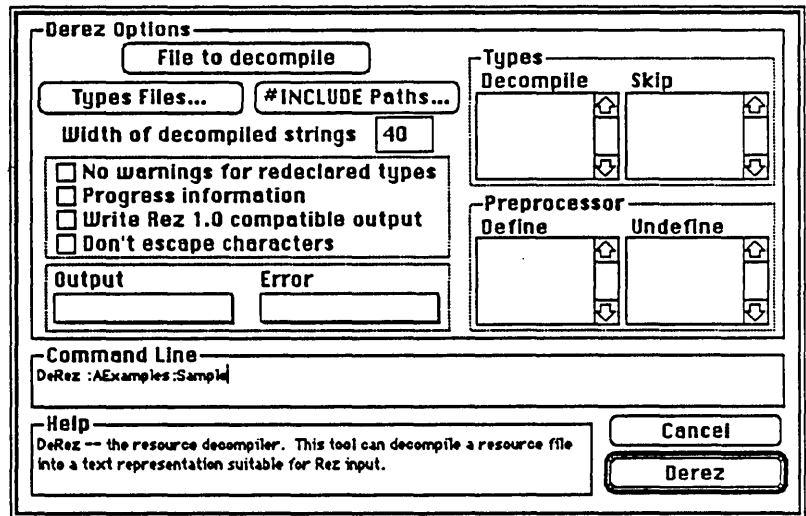
-c[ompatible] Generate output that is backward compatible with Rez 1.0.

-d(efine) macro[=data]

Define the macro variable *macro* to have the value *data*. If *data* is omitted, then *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, formfeed, vertical tab, and rubout) will be 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 how wide strings will be 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 quotes (') 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 quotes are needed to specify a literal type as long as it starts with a letter. No escape characters or anything fancy is 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 quotes 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 8.

Type declaration files in RIncludes folder:

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

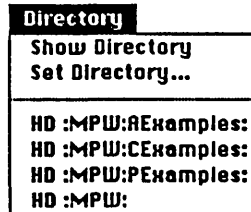
Directory—set or write the default directory

| | |
|--------------------|--|
| Syntax | Directory [-q <i>directory</i>] |
| Description | If specified, <i>directory</i> becomes the new default directory. Otherwise the pathname of the current default directory is written to standard output. <i>Note:</i> 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 values: 0 No error 1 Directory not found, command aborted |
| 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 | <pre>Directory</pre> <p>Writes the pathname of the current directory to standard output.</p> <pre>Directory HD:MPW:AExamples:</pre> <p>Sets the default directory to the folder AExamples in the folder MPW on the volume HD. The final colon is optional.</p> <pre>Directory Reports:</pre> <p>Sets the default directory to the volume Reports. Note that volume names must end in a colon.</p> <pre>Directory :Include:Pascal:</pre> <p>Sets the default directory to the folder Pascal in the folder Include in the current default directory.</p> |
| 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 below. 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 value 0 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.

DumpCode—write formatted resources

| | |
|--------------------|--|
| Syntax | DumpCode [<i>option...</i>] <i>resourceFile</i> |
| Description | <p>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.</p> <p>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.</p> <p>Some conventions about executable code resources are built into DumpCode, and affect the formatted output in special ways:</p> <ul style="list-style-type: none"><input type="checkbox"/> 'CODE' resources with ID 0 are formatted as a jump table (unloaded format).<input type="checkbox"/> Other 'CODE' resources have information about jump table entries in the first four bytes.<input type="checkbox"/> 'DRVR' resources have a special format at the beginning of the resource. <p>In addition, you can direct DumpCode to give a symbolic dump of data initialization descriptors and initial values.</p> |
| 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 values: 0 No problem 2 Fatal error 3 User interrupt |
| Options | <p><i>Note:</i> Numeric values for options can be specified as decimal constants, or as hex constants preceded by a "\$".</p> <p>-d Suppress the disassembly and dumping of code. (The default is to disassemble the code.)</p> <p>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.</p> <p>-di Suppress display of data initialization code.</p> |

- 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 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, then the rest of the segment is disassembled.
- rt *type[=ID]*** Dump only the single resource with type *type* and ID number *ID*. If *ID* is omitted, then 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,-(A7)
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.
 "Data Initialization" under the CvtObj command for a description of data initialization calls.
 "The Jump Table" in the Segment Loader chapter of *Inside Macintosh*, for a description of the jump table.
 Appendix F, "Object File Format."

DumpObj—write formatted object file

| | |
|--------------------|---|
| Syntax | DumpObj [<i>option...</i>] <i>objectFile</i> |
| 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 values: 0 No problem 2 Fatal error 3 User interrupt |
| Options | -d Suppress disassembly of code and display of data. The default is to disassemble code and to display data in hexadecimal and ASCII. -i Suppress substitution of names for IDs. The default is to pre-read the entire file, processing the Dictionary records, and then to 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 pre-read, which is also likely to fail. |

-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.

This option is useful in producing code that can be edited and submitted to the Assembler for reassembly.

The screenshot shows a dialog box titled "DumpObj Options". It has several sections:

- Object File:** A text input field.
- Output:** A text input field.
- Error:** A text input field.
- Checkboxes:**
 - Progress Info
 - No headers
 - Use IDs
 - File Locations
 - Names only
 - No disassembly Byte Range
- Modules:** A list box with a scroll bar.
- Command Line:** A text area containing "DumpObj".
- Help:** A text area containing "Dumpobj is used to display the contents of MPV object files".
- Buttons:** "Cancel" and "DumpObj" buttons.

-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 F.)

-m name

Dump a particular module. If *name* is an entry point, then 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 are defined 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 segment. If *byteN* is omitted, then the rest of the segment is disassembled.

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 F.

See also

DumpCode command.

Appendix F, "Object File Format."

Duplicate—duplicate files and directories

| | |
|--------------------|---|
| Syntax | Duplicate [-y -n -c] [-d -r] [-p] <i>name...</i> <i>targetName</i> |
| Description | <p>Duplicates <i>name</i> to <i>targetName</i>. (<i>Name</i> and <i>targetName</i> are file or directory names.) If <i>targetName</i> is a file or doesn't exist, then the file or directory <i>name</i> is duplicated and named <i>targetName</i>. If <i>targetName</i> is a directory, then the objects named are duplicated into that directory. (If more than one <i>name</i> is present, <i>targetName</i> must be a directory.) Created objects are given the same creation and modification dates as their source.</p> <p>If a directory is duplicated, then its contents (including all subdirectories) are also duplicated. No directory duplicated can be a parent of <i>targetName</i>.</p> <p><i>Name</i> can also be a volume; if <i>targetName</i> is a directory, then <i>name</i> is copied into <i>targetName</i>.</p> <p>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 command files to avoid this interaction.</p> |
| 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 | <p>The following status values may be returned:</p> <ul style="list-style-type: none">0 All objects were duplicated1 Missing or inaccessible parameters2 An error occurred4 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] [<i>parameters...</i>] |
| Description | <p>Writes its parameters, separated by spaces and terminated by a return, to standard output. If no parameters are specified, only a return is written.</p> <p>Echo is especially useful for checking the results of variable substitution, command substitution, and filename generation.</p> |
| Type | Built-in. |
| Input | None. |
| Output | Parameters are written to standard output. |
| Diagnostics | None. |
| Status | Status value 0 is always returned. |
| Option | <p>-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.</p> |
| Examples | <p>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.</p> <p>Echo {Status} Writes the current value of the {Status} variable; that is, the status of the last command executed.</p> <p>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".)</p> <p>Echo -n > EmptyFile If EmptyFile exists, this command deletes its contents; if the file doesn't exist, it is created.</p> |
| See also | Parameters and Quote commands. |

Eject—eject volumes

| | |
|--------------------|---|
| Syntax | <code>Eject [-m] volume...</code> |
| Description | <p>Flushes the volume, unmounts it, and then ejects it, if it is a floppy disk. A volume name must end with a colon (:). If <i>volume</i> is a number without a colon, it's interpreted as a drive number.</p> <p><i>Note:</i> 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 <code>-m</code> option. (See the File Manager chapter of <i>Inside Macintosh</i>.)</p> |
| Type | Built-in. |
| Input | None. |
| Output | None. |
| Diagnostics | Errors are written to diagnostic output. |
| Status | <p>The following status values may be returned:</p> <ul style="list-style-type: none">0 The disk was successfully ejected1 Syntax error2 An error occurred |
| Option | <code>-m</code> Leave the volume mounted. |
| Examples | <p><code>Eject Memos:</code> Ejects (and unmounts) the disk titled Memos.</p> <p><code>Eject 1</code> Ejects and unmounts the disk in drive 1 (the internal drive).</p> <p><code>Eject =:</code> Ejects and unmounts all volumes.</p> |
| See also | Mount, Unmount, and Volumes commands. |

Entab—convert runs of spaces to tabs

| | |
|--------------------|---|
| Syntax | Entab [<i>option...</i>] [<i>file...</i>] |
| Description | <p>Copies the specified text files to standard output, replacing runs of spaces with tabs. The default behavior of Entab is to do the following:</p> <ol style="list-style-type: none">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. Then 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. <p>Options are also provided for controlling the processing of blanks between quoted strings.</p> |
| 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 | <p>The following status codes may be returned to the Shell:</p> <ul style="list-style-type: none">0 Normal termination1 Parameter or option error2 Execution terminated |

Options

- 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 quote characters. *Quote...* is a string of one or more nonblank characters. If **-l** is specified, then **-r** must also be specified. Single quotes (') and double quotes (") 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 quotes. *Quote...* is a string of one or more nonblank characters. This is the default option; single quotes (') and double quotes (") 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, then **-l** must also be specified.
Note: Entab does not check that a particular left quote character matches a particular right quote 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, then 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 except for tab characters. Thus backspace characters may cause incorrect results.

The maximum width for an input line is 255 characters.

See also

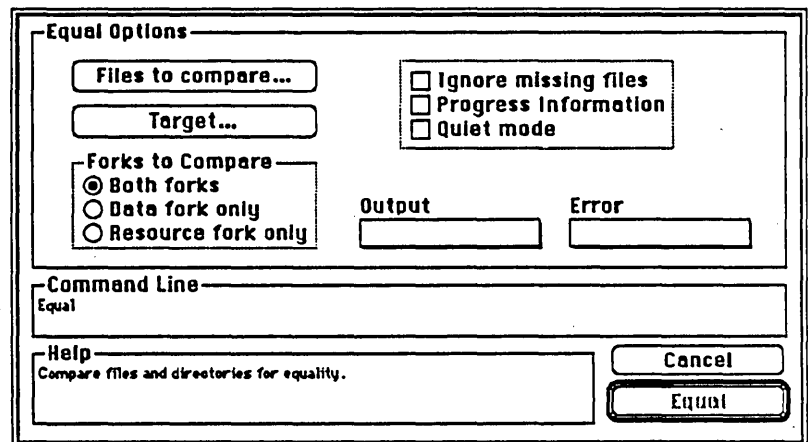
Tab command.

Equal—compare files and directories

| | |
|--------------------|--|
| Syntax | Equal [<i>option...</i>] <i>name...</i> <i>targetName</i> |
| Description | <p>Compares <i>name</i> to <i>targetName</i>. 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>-l</i> option ignores files in <i>targetName</i> that are not present in <i>name</i>.</p> <p>If <i>targetName</i> is a file, every <i>name</i> must also be a file. The specified files are compared with <i>targetName</i>.</p> <p>If <i>targetName</i> is a directory and <i>name</i> is a file, Equal checks in <i>targetName</i> for the file <i>name</i> and compares the two files. That is, the command</p> <pre>Equal File1 Dir1</pre> compares File1 with :Dir1:File1. <p>If more than one name is specified, Equal compares each name with the corresponding file or directory in <i>targetName</i>. All subdirectories are also compared. The command</p> <pre>Equal File1 Dir1 Dir2</pre> compares File1 with :Dir2:File1 and then compares Dir1 with :Dir2:Dir1. <p>If <i>targetName</i> is a directory, <i>name</i> is a directory, and only one name is specified, then the Equal command directly compares the two directories. That is, the command</p> <pre>Equal Dir1 Dir2</pre> 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 | <p>The following status values may be returned:</p> <ul style="list-style-type: none">0 Identical files1 Syntax error2 Inaccessible or missing parameter3 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] <i>volume...</i> |
| Description | <p>Initializes the specified volumes— the previous contents are destroyed. A volume name must end with a colon (:). If <i>volume</i> is a number without a colon, it's interpreted as a disk drive number.</p> <p>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.</p> |
| Type | Built-in. |
| Input | None. |
| Output | None. |
| Diagnostics | Errors are written to diagnostic output. |
| Status | <p>The following status values may be returned:</p> <ul style="list-style-type: none">0 Successful initialization1 Syntax error2 No such volume, or boot volume3 Errors during the initialization procedure |
| Options | <p>-y Answer "yes" to the confirmation dialog, causing initialization to begin immediately.</p> <p>-s Format the disk for single-sided use (that is, as a 400K, non-HFS disk).</p> |
| Examples | <p>Erase Reports: Initializes the volume titled Reports.</p> <p>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.</p> |

Evaluate—evaluate an expression

Syntax

Evaluate [*expression...*]

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 below.

The operators and precedence are mostly those of the C language; they're described below.

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 decimal number.

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 either decimal or hexadecimal integers representable by a 32-bit signed value. Hexadecimal numbers begin with either \$ or 0x. Every expression is computed as a 32-bit signed value. Overflows are ignored.

The pattern-matching operators =~ and !=~ are like == and != except that the right-hand side is a regular expression which is matched against the left-hand 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 below.)

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). <i>Note:</i> To redirect Evaluate's output (or diagnostic output), you'll need to enclose the Evaluate command in parentheses; otherwise, the > and ≥ symbols will be interpreted as expression operators, and an error will occur. (See the third example below.) |
| Diagnostics | Errors are written to diagnostic output. |
| Status | The following status values may be returned: 0 Valid expression 1 Invalid expression |
| Options | None. |

Examples

```
Evaluate (1+2) * (3+4)
```

Does the computation and writes the result to standard output.

```
Set lines `Evaluate {lines} + 1`
```

The Set command increments the value of the Shell variable {lines}—the Evaluate command enclosed in command substitution characters (`...`) is replaced by its output.

```
( 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-hand 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 <i>commandFile</i> |
| Description | <p>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 <i>script</i> are ignored. Any parameters to the enclosing script are available within <i>script</i>.</p> <p><i>Note:</i> If <i>script</i> 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.</p> |
| Type | Built-in. |
| Input | None. |
| Output | None. |
| Diagnostics | None. |
| Status | Execute returns the status returned by <i>script</i> . |
| Options | None. |
| Example | <pre>Execute "\${ShellDirectory}Startup</pre> <p>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.</p> |
| See also | <p>"Defining and Redefining Variables" in Chapter 5.</p> <p>"The Startup and UserStartup Files" in Chapter 5.</p> |

Exists—confirm the existence of a file or directory

| | |
|--------------------|---|
| Syntax | Exists [-d -f -w] [-q] <i>name</i> ... |
| Description | Determines the existence of the file or directory <i>name</i> . The options help you to distinguish between directories and files and different access permissions. The non-existence of <i>name</i> 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 standard output. |
| Status | The following status codes may be returned: 0 No error 1 Syntax error 2 Other error |
| Options | -d Check if <i>name</i> is a directory. -f Check if <i>name</i> is a file (as opposed to a directory). -w Check if the user has write access to the file <i>name</i> . A file cannot be written to if it is open or locked. -q Do not quote pathnames that are written to standard output. |
| Example | <pre>If Not ``Exists -d HD:dir`` NewFolder HD:dir End Duplicate *.c HD:dir</pre> <p>This example creates a new directory and copies all files ending with ".c" in current directory to this new directory.</p> |
| See also | Newer command. |

Exit—exit from a script

| | |
|--------------------|--|
| Syntax | Exit [<i>status</i>] [If <i>expression</i>] |
| Description | If the <i>expression</i> 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. <i>Status</i> 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 <i>expression</i> ” is omitted, the Exit is unconditional. (For a definition of <i>expression</i> , refer to the description of the Evaluate command.) |
| Type | Built-in. |
| Input | None. |
| Output | None. |
| Diagnostics | Errors are written to diagnostic output. |
| Status | If <i>status</i> 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 <i>name...</i>] |
| Description | <p>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.)</p> <p><i>Note:</i> 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.)</p> <p>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.)</p> |
| 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 values: 0 No errors 1 Syntax error |
| Options | <p>-r Reverse the sense of the output, causing Export to generate Unexport commands for all exported variables.</p> <p>-s Suppress the printing of "Export" before the exported variables.</p> |
| Example | <pre>Set AIncludes "{MPW}AIncludes:" Export AIncludes</pre> <p>Defines the variable {AIncludes} as the pathname "{MPW}AIncludes:", and makes it available to scripts and programs.</p> |
| 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 values:
- 0 Normal termination
 - 1 Parameter or option error
 - 2 Execution terminated
- 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

|                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                          |                                                    |           |                           |           |                                           |           |                                                                                                                                                                                    |                        |                                                                                          |           |                                                                                                                  |           |                                                                                                                      |           |                                                                                       |           |                                                                                                                                               |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|----------------------------------------------------|-----------|---------------------------|-----------|-------------------------------------------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|------------------------------------------------------------------------------------------|-----------|------------------------------------------------------------------------------------------------------------------|-----------|----------------------------------------------------------------------------------------------------------------------|-----------|---------------------------------------------------------------------------------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>            | Files [ <i>option...</i> ] [ <i>slugname...</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                          |                                                    |           |                           |           |                                           |           |                                                                                                                                                                                    |                        |                                                                                          |           |                                                                                                                  |           |                                                                                                                      |           |                                                                                       |           |                                                                                                                                               |
| <b>Description</b>       | 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                          |                                                    |           |                           |           |                                           |           |                                                                                                                                                                                    |                        |                                                                                          |           |                                                                                                                  |           |                                                                                                                      |           |                                                                                       |           |                                                                                                                                               |
| <b>Type</b>              | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                          |                                                    |           |                           |           |                                           |           |                                                                                                                                                                                    |                        |                                                                                          |           |                                                                                                                  |           |                                                                                                                      |           |                                                                                       |           |                                                                                                                                               |
| <b>Input</b>             | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                          |                                                    |           |                           |           |                                           |           |                                                                                                                                                                                    |                        |                                                                                          |           |                                                                                                                  |           |                                                                                                                      |           |                                                                                       |           |                                                                                                                                               |
| <b>Output</b>            | File information is written to standard output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                          |                                                    |           |                           |           |                                           |           |                                                                                                                                                                                    |                        |                                                                                          |           |                                                                                                                  |           |                                                                                                                      |           |                                                                                       |           |                                                                                                                                               |
| <b>Diagnostics</b>       | Errors and warnings are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                          |                                                    |           |                           |           |                                           |           |                                                                                                                                                                                    |                        |                                                                                          |           |                                                                                                                  |           |                                                                                                                      |           |                                                                                       |           |                                                                                                                                               |
| <b>Status</b>            | Files may return the following status values:<br>0 All names were processed successfully<br>1 Syntax error<br>2 An error occurred                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                          |                                                    |           |                           |           |                                           |           |                                                                                                                                                                                    |                        |                                                                                          |           |                                                                                                                  |           |                                                                                                                      |           |                                                                                       |           |                                                                                                                                               |
| <b>Options</b>           | <table><tr><td><b>-c</b> <i>creator</i></td><td>List only those files with the given file creator.</td></tr><tr><td><b>-d</b></td><td>List subdirectories only.</td></tr><tr><td><b>-f</b></td><td>Give full pathnames for all files listed.</td></tr><tr><td><b>-i</b></td><td>Treat directories on the command line as files (ignore differences). That is, don't list the contents of directories, just list the directory and any other information requested.</td></tr><tr><td><b>-m</b> <i>count</i></td><td>Multi-column output. This option is not valid if specified with <b>-l</b> or <b>-x</b>.</td></tr><tr><td><b>-n</b></td><td>No header in the long or extended format. Without the <b>-l</b> or <b>-x</b> option, this option has no meaning.</td></tr><tr><td><b>-q</b></td><td>Don't quote names in the output. Normally, the Files command quotes names that contain spaces or special characters.</td></tr><tr><td><b>-r</b></td><td>Recursively list subfolders encountered; that is, list every file in every directory.</td></tr><tr><td><b>-s</b></td><td>Suppress the printing of directory names. Useful when combined with the <b>-r</b> option to get listing of all files (excluding directories).</td></tr></table> | <b>-c</b> <i>creator</i> | List only those files with the given file creator. | <b>-d</b> | List subdirectories only. | <b>-f</b> | Give full pathnames for all files listed. | <b>-i</b> | Treat directories on the command line as files (ignore differences). That is, don't list the contents of directories, just list the directory and any other information requested. | <b>-m</b> <i>count</i> | Multi-column output. This option is not valid if specified with <b>-l</b> or <b>-x</b> . | <b>-n</b> | No header in the long or extended format. Without the <b>-l</b> or <b>-x</b> option, this option has no meaning. | <b>-q</b> | Don't quote names in the output. Normally, the Files command quotes names that contain spaces or special characters. | <b>-r</b> | Recursively list subfolders encountered; that is, list every file in every directory. | <b>-s</b> | Suppress the printing of directory names. Useful when combined with the <b>-r</b> option to get listing of all files (excluding directories). |
| <b>-c</b> <i>creator</i> | List only those files with the given file creator.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                          |                                                    |           |                           |           |                                           |           |                                                                                                                                                                                    |                        |                                                                                          |           |                                                                                                                  |           |                                                                                                                      |           |                                                                                       |           |                                                                                                                                               |
| <b>-d</b>                | List subdirectories only.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                          |                                                    |           |                           |           |                                           |           |                                                                                                                                                                                    |                        |                                                                                          |           |                                                                                                                  |           |                                                                                                                      |           |                                                                                       |           |                                                                                                                                               |
| <b>-f</b>                | Give full pathnames for all files listed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                          |                                                    |           |                           |           |                                           |           |                                                                                                                                                                                    |                        |                                                                                          |           |                                                                                                                  |           |                                                                                                                      |           |                                                                                       |           |                                                                                                                                               |
| <b>-i</b>                | Treat directories on the command line as files (ignore differences). That is, don't list the contents of directories, just list the directory and any other information requested.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                          |                                                    |           |                           |           |                                           |           |                                                                                                                                                                                    |                        |                                                                                          |           |                                                                                                                  |           |                                                                                                                      |           |                                                                                       |           |                                                                                                                                               |
| <b>-m</b> <i>count</i>   | Multi-column output. This option is not valid if specified with <b>-l</b> or <b>-x</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                          |                                                    |           |                           |           |                                           |           |                                                                                                                                                                                    |                        |                                                                                          |           |                                                                                                                  |           |                                                                                                                      |           |                                                                                       |           |                                                                                                                                               |
| <b>-n</b>                | No header in the long or extended format. Without the <b>-l</b> or <b>-x</b> option, this option has no meaning.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                          |                                                    |           |                           |           |                                           |           |                                                                                                                                                                                    |                        |                                                                                          |           |                                                                                                                  |           |                                                                                                                      |           |                                                                                       |           |                                                                                                                                               |
| <b>-q</b>                | Don't quote names in the output. Normally, the Files command quotes names that contain spaces or special characters.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                          |                                                    |           |                           |           |                                           |           |                                                                                                                                                                                    |                        |                                                                                          |           |                                                                                                                  |           |                                                                                                                      |           |                                                                                       |           |                                                                                                                                               |
| <b>-r</b>                | Recursively list subfolders encountered; that is, list every file in every directory.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                          |                                                    |           |                           |           |                                           |           |                                                                                                                                                                                    |                        |                                                                                          |           |                                                                                                                  |           |                                                                                                                      |           |                                                                                       |           |                                                                                                                                               |
| <b>-s</b>                | Suppress the printing of directory names. Useful when combined with the <b>-r</b> 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: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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Find [-c <i>count</i> ] <i>selection</i> [ <i>window</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Description</b> | <p>Creates a selection in <i>window</i>. 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.</p> <p><i>Selection</i> is a selection as defined in Chapter 6 and in Appendix B.</p> <p><i>Note:</i> 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.</p> <p>All searches are case insensitive by default. You can specify case-sensitive searches by first setting the Shell variable (CaseSensitive) to a nonzero value. (You can automatically set (CaseSensitive) by checking Case Sensitive in the dialog boxes displayed by the Find and Replace menu items.)</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Status</b>      | <p>The following status values may be returned:</p> <ul style="list-style-type: none"><li>0 At least one instance of the selection was found</li><li>1 Syntax error</li><li>2 Any other error</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Option</b>      | <p><b>-c <i>count</i></b> For a count of <i>n</i>, find the <i>n</i>th occurrence of the selection.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Examples</b>    | <p>Find •<br/>Positions the insertion point at the beginning of the target window.</p> <p>Find -c 5 /procedure/ Sample.p<br/>Selects the fifth occurrence of "procedure" in the window Sample.p.</p> <p>Find 332<br/>Selects line 332 in the target window.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>See also</b>    | <p>"Selections" and "Pattern Matching" in Chapter 6.<br/>"Find Menu" in Chapter 3.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

---

---

## Font—set font characteristics

|                    |                                                                                                                                                                                                                                                                                                                                     |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Font <i>fontname</i> <i>fontsize</i> [ <i>window</i> ...]                                                                                                                                                                                                                                                                           |
| <b>Description</b> | Change the font family and point size of all text in <i>window</i> to <i>fontname</i> and <i>fontsize</i> . Both <i>fontname</i> and <i>fontsize</i> are required. It's an error to specify a window that doesn't exist. If no windows are specified, the command operates on the target window (the second window from the front). |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                           |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                               |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                               |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                            |
| <b>Status</b>      | Font may return the following status values:<br>0 Successful completion<br>1 Error in parameters<br>2 An illegal fontname or fontsize was specified                                                                                                                                                                                 |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                               |
| <b>Example</b>     | Font Monaco 12<br>Changes the font of the target window to Monaco 12 point.                                                                                                                                                                                                                                                         |

---

---

## For...—repeat commands once per parameter

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | For <i>name</i> In <i>word...</i><br><i>command...</i><br>End                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Description</b> | <p>Executes the list of commands once for each word from the “In <i>word...</i>” list. The current word is assigned to variable <i>name</i>, and you can therefore reference it in the list of commands by using the notation {<i>name</i>}. You must end each line with either a return character (as shown above) or with a semicolon ( ; ).</p> <p>The Break command can be used to terminate the loop. The Continue command can be used to terminate the current iteration of the loop.</p> <p>The pipe specification ( ), conditional command terminators (&amp;&amp; and   ), and input/output specifications (&lt;, &gt;, &gt;&gt;, ≥, and ≥≥) may appear following the End, and apply to all of the commands in the list.</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Status</b>      | <p>The following status values may be returned:</p> <ul style="list-style-type: none"><li>0 The list of words or list of commands was empty</li><li>-3 There was an error in the parameters to For</li></ul> <p>Otherwise, the status of the last command executed is returned.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Options</b>     | 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 that 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.

---

---

## GetErrorText—display text for system error numbers

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>GetErrorText [-f filename] [-s filename] [-n] [-p] errnbr[,insert,...] ...</code><br><code>GetErrorText -i idnbr ...</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Description</b> | <p>Displays the error messages corresponding to a set of specified error numbers or ID numbers. By default, <code>GetErrorText</code> assumes the error numbers correspond to Macintosh Operating System system error numbers. The file <code>SysErrs.Err</code> 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 <code>-s</code> option.</p> <p>In addition to system errors, some tools have their own error message files. For example, in the case of the Assembler, it is in the data resource fork of <code>Asm</code> itself. For such tools, you can display the error messages corresponding to tool error numbers by specifying the <code>-f</code> option. In this case, you may specify sample inserts, along with the error numbers, for error messages that take inserts, as shown above.</p> <p><code>GetErrorText</code> can also display the meanings of the ID numbers reported by the System Error Handler in alert dialog boxes. The <code>-i</code> option is used for this purpose.</p> |
| <b>Type</b>        | Tool.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Input</b>       | All input is specified through the parameters.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Output</b>      | The error messages are written to the standard output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Diagnostics</b> | Errors are written to the diagnostic file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Status</b>      | <p><code>GetErrorText</code> may return the following status values:</p> <ul style="list-style-type: none"><li>0 Normal termination</li><li>1 Parameter or option error</li><li>2 Execution terminated</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

|                |                           |                                                                                                                                                        |
|----------------|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Options</b> | <b>-f <i>filename</i></b> | A tool's error message filename. Either <b>-f</b> or <b>-s</b> , but not both, may be specified.                                                       |
|                | <b>-i <i>fdnbr</i></b>    | Report the meaning of the specified System Error Handler ID number.                                                                                    |
|                | <b>-s <i>filename</i></b> | The error message file name for a system error. Either <b>-f</b> or <b>-s</b> , but not both, may be specified. The default is <b>-s SysErrs.Err</b> . |
|                | <b>-n</b>                 | Do not generate error numbers as part of the error messages. This option is ignored if system errors are displayed.                                    |
|                | <b>-p</b>                 | 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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | GetFileName [-t TYPE]...   -p   -d] [-q] [-m <i>message</i> ] [-b <i>buttontitle</i> ] [ <i>pathname</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Description</b> | 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 <i>pathname</i> if specified. If <i>pathname</i> includes a local filename and if SFPutFile is called, the local filename is used as the default filename. See the examples.                                                                                                                                                                                                                                                                                                                      |
| <b>Type</b>        | Tool.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Output</b>      | 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Diagnostics</b> | Parameter errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Status</b>      | The following status values may be returned:<br>0 User specified a file and no errors occurred<br>1 Parameter or option error<br>2 System error<br>4 User canceled the standard file dialog box                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Options</b>     | <p><b>-p</b> Display an SFPutFile dialog box.</p> <p><b>-d</b> Display an SFGetFile dialog for selecting a directory.</p> <p><b>-m <i>msg</i></b> Specify a prompt message.</p> <p><b>-b <i>buttontitle</i></b> Specify the title for the default button in the various dialog boxes. If this option is not specified, <i>Open</i> is used in the standard SFGetFile dialog box, <i>Save</i> is used in the standard SFPutFile dialog box, and <i>Directory</i> is used in the directory SFGetFile dialog box.</p> <p><b>-q</b> Suppress quoting the filename written to standard output.</p> <p><b>-t <i>type</i></b> Specify a type to use in filtering the SFGetFile. Up to four types may be specified.</p> |

**Examples**

```
open `GetFileName -t TEXT {pinterfaces}`
```

Opens the text file in directory {pinterfaces} chosen in SFGGetFile 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.

---

---

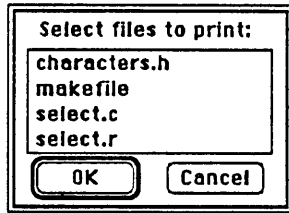
## GetListItem—display items for selection in a dialog box

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | GetListItem [ <i>option...</i> ] [ <i>items...</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Description</b> | Takes the items on the command line (or, if no items are present on command line, then 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Type</b>        | Tool.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Input</b>       | Reads standard input for the items if none are specified on the command line.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Output</b>      | Selected items are written to standard output if the OK button is clicked.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Status</b>      | GetListItem may return the following status values:<br>0 No errors<br>1 Syntax error (bad option)<br>2 Cancel button was clicked                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Options</b>     | <p><b>-d</b> <i>item</i>      <i>Item</i> is entered as an element in the list and comes up selected. This option may be specified more than once.</p> <p><b>-m</b> <i>message</i>    Display <i>message</i> above the list of items.</p> <p><b>-q</b>                Don't quote items in the output.</p> <p><b>-r</b> <i>rows</i>         Make the list with this many rows.</p> <p><b>-w</b> <i>width</i>        Make the list this many pixels wide.</p> <p><i>Note:</i> GetListItem uses the <b>-r</b> and <b>-w</b> values only as a guideline. For example, if the value given for <i>rows</i> 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 <b>-r</b> or <b>-w</b> arguments are out of range. Rather, it makes a reasonable guess at a value.</p> |

## Examples

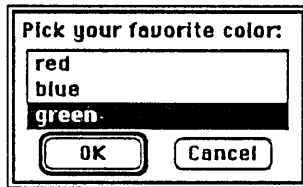
```
print `files -t TEXT | GetListItem -m "Select files to print:"`
```

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 pre-selected, 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:

|                          |                                                           |
|--------------------------|-----------------------------------------------------------|
| <i>commandName</i>       | Information about <i>commandName</i>                      |
| <code>commands</code>    | A list of all MPW commands                                |
| <code>expressions</code> | A summary of expressions                                  |
| <code>patterns</code>    | A summary of pattern specifications (regular expressions) |
| <code>selections</code>  | A summary of selection operators                          |
| <code>characters</code>  | A summary of MPW Shell special characters                 |
| <code>shortcuts</code>   | A summary of MPW shortcuts                                |

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:

|                           |                                                                                         |
|---------------------------|-----------------------------------------------------------------------------------------|
| <code>[ optional ]</code> | Square brackets mean that the enclosed elements are optional.                           |
| <code>repeated...</code>  | Ellipses indicate that the preceding item can be repeated one or more times.            |
| <code>a   b</code>        | A vertical bar indicates an either/or choice.                                           |
| <code>(grouping)</code>   | Parentheses indicate grouping (useful with " " and "...").                              |
| <code>&lt; input</code>   | If <i>input</i> is not specified, the command reads from standard input.                |
| <code>&gt; output</code>  | The command writes to standard output.                                                  |
| <code>≥ progress</code>   | Progress information is written to diagnostic output (with the <code>-p</code> 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, then the first entry is written to standard output. Here is an excerpt 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...
```

|                    |                                                                                                                                                                                                                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                         |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                             |
| <b>Output</b>      | Command information is written to standard output.                                                                                                                                                                                                                                |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                          |
| <b>Status</b>      | The following status values may be returned: <ul style="list-style-type: none"> <li>0 Information was found for the given command</li> <li>1 Syntax error</li> <li>2 A command could not be found, or error in parameters</li> <li>3 The help file could not be opened</li> </ul> |
| <b>Option</b>      | <b>-f <i>helpFile</i></b> Specify help file to be searched. (A help file is an ordinary MPW text file.) The default file is MPW.Help.                                                                                                                                             |

**Example**

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
```

---

---

## If...—conditional command execution

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | If <i>expression</i><br><i>command...</i><br>[ Else If <i>expression</i><br><i>command...</i> ] ...<br>[ Else<br><i>command...</i> ]<br>End                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Description</b> | <p>Executes the list of commands following the first <i>expression</i> 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.</p> <p>The pipe specification (<code> </code>), conditional command terminators (<code>&amp;&amp;</code> and <code>  </code>), and input/output specifications (<code>&lt;</code>, <code>&gt;</code>, <code>&gt;&gt;</code>, <code>≥</code>, and <code>≥≥</code>) may appear following the End, and apply to all of the commands in the list.</p> <p>For a definition of <i>expression</i>, see the description of the Evaluate command.</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Status</b>      | 0   None of the lists of commands were executed<br>-1   Invalid expression                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|                    | Otherwise, it returns the value returned by the last command executed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Options</b>     | 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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Lib [ <i>option...</i> ] <i>objectFile...</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Description</b> | <p>Combines the specified object files into a single file. By convention, input files end in the suffix “.o”, which must be present. In addition, input files must have type 'OBJ' and creator 'MPS'.</p> <p>Lib is used for the following:</p> <ul style="list-style-type: none"><li>□ Combining object code from different languages into a single file.</li><li>□ 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.</li><li>□ Deleting unneeded modules (with the <b>-dm</b> option), changing segmentation (the <b>-sg</b> and <b>-sn</b> options), or changing the scope of a symbol from external to local (the <b>-dn</b> option). (These options are useful when you construct a specialized library for linking a particular program.)</li></ul> <p>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.</p> <p>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.</p> <p>See “Library Construction” in Chapter 10 for a detailed discussion of the behavior and use of Lib.</p> |
| <b>Type</b>        | Tool.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Input</b>       | Lib does not read standard input.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Output</b>      | 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 <b>-o</b> option. The output file is given type 'OBJ' and creator 'MPS'.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Diagnostics</b> | Errors and warnings are written to diagnostic output. Progress information is also written to the diagnostic file if you specify the <b>-p</b> option.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Status</b>      | Lib may return the following status values:<br>0 No problem<br>2 Fatal error<br>3 User interrupt                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

## Options

- b** Do a big execution of Lib, that is, **-bf** and **-bs 4** options.
- bf** Allow a big number of files; that is, keep only one input file open at a time. If Lib fails with a “too many files open” message, use this option.
- bs *nn*** Set the buffer size for input to *nn* blocks (512 bytes each). If Lib fails with a “heap error” or “out of memory” message, try this option. Values for *nn* must be between 2 and 64. (The default is 16.)  
*Note:* Numeric values can be specified as decimal constants, or as hex constants preceded by a “\$”.
- 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. *name* may be either an external module or entry-point name. For each entry-point *name*, the entire module containing the entry point is deleted, together with all other entry names in the module. The contents of the module and all entry points are removed from the output file.  
*Note:* References to names deleted in this way will persist as references “by name.” That is, if the references are from active code, they’ll need to 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 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. **-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 will still be 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 segment names. 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 **-sn** 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.)

**-w** Suppress warning messages.

### 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 F.

---

---

## Line—find a line number

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Line <i>n</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Description</b> | <p>Line finds line <i>n</i> in the target window. Parameter <i>n</i> is usually an integer, but may be any selection expression. The target window becomes the active (frontmost) window.</p> <p>Line is a script containing these two commands:</p> <pre>Find "{1}" "{target}"      # Find line <i>n</i> in the target window Open "{target}"           # Bring the target window to the top</pre>                                                                                                                                                                                                                                                                                                                                              |
| <b>Type</b>        | Script.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Status</b>      | <p>Status values can be returned by either the Find or the Open commands that make up the Line script:</p> <ul style="list-style-type: none"><li>0 No errors</li><li>1 Syntax error</li><li>2 No target window; parameter not a number; other error</li><li>3 System error</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Examples</b>    | <pre>Line 123</pre> <p>Finds line 123 in the target window and makes the target window the new active window.</p> <pre>### Undefined symbol: length       File "Count.c"; Line 75</pre> <p>The File and Line commands above are part of an error message produced by the C Compiler. The Assembler and Pascal Compilers produce errors when using similar formats. You can execute such error messages to find the line that contains the error.</p> <p>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.</p> |
| <b>See also</b>    | 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 ' and creator 'MPS ', and must end in the suffix ".o". Linked segments from the input object files are placed in code resources in the resource fork of the output file. The default output file name is **Link.Out**—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 9 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; you can create unnamed resources with the **-rn** option.

The Linker executes in three 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, the Linker produces a linker map or cross-reference listing. The Linker also eliminates unused code.
- **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.)

**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 file name is Link.Out—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, then the Linker sets the modification date on the file to "zero" (Jan. 1, 1904, 12:00 a.m.). This guarantees that Make will recognize that the file needs to be relinked.

If you specify the **-l** option, the Linker writes a **location map** to standard output. The map is produced in location ordering, that is, sorted by *segNum*, *segOffset*. The format is divided into several fields:

```
name segName segNum, segOffset [@JTOffset] [#] [E] [C] [fileNum, defOffset]
```

See Chapter 10 for more 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** The following status values may be returned:

```
0 No problem
2 Fatal error
3 User interrupt
```

**Options** *Note:* Numeric values for options can be specified as decimal constants, or as hex constants preceded by the dollar sign character (\$).

- b** Do a big link; that is, do both **-bf** and **-bs 4** options.
- bf** Allow a big number of files; that is, keep only one input file open at a time. If a link fails with a “too many files open” message, use this option.
- bs blocks** Set the buffer size for the Linker to *blocks* blocks (512 bytes each). If a link fails with a “heap error” or “out of memory” message, try this option. Values for *blocks* must be between 2 and 64. (The default is 16.)
- 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 on output. 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').
- l** Write a location-ordered map to standard output. Usually, this option will be used with output redirection in effect. For example,  

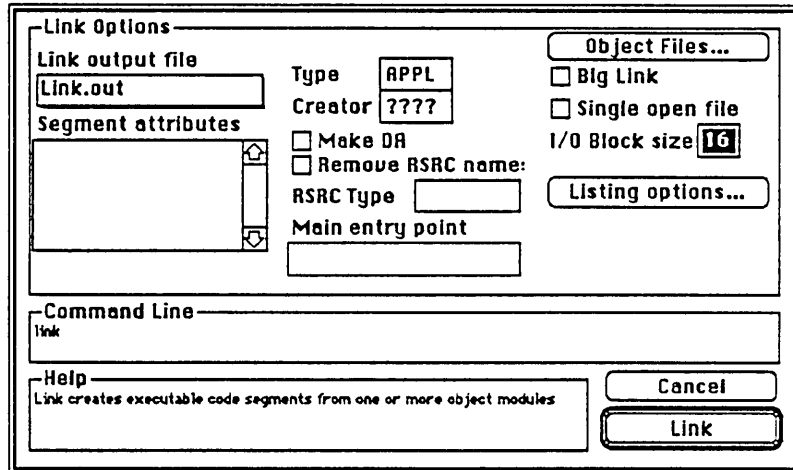
```
Link ObjFile -l > MyMapFile
```
- la** List anonymous symbols in the location map (with the **-l** option). The default is to omit anonymous symbols from the map.
- lf** In the location map data (**-l** option), include the location where symbols are defined in the input file, that is, the file number and byte offset of the module or entry-point record. (These records are discussed in detail in Appendix F.) The default is to omit the symbol definition location.

- 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 the Linker's active-code analysis feature, then the main entry point should be the first byte of code in the first Linker input file. (A desk accessory has no jump table.)
- ma *name = alias*** "Module alias"—give the module or entry-point *name* the new 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.
- o *outputFile*** Place the Linker output in *outputFile*. If no **-o** option is specified, the default output filename is Link.Out. The **-o** option uses the file in the system folder (or next level) if:
1. the output file is not a full pathname, and
  2. it is not in the prefix folder, and
  3. it does not exist in the system folder or the volume root level.
- For more information on the Macintosh search path conventions see *Inside Macintosh*, Volume IV.
- opt** Optimizes object Pascal. This option eliminates the need for the Optimize tool distributed with MacApp.
- 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, then all segments except 0 and 1 are given the attribute value *nn*. (If you intend to set the attributes of all segments, then 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).
- 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, the Linker uses PC-relative offsets, and attempts to edit JSR, JMP, LEA, or PEA instructions from A5-relative to PC-relative addressing mode. Other instructions will generate an error message.
- sg *newSeg=oldSeg1[,oldSeg2 ]...***  
Change segment names. All code in the segments named *oldSeg1*, *oldSeg2*,... is placed in the segment named *newSeg*. If no *oldSeg* (and no =) is specified, the Linker will map all code 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 two major differences between **-sn** and **-sg**:
- **-sn** allows an arbitrary string for the new name, whereas **-sg** is intended only for identifiers separated by commas.
  - 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".
- ss *size*** Change the maximum segment size to *size*. The default value is 32760 (32K less a few overhead bytes). The value *size* can be any value greater than 32760.
- 64K ROM note:* Caution! Applications with segments greater than 32K in size may not load correctly on Macintoshes with 64K ROMs.
- 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 @
-l -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 will be an application, which 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. The Linker location-ordered listing is placed in the file ROMLocListing.

For additional examples, see "Linking" in Chapter 10 and the makefiles in the Examples folders for the languages you are using.

See also

Lib command and Appendix F, "Object File Format."

"Linking" and "More About Linking" in Chapters 9 and 10.

The Segment Loader and the Resource Manager chapters in *Inside Macintosh*.

Inside Macintosh, Volume IV, for information on the 128K ROM, System Folder, and Finder.

Loop...End—repeat command list until Break

| | |
|--------------------|---|
| Syntax | Loop <i>command...</i> End |
| Description | <p>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. Return characters must appear as shown above, or be replaced with semicolons (;).</p> <p>The pipe specification (), conditional command terminators (&& and), and input/output specifications (<, >, >>, ≥, and ≥≥) may appear following the End, and apply to all of the commands in the list.</p> |
| Type | Built-in. |
| Input | None. |
| Output | None. |
| Diagnostics | None. |
| Status | Loop returns the status of the last command executed. |
| Options | None. |
| Example | <p>The command file below runs a command several times, once for each parameter.</p> <pre>### 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</pre> <p>Notice that Shift is used to step through the parameters, and that Break ends the loop when all the parameters have been used.</p> |
| 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 then the first target on the left side of a dependency rule in the makefile will be 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, and 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 “Using Make” in Chapter 10 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 for lower-level dependencies that need to be rebuilt are issued first.

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 values 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 output unconditionally 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 will find all the roots (that is, the top level targets) of the dependency graph. (See the **-s** option.) If a target is specified, **-r** will find 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 outputting 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 the 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 will indicate if targets do not exist, whether or not 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`, printing progress information. `Sample`'s dependency relations are described in the makefile `:AExamples:MakeFile`:

```
Sample      ff      Sample.r
Rez Sample.r -o Sample -a
SetFile -a B Sample -c ASMP -t APPL #set bundle bit

Sample      ff      Sample.r Sample.a.o
Link Sample.a.o -o Sample

Sample.a.o  f      Sample.a
Asm Sample.a
```

The **f** (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 side are newer, the subsequent Shell commands are written to standard output. (See Chapter 10 for details.)

See also

"Using Make" in Chapter 10, for the format of a makefile, examples, and other information about using Make.

Makefiles for building sample programs are contained in the Examples folders:

- `:AExamples:Makefile`
- `:PEExamples:Makefile`
- `:CEExamples:Makefile`

MakeErrorFile—create error message textfile

- Syntax** `MakeErrorFile [option...] [file...]`
- Description** `MakeErrorFile` creates error message files that are 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, 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 will be written to standard output. The error file output is usually written to a file with the same name as the input file but with a ".Err" extension (unless the `-o` option was used to specify the output name). If no input file was specified, then by default the input comes from standard input and the default error output file name is "Out.Err".
- Diagnostics** Errors and warnings are written to diagnostic output.
- Status** The following status values 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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Mark [ <i>-y</i>   <i>-n</i> ] <i>selection name</i> [ <i>window</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Description</b> | <p>Mark assigns the marker <i>name</i> to the range of text specified by the <i>selection</i> in <i>window</i>. 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 <i>window</i> is the current active window. A marker is associated to 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.</p> <p>A dialog box requests confirmation if the marker name conflicts with an existing marker name. The <i>-y</i> or <i>-n</i> option can be used in scripts to avoid this interaction.</p> <p>Deletion and insertion operations affect markers according to these rules:</p> <ul style="list-style-type: none"><li><input type="checkbox"/> 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.</li><li><input type="checkbox"/> 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 10 characters to the inside of a marker’s range will increase the range of the marker by 10 characters. Another way to say this is: A marker has responsibility for all the characters added to (or deleted from) its range.</li><li><input type="checkbox"/> Any deletion that totally encloses a marker will delete the marker.</li></ul> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Status</b>      | <p>The following status values may be returned:</p> <ul style="list-style-type: none"><li>0 No errors</li><li>1 Syntax error</li><li>2 Error in processing</li><li>3 System error</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Options</b>     | <p><i>-y</i> Answer “yes” to any confirmation dialog that occurs, causing the old marker to be replaced with the new marker.</p> <p><i>-n</i> Answer “no” to any confirmation dialog that occurs, so that the old marker is left intact.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

**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

|                    |                                                                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Markers [ -q ] <i>window</i> }                                                                                                                                           |
| <b>Description</b> | Markers prints the names of all markers associated with <i>window</i> . The names are written one per line, and are ordered from the beginning to the end of the window. |
| <b>Type</b>        | Tool.                                                                                                                                                                    |
| <b>Input</b>       | None.                                                                                                                                                                    |
| <b>Output</b>      | The list of marker names is written to standard output.                                                                                                                  |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                 |
| <b>Status</b>      | The following status values may be returned:<br>0 No errors<br>1 Syntax error<br>2 Error in processing<br>3 System error                                                 |
| <b>Options</b>     | -q Do not quote marker names that contain special characters. (The default is to quote names with spaces or other special characters.)                                   |
| <b>Example</b>     | Markers "(Target)"<br>Lists all markers associated with the target window.                                                                                               |
| <b>See also</b>    | "Mark Menu" in Chapter 3.<br>"Markers" in Chapter 6.                                                                                                                     |

---

---

## MDSCvt—convert MDS Assembler source

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | MDSCvt [ <i>option ...</i> ] [ <i>file ...</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Description</b> | <p>Converts the specified Macintosh MC68000 Development System (MDS) Assembler source files to the syntax required by the MPW Assembler. The following elements are converted:</p> <ul style="list-style-type: none"><li><input type="checkbox"/> tokens within statements</li><li><input type="checkbox"/> special tokens within macros</li><li><input type="checkbox"/> directives</li></ul> <p>For a description of these conversions, refer to “MDS Conversion” in Appendix E of the <i>MPW Assembler 2.0 Reference</i>.</p>                                                                                                                                                                                     |
| <b>Type</b>        | Tool.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Input</b>       | Standard input is converted if no filenames are specified. If the <b>-main</b> , <b>-g</b> , or <b>-l</b> option is used, only one filename should be specified.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Output</b>      | If input is from the standard input file, the converted output is written to standard output. If the input file name is Name, the converted output is written to Name.a. The <b>-n</b> , <b>-prefix</b> , and <b>-suffix</b> options let you modify the naming conventions for the output file.                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Diagnostics</b> | Parameter errors and progress information are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Status</b>      | <p>The following status values may be returned:</p> <ul style="list-style-type: none"><li>0 Normal termination</li><li>1 Parameter or option error</li><li>2 Execution aborted</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Options</b>     | <ul style="list-style-type: none"><li><b>-d</b> Detab the input. All tabs are removed and replaced with spaces. The default setting is 8 spaces; this value can be changed with the <b>-t</b> option.</li><li><b>-e</b> Detab the input (like the <b>-d</b> option), and entab the output as a function of the tab setting (either 8, or the value specified with the <b>-t</b> option).</li><li><b>-f directivesFile</b> Set the case (upper/lower) of directives according to the entries in <i>directivesFile</i>. The file MDSCvt.Directives is supplied for this purpose; you can edit it to change the capitalization. If you don't use this option, then all directives are converted to uppercase.</li></ul> |

- g *globals*** Convert a main program source and reserve globals space below A5. *Globals* may be specified in decimal or hexadecimal (by preceding the value with a \$). The value specified must be negative. For example,
  - g -512
  - g \$200
- i** Convert include files. No PROC or MAIN or END will be generated by MDSCvt.
- m** Do *not* insert MDS-compatible mode-setting directives (BLANKS ON and STRING ASIS) into the converted source.
- main** Convert a main program source. The conversion is done to make the file look like the main code and data modules. Only one file should be converted when using this option.
- n** Do not add the ".a" extension to the input filename to produce the output filename. (If you use this option, you must also specify either **-prefix** or **-suffix**.)
- p** Write MDSCvt's version information and conversion status to diagnostic output.
- pre[fix] *string*** If the input filename is "Name", the output filename is produced by prefixing the *string* to Name, that is, "*string*Name.a". (You can suppress the ".a" suffix by using the **-n** option, or change it by using the **-sufflx** option.)
- suf[fix] *string*** If the input filename is "Name", the output filename is produced by appending the *string* to the filename, that is, "Name*string*". The default suffix is ".a".
- t *tabSetting*** Set the tab value for input and output files to *tabSetting* value (2 to 255). The default setting is assumed to be 8.
- u *c*** When MDSCvt detects a name in the opcode field that is the same as an MPW directive, it appends the character *c* to make the name unique. (The default character is #.)
- ! *identifier*** Convert a main program source and define the main program's entry point by the specified identifier. This option corresponds to the MDS Linker's ! command.

### Example

```
MDSCvt -t 8 MDSFile1.Asm MDSFile2.Asm
```

Convert MDS Assembler source files MDSFile1.Asm and MDSFile2.Asm to MPW Assembler source files MDSFile1.Asm.a and MDSFile2.Asm.a. The **-t** option sets the tab setting for both files to 8, and entabs the output files based on that value. It is assumed that neither file is a main program because the **-main** option has not been specified. If either file is a main program, then the **-main** option should be specified and only that file should be specified as input to MDSCvt.

**Limitations** See Appendix E in the *MPW Assembler 2.0 Reference* for details of conversions that can and cannot be done with MDSCvt.

**See also** Appendix E, "MDS Conversion," in the *MPW Assembler 2.0 Reference*.

---

---

## Mount—mount volumes

|                    |                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Mount <i>drive</i> ...                                                                                                                                                                                                                                                                                                                            |
| <b>Description</b> | <p>Mounts the disks in the specified drives, making them accessible to the file system. <i>Drive</i> is the drive number.</p> <p>Mounting is normally automatic when a disk is inserted. The Mount command is needed for mounting multiple hard disks, which cannot be “inserted,” or if a volume has been unmounted via the Unmount command.</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                         |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                             |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                             |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                          |
| <b>Status</b>      | <p>The following status values may be returned:</p> <ul style="list-style-type: none"><li>0 The disk was mounted</li><li>1 Syntax error</li><li>2 An error occurred</li></ul>                                                                                                                                                                     |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                             |
| <b>Example</b>     | <p>Mount 1</p> <p>Mounts the disk in drive 1 (the internal drive).</p>                                                                                                                                                                                                                                                                            |
| <b>See also</b>    | Unmount and Volumes commands.                                                                                                                                                                                                                                                                                                                     |

---

---

## Move—move files and directories

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>Move [-y   -n   -c] [-p] name... targetName</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Description</b> | <p>Moves <i>name</i> to <i>targetName</i>. (<i>Name</i> and <i>targetName</i> are file or directory names.) If <i>targetName</i> is a directory, then one or more objects (files and/or directories) are moved into that directory. If <i>targetName</i> is a file or doesn't exist, then file or directory <i>name</i> replaces <i>targetName</i>. In either case, the old objects are deleted. Moved objects retain their current creation and modification dates.</p> <p>If a directory is moved, then its contents, including all subdirectories, are also moved. No directory moved can be a parent of <i>targetName</i>.</p> <p>A dialog box requests a confirmation if the move would overwrite an existing file or folder. The <b>-y</b>, <b>-n</b>, or <b>-c</b> option can be used to avoid this interaction.</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Diagnostics</b> | Errors are written to diagnostic output. Progress and summary information is also written to diagnostic output if the <b>-p</b> option is specified.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Status</b>      | Move may return the following status values:<br>0 All objects were moved<br>1 Syntax error<br>2 An error occurred during the move<br>4 Cancel was selected or implied with the <b>-c</b> option                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Options</b>     | <p><b>-y</b> Answer "yes" to any confirmation dialog that may occur, causing conflicting files or folders to be overwritten.</p> <p><b>-n</b> Answer "no" to any confirmation dialog that may occur, skipping the move for files or folders that already exist.</p> <p><b>-c</b> Answer "cancel" to any confirmation dialog that may appear, causing the move to stop when a name conflict is encountered.</p> <p><b>-p</b> List progress information as the move takes place.</p>                                                                                                                                                                                                                                                                                                                                          |

**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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | MoveWindow <b>h v</b> [ <i>window</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Description</b> | Moves the upper-left corner of the specified <i>window</i> to the location ( <b>h,v</b> ) where <b>h</b> and <b>v</b> are horizontal and vertical integers. (Use a blank line to separate the numbers <b>h</b> and <b>v</b> 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. If no window is specified, the target window (the second window from the front) is assumed. |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Status</b>      | MoveWindow may return the following status values:<br>0 No errors<br>1 Syntax error (error in parameters)<br>2 The specified window does not exist<br>3 The <b>h v</b> location specified is invalid                                                                                                                                                                                                                                                                                                                                                              |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Examples</b>    | <pre>MoveWindow 72 72</pre> <p>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.)</p> <pre>MoveWindow 0 0 "(Worksheet)"</pre> <p>Moves the Worksheet window to the upper-left corner of the screen (below the menu bar).</p>                                                                                                                                                |
| <b>See also</b>    | SizeWindow, ZoomWindow, StackWindows, and TileWindows commands.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |



---

---

## New—open a new window

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | New [ <i>name...</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Description</b> | <p>Opens a new window as the active (frontmost) window. If <i>name</i> is not specified, the Shell generates a unique name for the new window, of the form "Untitled-<i>n</i>", where <i>n</i> is a decimal number. If <i>name</i> already exists, an error results.</p> <p>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.</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Status</b>      | <p>New may return the following status values:</p> <ul style="list-style-type: none"><li>0 No errors</li><li>1 Syntax error (error in parameters)</li><li>2 Unable to complete operation; a file with the specified name already exists</li><li>3 System error</li></ul>                                                                                                                                                                                                                                                                 |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Examples</b>    | <p>New</p> <p>Opens a new window with a Shell-generated name.</p> <p>New Test.a Test.p Test.c</p> <p>Creates three windows called Test.a, Test.p, and Test.c.</p>                                                                                                                                                                                                                                                                                                                                                                        |
| <b>See also</b>    | Open command.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

---

---

## Newer—compare modification dates between files

|                    |                                                                                                                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Newer [-e] [-c] [-q] <i>name...</i> <i>target</i>                                                                                                                                                                                                                                                                    |
| <b>Description</b> | Compares the modification dates of <i>name</i> and <i>target</i> . Files that have a more recent modification date than <i>target</i> have their names written to standard output. If the target is a non-existent file or directory then all names that exist are considered newer than the target.                 |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                            |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                |
| <b>Output</b>      | Newer files are written to standard output. The names are written out one per line as they appear on the command line.                                                                                                                                                                                               |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                             |
| <b>Status</b>      | The following status values may be returned:<br>0 No error<br>1 Syntax error<br>2 File not found                                                                                                                                                                                                                     |
| <b>Options</b>     | <b>-c</b> Compare creation dates instead of modification dates or for creation dates when used with the <b>-c</b> option.<br><b>-e</b> Look for files with equal modification dates (or creation dates when used with the <b>c</b> option).<br><b>-q</b> 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

|                    |                                                                                                                                                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | NewFolder <i>name</i> ...                                                                                                                                                                                                     |
| <b>Description</b> | Creates new directories with the names specified. Any parent directories included in the <i>name</i> specification must already exist.<br><i>Note:</i> This command can be used only on hierarchical file system (HFS) disks. |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                     |
| <b>Input</b>       | None.                                                                                                                                                                                                                         |
| <b>Output</b>      | None.                                                                                                                                                                                                                         |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                      |
| <b>Status</b>      | The following status values may be returned:<br>0 Folders were created for each name listed<br>1 Syntax error<br>2 An error occurred<br>3 Attempt to use NewFolder on non-HFS volume                                          |
| <b>Options</b>     | None.                                                                                                                                                                                                                         |
| <b>Examples</b>    | NewFolder Memos<br>Creates Memos as a subdirectory of the current directory.<br><br>NewFolder Parent :Parent:Kid<br>Creates Parent as a subdirectory of the current directory, and Kid as a subdirectory of Parent.           |

---

---

## Open—open a window

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Open [-n   -r] [-t] [ <i>name</i> ...]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Description</b> | Opens a file as the active (frontmost) window. If neither <i>name</i> nor the <b>-n</b> option is specified then StdFile's GetFile routine is called, allowing the user to use a dialog box to choose a file. If <i>name</i> is already open as a window, that window becomes the active (frontmost) window.                                                                                                                                                                                                              |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Status</b>      | Open may return the following status values:<br>0 No errors<br>1 Error in parameters<br>2 Unable to complete operation; specified file not found<br>3 System error                                                                                                                                                                                                                                                                                                                                                        |
| <b>Options</b>     | <b>-n</b> Open a new window with the title <i>name</i> . If <i>name</i> is not specified, a unique name is generated for the new window. If file <i>name</i> already exists, that file is opened.<br><b>-r</b> Open a read-only window associated with the file <i>name</i> . If file <i>name</i> doesn't exist, an error occurs.<br><b>-t</b> 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. |
| <b>Examples</b>    | Open<br>Displays StdFile from which to choose a file to open.<br><br>Open -r -t Test.a<br>Opens the file Test.a as the target window, read-only.<br><br>Open *.a<br>Opens all of the files that end with ".a".                                                                                                                                                                                                                                                                                                            |
| <b>See also</b>    | Target, New, and Close commands.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

---

---

## Parameters—write parameters

|                    |                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Parameters [ <i>parameters ...</i> ]                                                                                                                                                                                                                                                                                                                         |
| <b>Description</b> | 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. |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                    |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                        |
| <b>Output</b>      | Parameters are written to standard output.                                                                                                                                                                                                                                                                                                                   |
| <b>Diagnostics</b> | None.                                                                                                                                                                                                                                                                                                                                                        |
| <b>Status</b>      | A status value of 0 is always returned.                                                                                                                                                                                                                                                                                                                      |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                        |
| <b>Example</b>     | <pre>Parameters One Two "and Three"</pre> <p>Writes the following three lines to standard output:</p> <pre>{0} Parameters {1} One {2} Two {3} and Three</pre> <p>Recall that "... " and '...' quotes are removed before parameters are passed to commands.</p>                                                                                               |
| <b>See also</b>    | Echo and Quote commands.<br>"Parameters to Scripts" in Chapter 5.                                                                                                                                                                                                                                                                                            |

---

---

## Pascal—Pascal Compiler

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Pascal [ <i>option...</i> ] [ <i>file...</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Description</b> | Compiles the specified Pascal source files (programs or units). You can specify zero or more filenames. Each file is compiled separately—compiling file <i>Name.p</i> creates object file <i>Name.p.o</i> . By convention, Pascal source filenames end in a “.p” suffix. See the <i>MPW Pascal 2.0 Reference</i> for details of the language definition.                                                                                                                                               |
| <b>Type</b>        | Tool.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Input</b>       | If no filenames are specified, standard input is compiled, with output directed to the file <i>p.o</i> . You can terminate input by pressing Command-Enter.                                                                                                                                                                                                                                                                                                                                            |
| <b>Output</b>      | Nothing is written to standard output. For each input file <i>name</i> , object code is sent to the file <i>name.o</i> .                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Diagnostics</b> | Errors are written to diagnostic output. Progress and summary information is also written to diagnostic output if the <b>-p</b> option is selected.                                                                                                                                                                                                                                                                                                                                                    |
| <b>Status</b>      | The following status values may be returned:<br>0 Successful completion<br>1 Error in parameters<br>2 Compilation halted                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Options</b>     | <b>-b</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.)<br><b>-c</b> Syntax check only—no object file is generated.<br><b>-d name=TRUE   FALSE</b> Set the compile time variable <i>name</i> to TRUE or FALSE.<br><b>-e errLogFile</b> Write all errors to the error log file <i>errLogFile</i> . A copy of the error report will still be sent to diagnostic output. |

**-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 as follows:

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, then no other searching is applied.

If the file wasn't found, and the *pathname* used to specify the file was a *partial pathname* (no colons in the name or a leading colon), then 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.

**-k** *prefixpath*

Put the files specified in \$LOAD commands in the directory specified by *prefixpath*.

**-mc68020**

Generate code to take advantage of the MC68020 processor.

**-mc68881**

Generate code to take advantage of the MC68881 coprocessor.

**-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:** This 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.

**-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.

**-w**

Turn off peephole optimizer.



- y *pathname*** Put the Compiler's temporary intermediate (".o.i") files in the directory specified by *pathname*.
- z** Turn off the output of embedded procedure names in the object code. This option is equivalent to specifying {\$D-} in the source code. .

**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, and performing no range checking.

**Note**

Listing files are not produced directly by the Compiler. Refer to the PasMat and PasRef tools.

**Availability**

The Pascal Compiler is available as part of a separate Apple product, MPW Pascal.

**See also**

PasMat and PasRef commands.  
*MPW Pascal 2.0 Reference.*

---

---

## PasMat—Pascal program formatter

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>PasMat [ option... ] [ inputfile [ outputfile ] ]</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Description</b> | <p>Reformats Pascal source code into a standard format, suitable for printouts or compilation. PasMat accepts full programs, external procedures, blocks, and groups of statements.</p> <p><i>Note:</i> 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.</p> <p>PasMat options let you do the following:</p> <ul style="list-style-type: none"><li><input type="checkbox"/> Convert a program to uniform case conventions.</li><li><input type="checkbox"/> Indent a program to show its logical structure, and adjust lines to fit into a specified line length.</li><li><input type="checkbox"/> Change the comment delimiters (* *) to { }.</li><li><input type="checkbox"/> Remove the underscore character ( _ ) from identifiers, rename identifiers, or change their case.</li><li><input type="checkbox"/> Format include files named in MPW Pascal include directives.</li></ul> <p>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 and see.</p> <p>See Appendix K of the manual <i>MPW Pascal 2.0 Reference</i> for details of PasMat directives and their functions. The first dialog box of the Pascal Commando dialog is reproduced here for your convenience.</p> |
| <b>Type</b>        | Tool.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Input</b>       | If no input files are specified, standard input is formatted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Output</b>      | 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, 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 will give 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 values:

- 0 Normal termination
- 1 Parameter or option error
- 2 Execution terminated

## Options

Most of the following options modify the initial default settings of the directives described in Appendix K of the *MPW Pascal 2.0 Reference*.

**-a** Set **a-** to disable CASE label bunching.

**-b** Set **b+** to enable IF bunching.

The screenshot shows the 'Pasmat Options' dialog box. It is divided into several sections:

- I/O Specifications...**
- Identifier Handling...**
- Bunching**
  - IF's
  - FOR/WHILE/WITH's
  - CASE label's
  - ELSE/IF on new line
  - BEGIN on same line
  - THEN on new line
- Spacing**
  - None around ops
  - None around :-=
  - None after commas
- Indenting**
  - Procedure bodies
  - Between BEGIN/END
  - Fields under id
  - Tabbing value:
- Miscellaneous**
  - No formatting
  - (\* \*) changed to { }
  - Align UAR colons
  - Progress
- Grouping**
  - Assignment/Calls
  - Formal parameters
  - \*Smart\* grouping
  - Separate CASE tags
- Command Line**  
Pasmat
- Help**  
Reformats Pascal source into a standard format, suitable for printouts or compilation. Accepts full programs/units, procs, blocks, and statements. Most options should be \*embedded\* in source and NOT specified here!
- Buttons:** Cancel, Pasmat

**-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=** 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 includes 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 includes. This option is implied if the **-i** option is used.
- 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 includes (**-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, then 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`. Includes 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 the following 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 include files and conditional compiler directives are to be formatted.
- The Pascal include directive should be the last thing on the input line if includes are to be processed. Includes are processed to a maximum nesting depth of five. All includes 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.

## Availability

PasMat is available as part of a separate Apple product, MPW Pascal 2.0.

## See also

Pascal and PasRef commands.

Appendix K of the *MPW Pascal 2.0 Reference*.

---

---

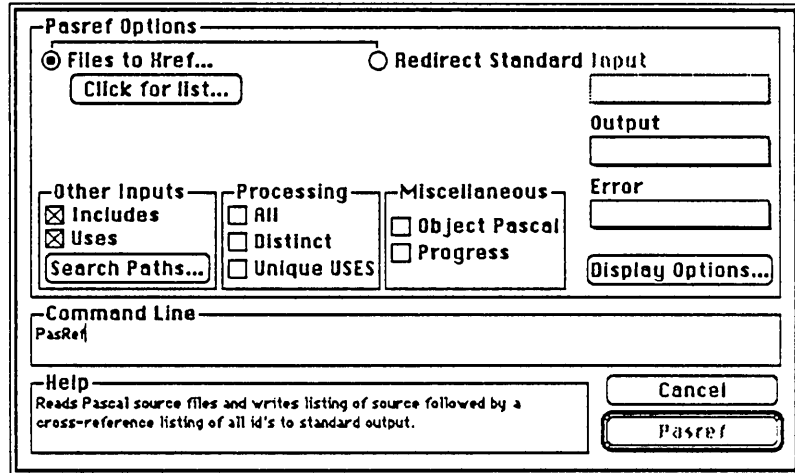
## PasRef—Pascal cross-referencer

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | PasRef [ <i>option...</i> ] [ <i>sourceFile...</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Description</b> | <p>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).</p> <p>See the <i>MPW Pascal 2.0 Reference</i> for details of the Pascal language. The first dialog box of PasRef's Commando dialog is reproduced here for your convenience.</p> <p>Identifiers may be up to 63 characters long, and are displayed in their entirety unless overridden with the <b>-x</b> directive. Identifiers may remain as they appear in the input, or they can be converted to all lowercase (<b>-l</b>) or all uppercase (<b>-u</b>).</p> <p>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.</p> <p>Uses declarations can also be processed by PasRef (their associated \$U <i>filename</i> 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).</p> <p>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). If it has already been processed, then the file is skipped.</p> |
| <b>Type</b>        | Tool.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Input</b>       | If no filenames are specified, standard input is processed. Unless the <b>-d</b> 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 <b>-d</b> option is the same as executing PasRef individually for each file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Output</b>      | 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Diagnostics</b> | Parameter errors and progress information are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Status</b>      | PasRef may return the following status values:<br>0 Normal termination<br>1 Parameter or option error<br>2 Execution terminated                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |



## Options

- a Process all files even if they are duplicates of ones 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. See the "Example" section for further details.
- nt | -nototal Do not display the total line count in the source listing. This option is ignored if no listing is being generated (-nl).

**-n[*u*] | -nouses**

Do not process USES declarations. (The default is to process USES declarations.) If **-nu** is specified, then the **-c** option is ignored.

**-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 is to assume 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 write 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 DRVRCtrl (ctlPB: ParmBlkPtr; dCtl: DCtlPtr): OSErr;
21 1 21 -- FUNCTION DRVRCtrl (ctlPB: ParmBlkPtr; dCtl: DCtlPtr): OSErr;
22 1 22 -- FUNCTION DRVRCtrl (ctlPB: ParmBlkPtr; dCtl: DCtlPtr): OSErr;
23 1 23 -- FUNCTION DRVRCtrl (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 -0 A 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        | 90  | ( 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) |
|         | 63* | ( 1) | 74*  | ( 1) | 143  | ( 1) |
|         |     |      | 146  | ( 1) | 168* | ( 1) |
|         |     |      | 173* | ( 1) |      |      |

-D-

|            |     |      |      |      |      |      |
|------------|-----|------|------|------|------|------|
| dCtl       | 19* | ( 1) | 20*  | ( 1) | 21*  | ( 1) |
|            | 39  | ( 1) | 43*  | ( 1) | 50   | ( 1) |
|            | 63* | ( 1) | 65   | ( 1) | 67   | ( 1) |
|            | 142 | ( 1) | 168* | ( 1) | 173* | ( 1) |
| DctlPtr    | 19  | ( 1) | 20   | ( 1) | 21   | ( 1) |
|            | 43  | ( 1) | 63   | ( 1) | 74   | ( 1) |
| dCtlRefNum | 39  | ( 1) | 54   | ( 1) | 168  | ( 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) |
| 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. Thus you can 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 means a reference to the identifier.

### 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 just don't want the lexical information, you may 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 is given 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.)

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 Pascal 2.0.

### See also

Pascal command.

*MPW Pascal 2.0 Reference.*

---

---

## Paste—replace selection with Clipboard contents

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Paste [-c <i>count</i> ] <i>selection</i> [ <i>window</i> ]                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Description</b> | <p>Finds <i>selection</i> 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.</p> <p>For a definition of <i>selection</i>, see "Selections" in Chapter 6; a summary of the selection syntax is contained in Appendix B.</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Status</b>      | <p>The following status values may be returned:</p> <ul style="list-style-type: none"><li>0 At least one instance of the selection was found</li><li>1 Syntax error</li><li>2 Any other error</li></ul>                                                                                                                                                                                                                      |
| <b>Option</b>      | <p><b>-c <i>count</i></b> For a count of <i>n</i>, replace the next <i>n</i> instances of the selection with the contents of the Clipboard.</p>                                                                                                                                                                                                                                                                              |
| <b>Examples</b>    | <p>Paste \$</p> <p>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.</p> <p>Paste /BEGIN/:/END/</p> <p>Selects everything from the next BEGIN to the following END, and replaces the selection with the contents of the Clipboard.</p>                                                            |
| <b>See also</b>    | <p>Copy, Cut, and Replace commands.</p> <p>"Edit Menu" in Chapter 3.</p> <p>"Selections" in Chapter 6.</p>                                                                                                                                                                                                                                                                                                                   |

---

---

## PerformReport—generate a performance report

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | PerformReport [ <i>option...</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Description</b> | 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 12.                                                                                                                                                                                                                            |
| <b>Type</b>        | Tool.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Input</b>       | Standard input is not processed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Output</b>      | The report file is written to standard output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Diagnostics</b> | 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 <b>-p</b> option is specified.                                                                                                                                                                                                                                                                                              |
| <b>Status</b>      | The following status values may be returned:<br>0 No errors<br>1 Warning issued<br>2 Error encountered<br>3 Heap error, usually insufficient memory                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Options</b>     | <b>-a</b> 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.)<br><b>-l <i>fileName</i></b> Read the link map of the file named <i>fileName</i> .<br><b>-m <i>fileName</i></b> Read the performance data file named <i>fileName</i> . The default name is Perform.out.<br><b>-n <i>NN</i></b> Show the top <i>NN</i> procedures. The default is 50.<br><b>-p</b> Write progress and summary information to the diagnostic output file. |
| <b>Example</b>     | <pre>Catenate "(MPW)ROM.Maps:MacIIROM.map &gt;&gt; <i>myFileName</i> PerformReport -m <i>myFileName</i> &gt; <i>myReport</i></pre> Adds the ROM map file to the end of the link map file, <i>myFileName</i> . Reads the files <i>myFileName</i> and Perform.out and writes the output to <i>myReport</i> .                                                                                                                                                                                                                          |
| <b>See also</b>    | Chapter 12, "Performance Measurement Tools."<br><i>MPW Pascal 2.0 Reference.</i><br><i>MPW C 2.0 Reference.</i>                                                                                                                                                                                                                                                                                                                                                                                                                     |

---

---

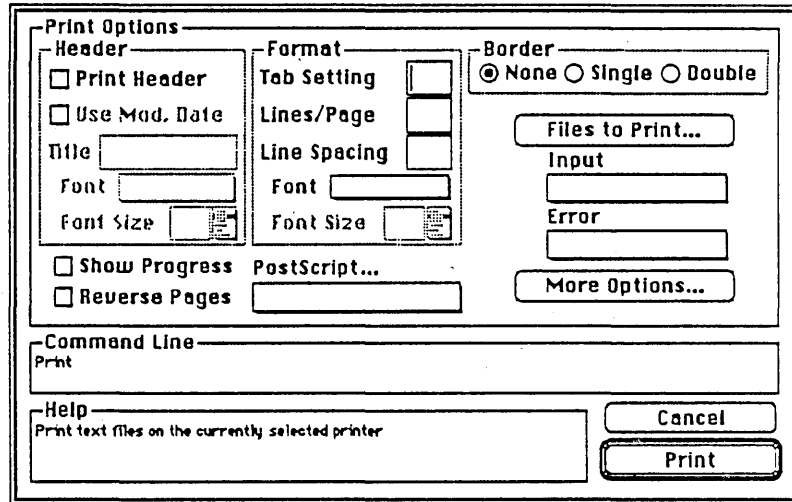
## Print—print text files

- 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 support automatic font substitution. 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**                If no files are specified and if input has been redirected to standard input, Print reads from standard input. You can terminate input by pressing Command-Enter.
- 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 values 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.
  - copies** | *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[size] 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 to 10.
- p** Write progress information to diagnostic output, indicating which files are printing and the number of lines and pages printed.
- 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. *quality* is one 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.
- 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 3-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 may 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 values:

- 0    Normal termination
- 1    Parameter or option error
- 2    Execution terminated

## 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, then 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 may 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, then 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 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 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, so the **-u** option will be ignored when the **-n** option is specified.

## 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 DRVRClose
74 74 1 DRVRControl
76 76 2 DrawWindow
83 83 3 PrintNum
93 93 3 GetVolStuff
108 108 3 PrtRsrcStr
168 168 1 DRVRRPrime
173 173 1 DRVRRStatus
```

```
*** 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 \$\$ 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.

---

---

## Quit—quit MPW

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Quit [-y   -n   -c]                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Description</b> | This command is equivalent to the menu item Quit. Quit executes the standard quit procedures, asking confirmation to save modified files, close all windows, and so on.                                                                                                                                                                                                                                                      |
| <b>Type</b>        | Miscellaneous.                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Status</b>      | <p>The following status values may be returned:</p> <ol style="list-style-type: none"><li>1 Syntax error</li><li>2 Command aborted</li></ol> <p><i>Note:</i> Quit cannot return a status of 0, because if there are no errors the command never returns.</p>                                                                                                                                                                 |
| <b>Options</b>     | <p><b>-y</b> Answer “yes” to any confirmation dialog that occurs, causing all modified windows to be saved before closing them.</p> <p><b>-n</b> Answer “no” to any confirmation dialog that occurs, causing all modified windows to be closed without saving any changes.</p> <p><b>-c</b> Answer “cancel” to any confirmation dialog that occurs. This effectively aborts the command if any windows need to be saved.</p> |
| <b>Examples</b>    | <pre>Quit -y</pre> <p>Quits MPW answering “yes” to any dialogs such as those prompting to save files.</p> <pre>Quit -c</pre> <p>Quits MPW unless any confirmation dialogs occur and dialog boxes are displayed.</p>                                                                                                                                                                                                          |
| <b>See also</b>    | Shutdown command.                                                                                                                                                                                                                                                                                                                                                                                                            |

---

---

## Quote—quote parameters

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Quote [ -n ] [ parameters... ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Description</b> | <p>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 quotes. If no parameters are specified, only a return is written.</p> <p>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 command script.</p> <p>The following special characters are quoted:</p> <p>Space Tab Return Null</p> <p># ; &amp;   ( ) ` ' " / \ { } ~ ? = [ ] + * . . ® &lt; &gt; ≥ ...</p> |
| <b>Type</b>        | Miscellaneous.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Output</b>      | Parameters are written to standard output and are enclosed in single quotes if they contain special characters.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Diagnostics</b> | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Status</b>      | Status value 0 is always returned.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Option</b>      | <p><b>-n</b> Don't write a return following the last parameter. The insertion point remains at the end of the output. The <b>-n</b> isn't written to standard output.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

## 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 quotes in the last Print command.

## See also

Echo and Parameters commands.



---

---

## Rename—rename files and directories

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Rename [-y   -n   -c] <i>name newName</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Description</b> | <p>The file, folder, or disk specified by <i>name</i> is renamed <i>newName</i>. A dialog box requests a confirmation if the rename would overwrite an existing file or folder. The <b>-y</b>, <b>-n</b>, or <b>-c</b> options can be used to avoid this interaction.</p> <p><i>Note:</i> You can't use the Rename command to change the directory a file is in. To do this, use the Move command.</p> <p><i>Note also:</i> Wildcard renames in the following form <i>will not work</i>:</p> <pre>Rename *.text *.p</pre> <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.</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Status</b>      | <p>The following status values may be returned:</p> <ul style="list-style-type: none"><li>0 Successful rename</li><li>1 Syntax error</li><li>2 <i>Name</i> does not exist</li><li>3 An error occurred</li><li>4 Cancel was selected or implied by the -c option</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Options</b>     | <ul style="list-style-type: none"><li><b>-y</b> Answer “yes” to any confirmation dialog that may occur, causing conflicting file or folder to be deleted.</li><li><b>-n</b> Answer “no” to any confirmation dialog that may occur, stopping the rename if <i>newName</i> already exists.</li><li><b>-c</b> Answer “cancel” to any confirmation dialogs, aborting the rename if <i>newName</i> already exists.</li></ul>                                                                                                                                                                                                                                                                                                                                                    |

## 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, but if a conflict occurs, 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-hand 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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Replace [-c <i>count</i> ] <i>selection replacement</i> [ <i>window</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Description</b> | <p>Finds <i>selection</i> in the specified window and replaces it with <i>replacement</i>. 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 will be repeated <i>count</i> times.</p> <p>For a definition of <i>selection</i>, see “Selections” in Chapter 6. A summary of the selection syntax is contained in Appendix B.</p> <p>You can include references to parts of the selection in the <i>replacement</i> by using the ® operator. The expression ®<i>n</i>, where <i>n</i> is a digit, is replaced with the string of characters that matches the regular expression tagged by ®<i>n</i> in the selection. (See “Tagging Regular Expressions With the ® Operator” in Chapter 6.)</p> <p>All searches are by default case insensitive. To specify case-sensitive matching, set the {CaseSensitive} variable before executing the command. (You can do this by checking the Case Sensitive box in the dialogs displayed by the Find and Replace menu items.)</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Status</b>      | <p>The following status values may be returned:</p> <ul style="list-style-type: none"><li>0 At least one instance of the selection was found</li><li>1 Syntax error</li><li>2 Any other error</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Options</b>     | <p>-c <i>count</i> Repeat the command <i>count</i> 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).</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

## Examples

```
Replace -c ∞ /myVar/ 'myVariable' Prog.p
```

Replaces every subsequent instance of the selection with the string in single quotes.

```
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.

---

---

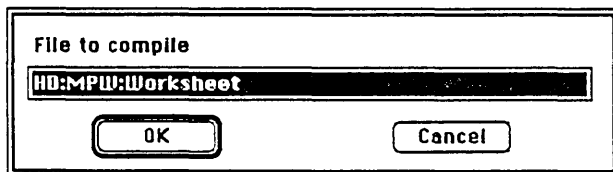
## Request—request text from a dialog box

|                    |                                                                                                                                                                                                                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Request [ -d <i>default</i> ] [ <i>message...</i> ]                                                                                                                                                                                                                              |
| <b>Description</b> | Displays an editable text dialog box with OK and Cancel buttons and the prompt <i>message</i> . If the OK button is selected, then all text that the user typed into the dialog box is written to standard output. The -d option lets you set a default response to the request. |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                        |
| <b>Input</b>       | Reads standard input for the message if no parameters are specified.                                                                                                                                                                                                             |
| <b>Output</b>      | Text from the dialog box is written to standard output.                                                                                                                                                                                                                          |
| <b>Diagnostics</b> | None.                                                                                                                                                                                                                                                                            |
| <b>Status</b>      | Request may return the following status values:<br>0 The OK button was selected<br>1 Syntax errors<br>2 The Cancel button was selected                                                                                                                                           |
| <b>Options</b>     | -d <i>default</i> The editable text field of the dialog box is initialized to <i>default</i> . 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.                                 |

### 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.

---

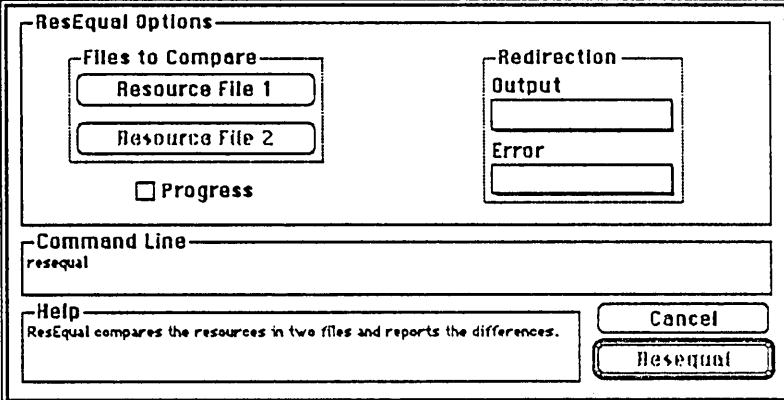
---

## ResEqual—compare resources in files

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |   |                 |   |                           |   |                   |   |                      |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|-----------------|---|---------------------------|---|-------------------|---|----------------------|
| <b>Syntax</b>      | ResEqual [ <i>option</i> ] <i>file1 file2</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |   |                 |   |                           |   |                   |   |                      |
| <b>Description</b> | <p>ResEqual compares the resources in two files and writes their differences to standard output.</p> <p>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.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |   |                 |   |                           |   |                   |   |                      |
| <b>Type</b>        | Tool.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |   |                 |   |                           |   |                   |   |                      |
| <b>Input</b>       | The <i>file1</i> and <i>file2</i> parameters specify the two files whose resources are to be compared.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |   |                 |   |                           |   |                   |   |                      |
| <b>Output</b>      | <p>Descriptions of the differences in the resources of the two files are written to standard output.</p> <p>The following messages appear when reporting differences:</p> <ul style="list-style-type: none"><li><input type="checkbox"/> In 1 but not in 2<br/>—<i>the resource type and ID are displayed</i>—</li><li><input type="checkbox"/> In 2 but not in 1<br/>—<i>the resource type and ID are displayed</i>—</li><li><input type="checkbox"/> Resources are different sizes<br/>—<i>the resource type and ID are displayed</i>—<br/>—<i>the size of the resource in each file is displayed</i>—</li><li><input type="checkbox"/> Resources have different contents<br/>—<i>the resource type and ID are displayed</i>—<br/>Contents of resource in file 1 at offset<br/>—<i>offset to the differing bytes from the start of the resource is displayed</i>—<br/>—<i>16 bytes at the offset are displayed</i>—<br/>Contents of resource in file 2 at offset<br/>—<i>offset to the differing bytes from the start of the resource is displayed</i>—<br/>—<i>16 bytes at the offset are displayed</i>—</li></ul> |   |                 |   |                           |   |                   |   |                      |
| <b>Diagnostics</b> | Parameter errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |   |                 |   |                           |   |                   |   |                      |
| <b>Status</b>      | <p>The following status values may be returned</p> <table><tr><td>0</td><td>Resources match</td></tr><tr><td>1</td><td>Parameter or option error</td></tr><tr><td>2</td><td>Files don't match</td></tr><tr><td>3</td><td>Execution terminated</td></tr></table>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | 0 | Resources match | 1 | Parameter or option error | 2 | Files don't match | 3 | Execution terminated |
| 0                  | Resources match                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |   |                 |   |                           |   |                   |   |                      |
| 1                  | Parameter or option error                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |                 |   |                           |   |                   |   |                      |
| 2                  | Files don't match                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |   |                 |   |                           |   |                   |   |                      |
| 3                  | Execution terminated                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |   |                 |   |                           |   |                   |   |                      |

**Option****-p**

Write progress information to diagnostic output.



The image shows a dialog box titled "ResEqual Options". It is divided into several sections:

- Files to Compare:** Contains two text input fields labeled "Resource File 1" and "Resource File 2".
- Redirection:** Contains two text input fields labeled "Output" and "Error".
- Progress:** A checkbox labeled "Progress" is currently unchecked.
- Command Line:** A text input field containing the command "resequat".
- Help:** A text area containing the text "ResEqual compares the resources in two files and reports the differences."
- Buttons:** "Cancel" and "Resequal" buttons are located at the bottom right.

**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 10 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

|                    |                                                                                                                                                                                                                                                                                                                |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Revert [ -y ] [ <i>window...</i> ]                                                                                                                                                                                                                                                                             |
| <b>Description</b> | Reverts the specified windows to their previously saved states. If no window is specified, Revert works on the target window. Revert causes a confirmation dialog box to appear, but you can avoid this dialog box by using the -y option to revert unconditionally to the last saved version of the document. |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                      |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                          |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                          |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                       |
| <b>Status</b>      | The following status values may be returned:<br>0 No errors<br>1 Parameter or option error<br>2 The specified window does not exist<br>3 A system error occurred                                                                                                                                               |
| <b>Option</b>      | -y Unconditionally revert all named windows to their previously saved states.                                                                                                                                                                                                                                  |
| <b>Examples</b>    | Revert<br>Displays a confirmation dialog box for reverting the target window to its last saved state.<br><br>Revert -y "{Worksheet}"<br>Reverts unconditionally to last saved worksheet.                                                                                                                       |
| <b>See also</b>    | Close and Save commands.                                                                                                                                                                                                                                                                                       |



---

---

## Rez—Resource Compiler

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |   |           |   |                     |   |                      |   |                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|-----------|---|---------------------|---|----------------------|---|----------------------|
| <b>Syntax</b>      | Rez [ <i>option...</i> ] [ <i>resourceDescriptionFile...</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |   |           |   |                     |   |                      |   |                      |
| <b>Description</b> | <p>Creates 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 the Resource Decompiler, DeRez. 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 <code>#include</code> and <code>read</code> directives in the resource description file), and from other resource files (via the <code>include</code> directive).</p> <p>Rez includes macro processing, full expression evaluation, and built-in functions and system variables. For details of Rez, and the format of a resource description file, see Chapter 8. For a summary of the format of a resource description file, see Appendix D.</p> |   |           |   |                     |   |                      |   |                      |
| <b>Type</b>        | Tool.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |   |           |   |                     |   |                      |   |                      |
| <b>Input</b>       | <p>Standard input is processed if no filenames are specified.</p> <p>For all input files on the command line, the following search rules are applied:</p> <ol style="list-style-type: none"><li>1. Try to open the file with the name specified “as is.”</li><li>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 <code>{RIncludes}</code> variable and try to open the file.</li></ol>                                                                                                                                                                                                                                                                                                     |   |           |   |                     |   |                      |   |                      |
| <b>Output</b>      | <p>No output is sent to the standard output file. By default, the resource fork is written to the file <code>Rez.Out</code>. You can specify an output file with the <code>-o</code> option.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |   |           |   |                     |   |                      |   |                      |
| <b>Diagnostics</b> | <p>If no errors or warnings are detected, Rez runs silently. Errors and warnings are written to diagnostic output.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |   |           |   |                     |   |                      |   |                      |
| <b>Status</b>      | <p>Rez may return the following status values:</p> <table><tr><td>0</td><td>No errors</td></tr><tr><td>1</td><td>Error in parameters</td></tr><tr><td>2</td><td>Syntax error in file</td></tr><tr><td>3</td><td>I/O or program error</td></tr></table>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | 0 | No errors | 1 | Error in parameters | 2 | Syntax error in file | 3 | I/O or program error |
| 0                  | No errors                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |   |           |   |                     |   |                      |   |                      |
| 1                  | Error in parameters                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |   |           |   |                     |   |                      |   |                      |
| 2                  | Syntax error in file                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |   |           |   |                     |   |                      |   |                      |
| 3                  | I/O or program error                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |   |           |   |                     |   |                      |   |                      |

## Options

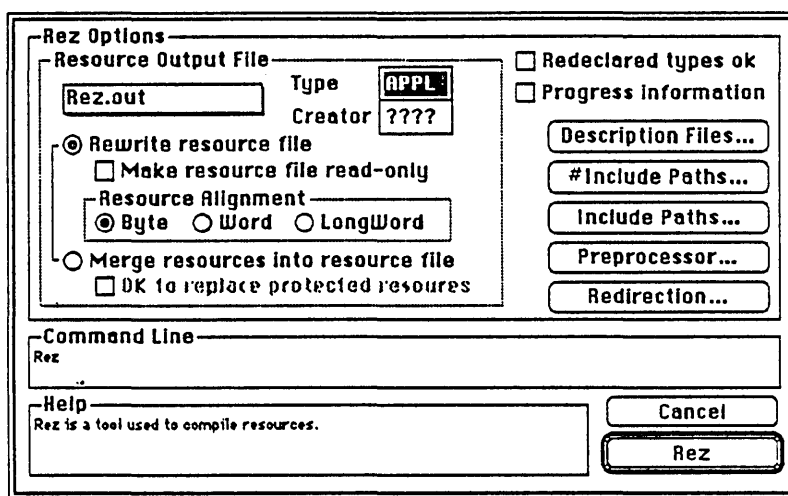
**-align** *word* [*longword*]

Aligns 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]**

Appends Rez's output to the output file rather than replacing the output file.

**Warning:** Rez will overwrite 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 file that has a protected bit set, unless the **-ov** option is specified. Although it is possible to append a resource directly into a running system file, it is not recommended. See also the **-ov** option below.



**-c[reator]** *creatorExpr*

Set the output file creator. (The default value is '????'.)

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

Define the macro variable *macro* to have the value *data*. If *data* is omitted, then *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 may be specified more than once. The paths will be searched in the order they appear on the command line.

```
rez -i (mpw)myStuff: -i hd:tools...
```

**-o** *outputFile*

Place the output in *outputFile*. The default output file is Rez.Out.

**-ov**

Overrides 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'.
- 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.

Chapters 5 and 8.

Standard resource type declarations in the {RIncludes} directory:

- Types.r
- SysTypes.r
- MPWTypes.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 hex or decimal for numeric values, or as a hex dump with associated printable Macintosh characters. The characters return (\$0D), tab (\$09), and null (\$00) are displayed as "↵", "Δ", and "." respectively. The same is true for a resource name shown as a text string.

*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** option).

**Diagnostics** Error messages go to diagnostic output.

**Status** The following status values 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 on.
- r[awdump]** Same as **-dump** but also dumps contents of data blocks, and so on.  
*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.

---

---

## Save—save windows

|                    |                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Save [-a   <i>windows...</i> ]                                                                                                                                                                                               |
| <b>Description</b> | Saves the contents of <i>window</i> or a list of <i>windows</i> 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). |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                    |
| <b>Input</b>       | None.                                                                                                                                                                                                                        |
| <b>Output</b>      | None.                                                                                                                                                                                                                        |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                     |
| <b>Status</b>      | Save may return the following status values:<br>0 No errors<br>1 Syntax error<br>2 Specified window does not exist                                                                                                           |
| <b>Option</b>      | -a Save all open windows. This option cannot be used when any windows are specified.                                                                                                                                         |
| <b>Examples</b>    | Save -a<br>Saves all open windows.<br><br>Save "{Active}" "{Worksheet}"<br>Saves the Worksheet window and the contents of the active window.                                                                                 |
| <b>See also</b>    | Close and Revert commands.                                                                                                                                                                                                   |

---

---

## Search—search files for a pattern

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |    |                                                                     |    |                                                                                            |    |                                                             |    |                                                               |                |                                                                                  |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|---------------------------------------------------------------------|----|--------------------------------------------------------------------------------------------|----|-------------------------------------------------------------|----|---------------------------------------------------------------|----------------|----------------------------------------------------------------------------------|
| <b>Syntax</b>      | Search [-s   -i ] [-r] [-q] [-f <i>file</i> ] / <i>pattern</i> / [ <i>file</i> ...]                                                                                                                                                                                                                                                                                                                                                                                                                                                               |    |                                                                     |    |                                                                                            |    |                                                             |    |                                                               |                |                                                                                  |
| <b>Description</b> | <p>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.</p> <p><i>Pattern</i> (defined in “Pattern Matching” in Chapter 6 and in Appendix B) is a regular expression, optionally enclosed in forward slashes (/).</p>                                                                                                           |    |                                                                     |    |                                                                                            |    |                                                             |    |                                                               |                |                                                                                  |
| <b>Type</b>        | Tool.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |    |                                                                     |    |                                                                                            |    |                                                             |    |                                                               |                |                                                                                  |
| <b>Input</b>       | Standard input is read if no files are specified.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |    |                                                                     |    |                                                                                            |    |                                                             |    |                                                               |                |                                                                                  |
| <b>Output</b>      | Each matching line is written to standard output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |    |                                                                     |    |                                                                                            |    |                                                             |    |                                                               |                |                                                                                  |
| <b>Diagnostics</b> | Error messages are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |    |                                                                     |    |                                                                                            |    |                                                             |    |                                                               |                |                                                                                  |
| <b>Status</b>      | <p>The following status values may be returned:</p> <table><tr><td>0</td><td>No error</td></tr><tr><td>1</td><td>Syntax error</td></tr><tr><td>2</td><td>Pattern not found</td></tr></table>                                                                                                                                                                                                                                                                                                                                                      | 0  | No error                                                            | 1  | Syntax error                                                                               | 2  | Pattern not found                                           |    |                                                               |                |                                                                                  |
| 0                  | No error                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |    |                                                                     |    |                                                                                            |    |                                                             |    |                                                               |                |                                                                                  |
| 1                  | Syntax error                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |    |                                                                     |    |                                                                                            |    |                                                             |    |                                                               |                |                                                                                  |
| 2                  | Pattern not found                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |    |                                                                     |    |                                                                                            |    |                                                             |    |                                                               |                |                                                                                  |
| <b>Options</b>     | <table><tr><td>-r</td><td>Write the lines <b>not</b> matching the pattern to standard output.</td></tr><tr><td>-q</td><td>Write only the matching lines to standard output. Do not prepend filename and line number.</td></tr><tr><td>-s</td><td>Case-sensitive search, overriding {CaseSensitive} variable.</td></tr><tr><td>-i</td><td>Case-insensitive search, overriding {CaseSensitive} variable.</td></tr><tr><td>-f <i>file</i></td><td>All lines that do not get written to standard output are written into this file.</td></tr></table> | -r | Write the lines <b>not</b> 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 <i>file</i> | All lines that do not get written to standard output are written into this file. |
| -r                 | Write the lines <b>not</b> 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 <i>file</i>     | 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 first 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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Set [ <i>name</i> [ <i>value</i> ] ]                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Description</b> | <p>Assigns the string <i>value</i> to the variable <i>name</i>. If <i>value</i> is omitted, Set writes the name and its current value to standard output. If both <i>name</i> and <i>value</i> are omitted, Set writes a list of all variables and their values to standard output. (This output is in the form of Set commands.)</p> <p><i>Note:</i> To make variable definitions available to enclosed scripts and programs, you must use the Export command.</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Output</b>      | If <i>value</i> or both <i>name</i> and <i>value</i> are omitted, variable names and their values are written to standard output.                                                                                                                                                                                                                                                                                                                                   |
| <b>Diagnostics</b> | Error messages are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Status</b>      | <p>The following status values may be returned:</p> <ul style="list-style-type: none"><li>0 No error</li><li>1 Syntax error</li><li>2 Variable “name” does not exist</li></ul>                                                                                                                                                                                                                                                                                      |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

## Examples

```
Set CIncludes "{(MPW)CFiles:CIncludes:"
```

Redefines the variable CIncludes.

```
Set CIncludes
```

Displays the new definition of CIncludes.

```
Set Commands d
```

```
 "{(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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |   |                       |   |                                            |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|-----------------------|---|--------------------------------------------|
| <b>Syntax</b>      | SetDirectory <i>directory</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                       |   |                       |   |                                            |
| <b>Description</b> | <p>Set the default directory and add the new default directory to the Directory menu if it is not already present. The <i>directory</i> parameter must be specified.</p> <p><i>Note:</i> Directory names should not contain any of the special characters shown below. These characters all have special meaning when they appear in menu items:</p> <p>- ; ^ ! &lt; / (</p> <p>The SetDirectory script is used to implement the Set Directory menu item in the Directory menu.</p> |   |                       |   |                                            |
| <b>Type</b>        | Script.                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |                       |   |                                            |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |   |                       |   |                                            |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |   |                       |   |                                            |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                            |   |                       |   |                                            |
| <b>Status</b>      | <p>The following status values may be returned:</p> <table><tr><td>0</td><td>Successful completion</td></tr><tr><td>1</td><td>Parameter error or unable to set directory</td></tr></table>                                                                                                                                                                                                                                                                                          | 0 | Successful completion | 1 | Parameter error or unable to set directory |
| 0                  | Successful completion                                                                                                                                                                                                                                                                                                                                                                                                                                                               |   |                       |   |                                            |
| 1                  | Parameter error or unable to set directory                                                                                                                                                                                                                                                                                                                                                                                                                                          |   |                       |   |                                            |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |   |                       |   |                                            |
| <b>Examples</b>    | <pre>SetDirectory "{MPW}"CEexamples:</pre> <p>Sets the default directory to the CExamples: folder in the {MPW} directory, and adds "{MPW}"CEexamples: to the Directory menu if it's not already there.</p> <pre>SetDirectory...</pre> <p>Uses the Commando dialog box to select the default directory interactively.</p>                                                                                                                                                            |   |                       |   |                                            |
| <b>See also</b>    | Directory, DirectoryMenu, and Files commands.                                                                                                                                                                                                                                                                                                                                                                                                                                       |   |                       |   |                                            |

---

---

## SetFile—set file attributes

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | SetFile [ <i>option...</i> ] <i>file...</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Description</b> | Sets attributes for one or more files. The options apply to all files listed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Type</b>        | Tool.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Diagnostics</b> | Error messages are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Status</b>      | The following status values may be returned:<br>0 The attributes for all files were set<br>1 Syntax error<br>2 An error occurred                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Options</b>     | <p><b>-c</b> <i>creator</i> Set the file creator. <i>Creator</i> must be 4 characters; for example,<br/>-c 'MPS '</p> <p><b>-t</b> <i>type</i> Set the file type. <i>Type</i> must be 4 characters; for example,<br/>-t 'TEXT'</p> <p><b>-d</b> <i>date</i> Set the creation date. <i>Date</i> is a string in the form<br/>"<i>mm/dd/yy</i> [ <i>hh:mm</i> [:<i>ss</i>] [ AM   PM ]]"<br/>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.</p> <p><b>-m</b> <i>date</i> Set the modification date: same format as above. A period (.) indicates the current date and time.</p> <p><b>-i</b> <i>h,v</i> Set the icon location. <i>h</i> and <i>v</i> are positive integer values and represent the horizontal and vertical pixel offsets from the upper-left corner of the enclosing window.</p> |

**-a attributes** Set the file attributes. *Attributes* is a string 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 to 0. For example,

```
Setfile -a vB Filename
```

clears the invisible bit and sets the bundle bit.

*Note:* These attributes are described in the File Manager chapter of *Inside Macintosh*. Note that setting the locked bit doesn't prevent the file from being changed.

## 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 file Foo's modification date.

```
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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | SetPrivilege [-f <i>priv</i> ] [-d <i>priv</i> ] [-c <i>priv</i> ] [-o <i>owner</i> ] [-g <i>group</i> ] [-r] [-i] <i>folder...</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Description</b> | SetPrivilege is equivalent to using the access privileges desk accessory. <i>Priv</i> is a character string (1, 2, or 3 characters long) that specifies privileges for the owner, the group, and everyone ( <b>o</b> , <b>g</b> , and <b>e</b> , respectively). An uppercase letter enables the privilege, lowercase disables the privilege. If a specific character is not in the string then the respective privileges are not changed.                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Output</b>      | When the <b>-i</b> option is used, folder information is written to standard output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Status</b>      | The following status values may be returned:<br><br>0 No error<br>1 Syntax error<br>2 Folder not found, or folder not an AppleShare folder<br>3 Use is not owner; could not modify privileges                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Options</b>     | <b>-o <i>new owner</i></b> Change owner of the folder to <i>new owner</i> .<br><b>-g <i>new group</i></b> Change group of the folder to <i>new group</i> .<br><b>-r</b> Recursively apply changes to enclosed folders.<br><b>-f <i>priv</i></b> See files. Set the privileges with respect to seeing files within folders (equivalent to read access).<br><b>-d <i>priv</i></b> See folders. Set the privileges with respect to seeing folders and directories and listing their contents.<br><b>-c <i>priv</i></b> Make changes. Set privileges allowing users to make changes to files and directories.<br><b>-i</b> Write folder information (owner, group, and access privileges) to standard output. The output is in the form of a SetPrivilege command. The <b>-r</b> option is the only option that may be used in conjunction with the <b>-i</b> option. |

## Examples

```
SetPrivilege -r -f OGe -d OGe -c Oge 0
"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 Oge
```

*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**      Version and revision numbers for an application or MPW tool specified by *file* are assumed to be maintained in the form "*ver.rev*", where *ver* is considered a version number and *rev* a revision number. These values may be displayed by an application's "about box" or when an MPW tool's **-p** option is used. Use SetVersion to independently maintain the version and revision numbers as a resource in the application or tool. Optionally, SetVersion can update a version and revision string in a source file. Pascal, C, and Rez source files are supported.

The current version and revision values are always assumed to be in the specified file's resource fork as a string resource 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 will be 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 may optionally be prefixed with an arbitrary string (**-prefix**), and the revision may be similarly suffixed with an arbitrary string (**-suffix**) for more complex version numbering (such as "Version x1.23B2").

SetVersion can perform the following functions on the version and revision values:

- Increment the version number by 1 (**-v**).
- Set the version number to a specific value (**-sv**).
- Increment the revision number by 1 (**-r**).
- Set the revision number to a specific value (**-sr**).

The 'MPST' resource attached to the application or tool is considered *the* location of the version and revision. If you attach the 'MPST' resource to the actual application or tool, it will "go" wherever the application or tool goes! Thus the application or tool filename is a required parameter to SetVersion. However, the values contained in the 'MPST' resource can be used to set a corresponding string constant in a source file used to generate the application or tool. This feature is optional, but it should be used for two reasons. First, it explicitly allows the source to reflect the version and revision numbers in the 'MPST' resource. Second, if, for any reason, the 'MPST' resource cannot be accessed, the constant can be used.



The following Pascal code fragment illustrates how the 'MPST' resource and its corresponding source string constant can be used to access the version and revision of an MPW tool. First, in the case of Pascal, the source constant is assumed to be declared as follows (all the formats are discussed under "Options" below):

```
CONST
 Version = '1.2'; {ver.rev string const.}
```

The following procedure can now be used to get the current version and revision numbers:

```
PROCEDURE GetVerRev(VAR VerRev: Str255);

 VAR
 H: StringHandle;
 i: Integer;

 BEGIN {GetVerRev - get current "ver.rev"}
 H := StringHandle(GetResource('MPST', 0)); {Get 'MPST' rsrc }
 IF H = NIL THEN {Use string const.}
 VerRev := Version {if not found }
 ELSE
 BEGIN
 i := Pos('Version', H^^) + 8; {Start of ver.rev }
 VerRev := Copy(H^^, i, Length(H^^)-i+1); {Extract from rsrc}
 END;
 END; {GetVerRev}
```

Normally, SetVersion is used with its **-r** option as part of a makefile to automatically increment the revision number each time the application or tool is rebuilt. For each (major) release the version number should be incremented and the revision reset to 0. Note that when SetVersion modifies the application or tool, or updates a source file, the modification date is *not* changed. Therefore, makefiles will not be affected by the use of SetVersion.

|                    |                                                                                                                                                                                          |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Type</b>        | Tool.                                                                                                                                                                                    |
| <b>Input</b>       | The <i>file</i> parameter specifies the filename of an application or tool containing the 'MPST' string resource.                                                                        |
| <b>Output</b>      | None.                                                                                                                                                                                    |
| <b>Diagnostics</b> | Errors are written to the diagnostic file.                                                                                                                                               |
| <b>Status</b>      | The following status values may be returned: <ul style="list-style-type: none"> <li>0 Normal termination</li> <li>1 Parameter or option error</li> <li>2 Execution terminated</li> </ul> |

## Options

**-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 the 'MPST' resource string in the application or tool. It is assumed that the constant is defined as a string constant in a #define, somewhere in the first 12800 characters (25 512-byte blocks) of the file, as follows:

```
#defineΔVersion "ver.rev"ΔΔΔΔΔΔΔΔΔΔ/*some comment*/
```

The Δ's 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 and revision 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** may be used to search for a different #define identifier if desired.

**-d** Write the (updated) version and revision values contained in the 'MPST' resource string to the diagnostic output file.

**-fmt nf.mf** 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.

**SetVersion Options**

**Application or MPW Tool Name**

**Source Files**

Pascal Source

C Source

Rez Source

**Resource Attributes**

Resource Type

Resource ID

**Options**

Display  Increment version

Progress  Increment revision

Set Version  Set Revision

**Layout**

Format  Prefix

Suffix

Version Id  Error

Version

**Command Line**

SetVersion

**Help**

Tool or application version/revision number ("ver.rev") maintainer.  
Generates/maintains a string resource in a tool or application.



**-resource file** Update the 'MPST' resource definition in the Resource Compiler source specified by the *file*. The definition is set to be the same as that specified by the 'MPST' resource string in the application or tool. It is assumed that the definition is somewhere in the first 12800 characters (25 512-byte blocks) of the file and is specified as follows:

```
type 'MPST' as 'STRΔ';
resourceΔ'MPST' (0) {
 "Version ver.rev"ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔ/*some comment*/
};
```

The Δ's 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 and revision values—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'" in the source. Note that, because this is a resource definition and destined to be placed in the application's or tool's resource fork, this option defines the actual string resource that SetVersion will seek in the application or tool. The "Version" in the string here is fixed, and *not* controlled by the **-verid** option.

- sr revision** Set the revision to the specified *revision* integer value.
- suffix suffix** Set the suffix string on the revision. The *suffix* may be any sequence of characters that does not begin with a digit (0-9). Once the suffix is set, it can be changed only by specifying another **-suffix** string, or removed by specifying the *suffix* as a period (.).
- sv version** Set the version to the specified *version* integer value.
- t type** Use the specified resource *type* instead of 'MPST'. (The **-i** option can be used to specify the resource ID.)
- v** Increment the version by 1.
- verid identifier** Use the specified constant identifier when searching for the **-[p]source CONST** identifier or **-csource #define** identifier.

### Example

```
setversion -d -sv 1 -r Asm -psource GlobalDcls -resource Asm.r
```

Increments the revision for the MPW Assembler (**-r**) in the resource fork of the file *Asm*. The version is fixed at 1 (**-sv**), so that *Asm* will display the version and revision as "1.rev" The Pascal include file, *GlobalDcls*, contains the Assembler's global declarations, including the Version string. This include file is updated to match the 'MPST' resource (**-psource**). The resource definitions for the Assembler, in *Asm.r*, will be similarly updated (**-resource**). Finally, this command displays the new version of the diagnostic output file (**-d**).

---

---

## Shift—renumber script parameters

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Shift [ <i>number</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Description</b> | Renames the command script positional parameters { <i>number</i> +1}, { <i>number</i> +2}... to {1}, {2}, and so on. If <i>number</i> 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.                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Status</b>      | The following status values may be returned:<br>0 Success<br>1 Syntax error                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Examples</b>    | <p>The following script repeats a command once for each parameter:</p> <pre>### Repeat - Repeat a command for several parameters ###<br/>#<br/># Repeat command parameter..<br/># Execute command once for each parameter in the<br/># parameter list. You can specify options by<br/># including them in quotes with the command name.<br/>#<br/>Set cmd "{1}"<br/>Loop<br/>    Shift<br/>    Break If "{1}" == ""<br/>    {cmd} "{1}"<br/>End</pre> <p>Here 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 this Repeat script to compile several C programs with progress information:</p> <pre>Repeat 'C -p' Sample.c Count.c Memory.c</pre> |
| <b>See also</b>    | "Parameters" in Chapter 5.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

---

---

## Shutdown—shutdown or software reboot

|                    |                                                                                                                                                                                                                                                    |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Shutdown [-y   -n   -c ][-r]                                                                                                                                                                                                                       |
| <b>Description</b> | 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. |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                          |
| <b>Input</b>       | None.                                                                                                                                                                                                                                              |
| <b>Output</b>      | None.                                                                                                                                                                                                                                              |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                           |
| <b>Status</b>      | The following status values may be returned:<br><br>1 Syntax error<br>2 Command aborted<br><br><i>Note:</i> Shutdown cannot return a status of 0 because if there are no errors the command never returns.                                         |
| <b>Options</b>     | <p>-y Answer "yes" to any confirmation dialog that occurs.</p> <p>-n Answer "no" to any confirmation dialog that occurs.</p> <p>-c Answer "cancel" to any confirmation dialog that occurs.</p> <p>-r Restart the machine.</p>                      |
| <b>Example</b>     | Shutdown -y<br><br>Reboots the machine, answering "yes" to any dialogs such as those prompting to save files.                                                                                                                                      |
| <b>See also</b>    | Quit command.                                                                                                                                                                                                                                      |

---

---

## SizeWindow—set a window's size

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | SizeWindow <b>h v</b> [ <i>window</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Description</b> | Sets the size of the specified <i>window</i> to be <b>h</b> by <b>v</b> pixels, where <b>h</b> and <b>v</b> are nonnegative integers referring to the horizontal and vertical dimensions respectively. (Use a blank space to separate the numbers <b>h</b> and <b>v</b> on the command line.) The default <i>window</i> is the target (second from the front) window; a specific <i>window</i> can optionally be specified. If the size specified would cause the window to be too big or small, an error is returned. |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Status</b>      | SizeWindow may return the following status values:<br>0 No errors<br>1 Syntax error (error in parameters)<br>2 The specified window does not exist<br>3 The <b>h,v</b> size specified is invalid, that is, too big or too small                                                                                                                                                                                                                                                                                        |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Examples</b>    | <pre>SizeWindow 200 200</pre> Makes the target window 200 pixels square in size.<br><br><pre>SizeWindow 500 100 "{Worksheet}"</pre> Makes the Worksheet window 500 x 100 pixels in size.                                                                                                                                                                                                                                                                                                                               |
| <b>See also</b>    | MoveWindow, ZoomWindow, StackWindows, and TileWindows commands.                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

---

---

## StackWindows—arrange windows diagonally

|                    |                                                                                                                                                                                                                                      |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | StackWindows                                                                                                                                                                                                                         |
| <b>Description</b> | Automatically sizes and moves all of the open Shell windows 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. |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                            |
| <b>Input</b>       | None.                                                                                                                                                                                                                                |
| <b>Output</b>      | None.                                                                                                                                                                                                                                |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                             |
| <b>Status</b>      | The following status values may be returned:<br>0 No errors<br>1 Syntax error (in parameters)                                                                                                                                        |
| <b>Options</b>     | None.                                                                                                                                                                                                                                |
| <b>Example</b>     | StackWindows<br>Stacks all of the Shell windows in a neat and orderly fashion.                                                                                                                                                       |
| <b>See also</b>    | TileWindows, SizeWindow, ZoomWindow, and MoveWindow commands.                                                                                                                                                                        |



---

---

## Tab—set a window's tab value

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Tab <i>number</i> [ <i>windows...</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Description</b> | <p>Sets the tab setting of the files in <i>window</i> to <i>number</i> spaces. If no windows are 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.</p> <p><i>Note:</i> The Tab command (and the Tabs menu item) modify the tab setting of an existing window. The {Tab} variable is used to initialize the tab setting of a new window, or as the default for files with no tab setting.</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Status</b>      | <p>Tab may return the following status values:</p> <ul style="list-style-type: none"><li>0 No errors</li><li>1 Syntax error</li><li>2 An illegal tab count was specified</li></ul>                                                                                                                                                                                                                                                                                                        |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Example</b>     | <p>Tab 4</p> <p>Sets the tab value of the target window to represent 4 spaces.</p>                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>See also</b>    | Entab command.                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

---

---

## Target—make a window the target window

|                    |                                                                                                                                                                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Target <i>name</i>                                                                                                                                                                                                                                               |
| <b>Description</b> | Makes window <i>name</i> the target window for editing commands (that is, the second window from the front). If window <i>name</i> isn't already open, then file <i>name</i> is opened as the target window. If <i>name</i> doesn't exist, an error is returned. |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                        |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                            |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                            |
| <b>Diagnostics</b> | Error messages are written to diagnostic output.                                                                                                                                                                                                                 |
| <b>Status</b>      | The following status values may be returned:<br>0 No errors<br>1 Error in parameters<br>2 The specified file does not exist<br>3 System error                                                                                                                    |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                            |
| <b>Example</b>     | Target Sample.a<br>Makes the window Sample.a the target window.                                                                                                                                                                                                  |
| <b>See also</b>    | Open command.<br>“Editing With the Command Language” in Chapter 5.                                                                                                                                                                                               |

---

---

## TileWindows—arrange windows in tile pattern

|                    |                                                                                                                                                                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | TileWindows                                                                                                                                                                                                                                         |
| <b>Description</b> | Automatically sizes and moves all open Shell windows so that they are all visible on the screen at once. Arranging your open windows like tiles, and then zooming a selected window to full size, makes dealing with many open windows much easier. |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                           |
| <b>Input</b>       | None.                                                                                                                                                                                                                                               |
| <b>Output</b>      | None.                                                                                                                                                                                                                                               |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                            |
| <b>Status</b>      | TileWindows returns the following status values:<br>0 No errors<br>1 Syntax error (error in parameters)                                                                                                                                             |
| <b>Options</b>     | None.                                                                                                                                                                                                                                               |
| <b>Example</b>     | TileWindows<br>Arranges all of the Shell windows in a tile pattern, so that all are visible.                                                                                                                                                        |
| <b>See also</b>    | SizeWindow, ZoomWindow, StackWindows, and MoveWindow commands.                                                                                                                                                                                      |

---

---

## TLACvt—convert Lisa TLA Assembler source

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | TLACvt [ <i>option...</i> ] [ <i>sourceFile...</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Description</b> | <p>Converts the specified Lisa Workshop TLA Assembler source files to the syntax required by the MPW Assembler. If the input file name is <i>name</i>, the converted output is written to <i>name.a</i>. The following elements are converted:</p> <ul style="list-style-type: none"><li><input type="checkbox"/> tokens within statements</li><li><input type="checkbox"/> special tokens within macros</li><li><input type="checkbox"/> directives</li></ul> <p>For the details of these conversions, see “TLA Conversion” in Appendix E of the companion volume <i>MPW Assembler 2.0 Reference</i>.</p> <p>The case (upper/lower) of directive names in the output may be controlled by editing the file TLACvt.Directives. This file contains a list of all the MPW Assembler directives needed for conversion. The pathname to this file must be specified with the <b>-f</b> option.</p> |
| <b>Type</b>        | Tool.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Input</b>       | If no filenames are specified, standard input is converted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Output</b>      | If input is from the standard input file, the converted output is written to standard output. If the input file name is <i>name</i> , the converted output is written to <i>name.a</i> . You can use the <b>-n</b> , <b>-prefix</b> , and <b>-suffix</b> options to modify the output file naming conventions.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Diagnostics</b> | Parameter errors and progress information are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Status</b>      | <p>The following status values may be returned:</p> <ul style="list-style-type: none"><li>0 Normal termination</li><li>1 Parameter or option error</li><li>2 Execution terminated</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Options</b>     | <ul style="list-style-type: none"><li><b>-d</b> Detab the input. All tabs are removed and replaced with spaces. The number of spaces is determined by the tab setting. (See the <b>-t</b> option below.)</li><li><b>-e</b> Detab the input (as done by the <b>-d</b> option) and entab the output as a function of the tab setting. (See the <b>-t</b> option below.)</li><li><b>-f directivesFile</b> The casing of directives is controlled by the file of directives specified by <i>directivesFile</i>. The file TLACvt.Directives is supplied for this purpose; you can edit it to change the capitalization. By default, all directives are converted to uppercase.</li><li><b>-m</b> Do <i>not</i> insert TLA-compatible mode-setting directives (BLANKS ON and STRING ASIS) into converted source.</li></ul>                                                                           |

- n** Do not add the “.a” extension to the input filename to produce the output filename. If you specify this option, you must also specify **-prefix** or **-suffix**.
- p** Writes TLACvt’s version information and conversion status to diagnostic output.
- pre[fix] string** If the input filename is Name, the output filename is produced by prefixing *string* to the name, that is, “stringName.a”. (The “.a” suffix may be suppressed by using the **-n** option or changed by using the **-suffix** option.)
- suf[fix] string** If the input filename is Name, the output filename is produced by appending *string* to the filename, that is, “Namestring”. The default suffix is “.a”.
- t tabSetting** Set the output file’s tab value to *tabSetting* (2 to 255). The default is to use the input file’s tab setting if there is one; otherwise a value of 8 is assumed. (8 is the default used by the Lisa Workshop’s MacCom utility when transferring text files—it’s assumed that MacCom was used to transfer the TLA files from the Lisa to the Macintosh.)
- u c** When TLACvt detects a name in the opcode field that is the same as an MPW directive, it appends the character *c* to make the name unique. (The default character is #.)

**Example**

```
TLACvt -t 8 TLAFile1.Text TLAFile2.Text
```

Converts the Lisa TLA Assembler source files TLAFile1.Text and TLAFile2.Text to the MPW Assembler source files TLAFile1.Text.a and TLAFile2.Text.a. The **-t** option sets the tab setting for both input files to 8, and entabs the output files based on a tab setting of 8.

**Limitations**

Limitations are noted in the detailed description of TLA conversions in the *MPW 2.0 Assembler Reference*.

**See also**

CvtObj command.  
 Appendix E, “TLA Conversion,” in the *MPW Assembler 2.0 Reference*.

---

---

## Translate—convert selected characters

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Translate [ options ] <i>src</i> [ <i>dst</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Description</b> | <p>Standard input is copied to standard output, with input characters specified in the <i>src</i> (<i>source</i>) parameter string mapped into the corresponding characters specified by the <i>dst</i> (<i>destination</i>) parameter string; all other characters are copied as is. If <i>dst</i> is omitted, all characters represented by the <i>src</i> are deleted. If the <i>dst</i> string is shorter than the <i>src</i>, all characters in the <i>src</i> that would map to or beyond the last character in the <i>dst</i> are mapped into the last character in <i>dst</i>, and adjacent instances of such characters in the input are represented by a single instance of the last character in <i>dst</i>.</p> <p>Both <i>src</i> and <i>dst</i> are specified as a standard Shell <i>character list</i> but not enclosed in square brackets. Thus the <i>src</i> and <i>dst</i> are a sequence 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 <math>\partial t</math>, <math>\partial n</math>, <math>\partial f</math>) are processed by the Shell command interpreter. In order to specify a minus sign, place it last in the character list. Finally, the <i>src</i> character list may be preceded by a <math>\neg</math> to negate the list; that is, all characters <i>except</i> those in the <i>src</i> are taken as the <i>src</i> string. Thus they are all deleted if <i>dst</i> is absent, or collapsed if the last character in <i>dst</i> is present.</p> <p><i>Note:</i> Case sensitivity of letters specified in the <i>src</i> list are governed by the {CaseSensitive} Shell variable. If CaseSensitive is set to 1, then only letters specified in the <i>src</i> are mapped or deleted. If CaseSensitive is 0, then uppercase and lowercase letters not explicitly mapped into <i>dst</i> characters are mapped identically.</p> |
| <b>Type</b>        | Tool.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Input</b>       | All input is read from the standard input file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Output</b>      | The translated input file is written to standard output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Status</b>      | Translate may return the following status values:<br>0 Normal termination<br>1 Parameter or option error<br>2 Execution terminated                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

## 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's.

The screenshot shows a dialog box titled "Translate Options". It contains several input fields and checkboxes. At the top, there are two fields: "Input characters to translate" and "Corresponding output characters". Below these are two checkboxes: "Progress" and "Set output font/tab". Further down, there are three fields labeled "Input", "Output", and "Error". At the bottom of the dialog, there is a "Command Line" section with the text "Translate" and a "Help" section with the text "Copies standard input to standard output with the substitution of specified characters." To the right of the "Help" section are two buttons: "Cancel" and "Translate".

## 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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Unalias [ <i>name...</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Description</b> | <p>Removes any alias definition associated with the alias <i>name</i>. (It is not an error if no definition exists for <i>name</i>.)</p> <p><b>Caution:</b> If no names are specified, all aliases are removed.</p> <p>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</p> <pre>Unalias</pre> <p>at the beginning of your script to make sure no unwanted substitutions occur.</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Diagnostics</b> | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Status</b>      | A status value of 0 is always returned.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Example</b>     | <pre>Unalias File</pre> <p>Remove the alias "File". (This alias is defined in the Startup file.)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>See also</b>    | Alias command.<br>"Command Aliases" in Chapter 5.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |



---

---

## Undo—undo last edit

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Undo [ <i>window</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Description</b> | <p>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 may be specified.</p> <p><i>Note:</i> 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.</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Status</b>      | <p>Undo may return the following status values:</p> <ul style="list-style-type: none"><li>0 No errors</li><li>1 Syntax error (error in parameters)</li><li>2 Any other error</li></ul>                                                                                                                                                                                                                                                                                                                                  |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Examples</b>    | <p>Undo<br/>Reverses the last edit operation in the target window.</p> <p>Undo "{Worksheet}"<br/>Reverses the last edit operation in the Worksheet window.</p>                                                                                                                                                                                                                                                                                                                                                          |
| <b>See also</b>    | Cut, Copy, and Paste commands.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

---

---

## Unexport—remove a variable definition from export

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Unexport [-r   -s   <i>name...</i> ]                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Description</b> | <p>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.</p> <p>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.)</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Output</b>      | If no names are given, Unexport writes a list of unexported variables to standard output.                                                                                                                                                                                                                                                                                                                                      |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Status</b>      | Unexport may return the following status values:<br>0 No error<br>1 Syntax error                                                                                                                                                                                                                                                                                                                                               |
| <b>Options</b>     | <p>-r Reverse the sense of the output, causing Unexport to generate Export commands for all unexported variables.</p> <p>-s Suppress the printing of "Unexport" before the unexported variables.</p>                                                                                                                                                                                                                           |

## 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

|                    |                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Unmark <i>name... window</i>                                                                                                                                                                                                                                                                                                                                      |
| <b>Description</b> | Unmark removes the marker(s) <i>name...</i> ,from the list of markers available for <i>window</i> . When a window is the current active window, the Mark menu item(s) will be adjusted.                                                                                                                                                                           |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                         |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                             |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                             |
| <b>Diagnostics</b> | Errors and warnings are written to the diagnostic output.                                                                                                                                                                                                                                                                                                         |
| <b>Status</b>      | The following status values may be returned<br>0 No errors<br>1 Syntax error<br>2 Error in processing<br>3 System error                                                                                                                                                                                                                                           |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                             |
| <b>Examples</b>    | <pre>Unmark `Markers` "{Target}"</pre> <p>Removes all markers associated with the target window.</p> <pre>Unmark Proc1 "{Active}"</pre> <p>Removes the "Proc1" marker from the active window's marker list. Because {Active} is, by definition, the current active window, the Mark menu will also be adjusted to reflect the deletion of the "Proc1" marker.</p> |
| <b>Limitation</b>  | Unmark does not support Undo.                                                                                                                                                                                                                                                                                                                                     |
| <b>See also</b>    | "Markers" in Chapter 6.                                                                                                                                                                                                                                                                                                                                           |

---

---

## Unmount—unmount volumes

|                    |                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Unmount <i>volume</i> ...                                                                                                                                                                                                                                                                                                                                           |
| <b>Description</b> | Unmounts the specified volumes. A volume name must end with a colon (:). If <i>volume</i> 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), then the boot volume becomes the current volume. |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                           |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                               |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                               |
| <b>Diagnostics</b> | Error messages are written to diagnostic output.                                                                                                                                                                                                                                                                                                                    |
| <b>Status</b>      | The following status values may be returned:<br>0 The volume was successfully unmounted<br>1 Syntax error<br>2 An error occurred                                                                                                                                                                                                                                    |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                               |
| <b>Examples</b>    | Unmount Memos:<br>Unmounts the volume titled Memos.<br><br>Unmount 1 2<br>Unmounts the volumes in drives 1 (the internal drive) and 2 (the external drive). (The command Eject 1 2 would unmount <i>and</i> eject the volumes.)                                                                                                                                     |
| <b>See also</b>    | Eject and Mount commands.                                                                                                                                                                                                                                                                                                                                           |

---

---

## Unset—remove Shell variables

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Unset [ <i>name...</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Description</b> | <p>Removes any variable definition associated with <i>name</i>. (It's not an error if no definition exists for <i>name</i>.)</p> <p><b>Caution:</b> 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.)</p> <p>The scope of the Unset command is limited to the current script; that is, variables in enclosing scripts are not affected.</p> |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Diagnostics</b> | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Status</b>      | A status value of 0 is always returned.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Options</b>     | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Example</b>     | <pre>Unset CaseSensitive</pre> <p>Removes the variable definition for {CaseSensitive}. This turns off case-sensitive searching for the editing commands.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>See also</b>    | Set, Export, and Unexport commands.<br>"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** Error messages are written to diagnostic output.
- Status** The following status values may be returned:
- 0 No errors
  - 1 Syntax error
  - 2 No such volume
- Options**
- l List volumes in long format, giving volume name, drive (0 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
- | <u>Name</u> | <u>Drive</u> | <u>Size</u> | <u>Free</u> | <u>Files</u> | <u>Dirs</u> |
|-------------|--------------|-------------|-------------|--------------|-------------|
| HD:         | 3            | 19171K      | 14242K      | 290          | 33          |
- Files `Volumes 1`  
Lists the files on the disk in drive 1 (the built-in 3.5-disk drive).

---

---

## Which—determine which file the Shell will execute

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Which [-a] [-p] [ <i>command</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Description</b> | Determines which command the Shell will execute when <i>command</i> 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 <i>command</i> is not specified then 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. |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Output</b>      | In the case of a tool, application, or script, the full path of the command is written to standard output. If <i>command</i> is an alias its definition is written to standard output. If <i>command</i> is a built-in command then it is simply echoed back to standard output.                                                                                                                                                                                                                                                        |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Status</b>      | The following status values may be returned:<br>0 No error<br>1 Syntax error<br>2 <i>Command</i> not found<br>3 Other error                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Options</b>     | <b>-a</b> All paths to <i>command</i> are written to standard output. This option allows the user to determine if there are multiple commands with the same name.<br><b>-p</b> Prints progress information as each directory in the variable <i>commands</i> 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

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Windows [-q]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Description</b> | 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.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Input</b>       | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Output</b>      | The list of open windows is written to standard output.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Diagnostics</b> | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Status</b>      | Status value 0 is always returned.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Option</b>      | <b>-q</b> Don't quote window names that contain special characters. (The default is to quote names that contain spaces or other special characters.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Examples</b>    | <p><code>Windows</code></p> <p>Lists the pathnames of all open windows.</p> <p><code>Print {PrintOptions} `Windows`</code></p> <p>Prints all 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.</p> <p><code>Echo "Open `Windows`    Set Status 0" &gt; SavedWindows</code></p> <p>Writes a command 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 Suspend script to save the list of open windows.</p> |

---

---

## ZoomWindow—enlarge or reduce a window

|                    |                                                                                                                                                                                                                                                                                                                                                               |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | ZoomWindow [ -s ] [ <i>window</i> ]                                                                                                                                                                                                                                                                                                                           |
| <b>Description</b> | Zooms the specified <i>window</i> to full size on the screen. The default <i>window</i> is the target (second from the front) window; a specific <i>window</i> can optionally be specified. The -s option forces the window to zoom back to its small size. This capability is especially valuable when used in conjunction with StackWindows or TileWindows. |
| <b>Type</b>        | Built-in.                                                                                                                                                                                                                                                                                                                                                     |
| <b>Output</b>      | None.                                                                                                                                                                                                                                                                                                                                                         |
| <b>Diagnostics</b> | Errors are written to diagnostic output.                                                                                                                                                                                                                                                                                                                      |
| <b>Status</b>      | ZoomWindow may return the following status values:<br>0 No errors<br>1 Syntax error (error in parameters)<br>2 The specified window does not exist                                                                                                                                                                                                            |
| <b>Option</b>      | -s Zoom the specified <i>window</i> back to its original, smaller size.                                                                                                                                                                                                                                                                                       |
| <b>Examples</b>    | <pre>ZoomWindow</pre> <p>Zooms the target window to full size.</p> <pre>ZoomWindow -s "{Worksheet}"</pre> <p>Zooms the Worksheet window back to its small size.</p>                                                                                                                                                                                           |
| <b>See also</b>    | SizeWindow, MoveWindow, StackWindows, and TileWindows commands.                                                                                                                                                                                                                                                                                               |

