Choose Printer.

Addison-Wesley Publishing Company, Inc.
Reading, Massachusetts Menlo Park, California New York Don Mills, Ontario Wokingham, England Amsterdam Bonn Sydney Singapore
Tokyo Madrid San Juan
WARRANTY INFORMATION

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

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

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

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.
Volume 1

Preface
Roadmap to the Apple IIGs Technical Manuals xxiii

Introductory manuals xxv
The technical introduction xxv
The programmer's introduction xxv
Machine reference manuals xxvi
The hardware reference manual xxvi
The firmware reference manual xxvi
The toolbox manuals xxvi
The Programmer’s Workshop manual xxvii
Programming-language manuals xxvii
Operating-system manuals xxviii
All-Apple manuals xxviii
How to use this manual xxix
Other materials you'll need xxxi
Notations and conventions xxxi
Typographic conventions xxxi
Watch for these xxxi
Future toolbox enhancements xxxii

Chapter 1
Introducing the Apple IIGs Toolbox 1-1

What is a tool set? 1-1
What can the tool sets do for you? 1-1
Are there any limitations? 1-2
What kinds of tool sets are provided? 1-2
The big five 1-3
Desktop interface tool sets 1-3
Math tool sets 1-4
Printer tool set 1-4
Sound tool sets 1-4
Specialized tool sets 1-4
Groups of routines within each tool set 1-5
Apple Desktop Bus Tool Set 1-5
Control Manager 1-5
Desk Manager 1-5
Dialog Manager 1-6
Event Manager 1-6
Font Manager 1-7
Integer Math Tool Set 1-7
LineEdit Tool Set 1-8
List Manager 1-8
Memory Manager 1-9
Menu Manager 1-9
Miscellaneous Tool Set 1-10
Print Manager 1-11
QuickDraw II 1-11
QuickDraw II Auxiliary 1-14
SANE Tool Set 1-14
Scheduler 1-14
Scrap Manager 1-15
Sound Tool Set 1-15
Standard File Operations Tool Set 1-15
Text Tool Set 1-16
Tool Locator 1-16
Window Manager 1-17

Chapter 2 Using the Apple IIGs Tool Sets 2-1
Starting up the required tool sets 2-1
Loading and starting up other tool sets 2-3
Calling the correct routine 2-5
Calling a routine from assembly language 2-5
Calling a routine from C 2-6
Passing parameters 2-6
Return from a call 2-7

Chapter 3 Apple Desktop Bus Tool Set 3-1
A preview of the Apple Desktop Bus Tool Set routines 3-1
About the Apple Desktop Bus commands 3-2
Using other Apple Desktop Bus devices 3-3
Polling the Apple Desktop Bus for data 3-3
Polling single-user applications 3-3
Polling multiuser applications 3-3
The ADB Change Address When Activated handler 3-4
The Collision Detect handler 3-4
Using the Apple Desktop Bus Tool Set 3-5
Completion routines 3-7
  AsyncADBReceive completion routine 3-8
  SRQ list completion routine 3-9
Apple Desktop Bus Tool Set housekeeping routines 3-10
Apple Desktop Bus Tool Set routines 3-13
Apple Desktop Bus Tool Set summary 3-28

Chapter 4  Control Manager 4-1
A preview of the Control Manager routines 4-1
Standard controls 4-3
Scroll bars 4-5
Active, inactive, and highlighted controls 4-7
Controls and windows 4-8
Part codes 4-8
Using the Control Manager 4-9
Control Manager icon font 4-11
Control records 4-11
  Simple button control records 4-14
  Check box control record 4-16
  Radio button control record 4-18
  Scroll bar control record 4-20
  Size box control record 4-23
Defining your own controls 4-24
  Draw routine 4-26
  Test routine 4-27
  Calculate indicator rectangle routine 4-28
  Initialize routine 4-29
  Dispose routine 4-30
  Position routine 4-31
  Thumb routine 4-32
  Drag routine 4-35
  Track routine 4-36
  New value routine 4-37
  Set parameters routine 4-38
  Move routine 4-39
  Record size routine 4-40
Control Manager housekeeping routines 4-41
Control Manager routines 4-45
Control Manager summary 4-85
Chapter 5  Desk Manager  5-1

A preview of the Desk Manager routines  5-1
Using classic desk accessories  5-3
  When the CDA menu can be displayed  5-3
  Writing classic desk accessories  5-3
Supporting new desk accessories  5-5
  Supporting new desk accessories  
    with TaskMaster  5-5
  Supporting new desk accessories  
    without TaskMaster  5-6
  Writing new desk accessories  5-6
Desk Manager housekeeping routines  5-9
Desk Manager routines  5-12
Desk Manager summary  5-30

Chapter 6  Dialog Manager  6-1

A preview of the Dialog Manager routines  6-1
Dialog boxes  6-4
Dialog and alert windows  6-7
Item templates  6-8
  Item types  6-10
    Item descriptor and item value  6-12
      MyItem  6-16
  Display rectangle  6-17
  Item ID  6-18
  Item flag  6-18
  Item color tables  6-18
Dialog records  6-19
Alerts  6-19
  MySound  6-22
Using the Dialog Manager  6-23
Filter procedures  6-25
    MyFilter  6-25
Dialog Manager housekeeping routines  6-27
Dialog Manager routines  6-31
Dialog Manager summary  6-88

Chapter 7  Event Manager  7-1

A preview of the Event Manager routines  7-1
Two managers in one  7-3
Event types  7-3
  Mouse events  7-3
  Keyboard events  7-3
  Window events  7-4
  Other events  7-4
Event priority 7-4
Event records 7-6
  Event codes 7-7
  Event messages 7-8
  Modifier flags 7-8
Event masks 7-10
Using the Event Manager 7-12
  Responding to mouse events 7-13
  Responding to keyboard events 7-13
  Responding to window events 7-14
  Responding to other events 7-14
  Posting and removing events 7-14
  Performing other operations 7-14
Capturing asynchronous key events 7-15
Journaling mechanism 7-19
Using alternative pointing devices 7-21
  Writing device drivers 7-21
  Installing device drivers 7-23
    Devices using their own cards 7-23
    Devices communicating
      through the serial port 7-24
    Devices communicating
      through the Apple Desktop Bus 7-24
Removing device drivers 7-25
  Devices using their own cards 7-25
  Devices communicating
    through the serial port 7-25
  Devices communicating
    through the Apple Desktop Bus 7-25
Event Manager housekeeping routines 7-26
Event Manager routines 7-31
Event Manager summary 7-50

Chapter 8 Font Manager 8-1
  A preview of the Font Manager routines 8-2
  Font records and font families 8-3
    Family names and numbers 8-3
    Font size 8-4
    Font style 8-5
    Font ID record 8-6
    Base families 8-7
    Real and scaled fonts 8-7
    Current and system fonts 8-8
Chapter 9  Integer Math Tool Set 9-1
A preview of the Integer Math Tool Set routines 9-1
Rounding and pinning 9-3
Using the Integer Math Tool Set 9-4
Integer Math Tool Set housekeeping routines 9-5
Integer Math Tool Set routines 9-8
Integer Math Tool Set summary 9-42

Chapter 10 LineEdit Tool Set 10-1
A preview of the LineEdit Tool Set routines 10-2
Edit records 10-4
The leDestRect and leViewRect fields 10-6
The leLineHite and leBaseHite fields 10-7
The leSelStart and leSelEnd fields 10-7
The leHiliteHook and leCaretHook fields 10-9
Using the LineEdit Tool Set 10-9
Moving or scrolling windows
that contain LineEdit items 10-11
LineEdit Tool Set housekeeping routines 10-12
LineEdit Tool Set routines 10-16
LineEdit Tool Set summary 10-47

Chapter 11 List Manager 11-1
A preview of the List Manager routines 11-1
List controls and list records 11-2
List control records 11-8
Using the List Manager 11-11
Selection modes 11-12
List Manager housekeeping routines 11-13
List Manager routines 11-16
List Manager summary 11-25
Chapter 14  Miscellaneous Tool Set  14-1
   A preview of the Miscellaneous Tool Set routines  14-1
   Using the Miscellaneous Tool Set  14-4
   Miscellaneous Tool Set housekeeping routines  14-6
   Miscellaneous Tool Set routines  14-9
   Miscellaneous Tool Set summary  14-64

Chapter 15  Print Manager  15-1
   A preview of the Print Manager routines  15-2
   Print dialog boxes  15-4
      Choose Printer dialog box  15-4
      Style dialog box  15-5
      Job dialog box  15-8
   Print records  15-9
      Printer information subrecord  15-11
      Style subrecord  15-12
         ImageWriter style subrecord values  15-13
         LaserWriter style subrecord values  15-13
      Job subrecord  15-14
   Printing modes and resolutions  15-15
   Using the Print Manager  15-19
      Printing loop  15-20
      Printing a specified range of pages  15-21
      Using QuickDraw II for printing  15-21
      Sequence of events  15-22
   Methods of printing  15-23
   Printer and port drivers  15-23
      Printer drivers  15-23
      Printer peripheral cards and printer ports  15-24
   Background processing  15-24
   Print Manager housekeeping routines  15-25
   Print Manager routines  15-29
   Print Manager summary  15-47
Contents

Volume 2

Figures and tables (Volume 2) xvii

Chapter 16  QuickDraw II 16-1

A preview of the QuickDraw II routines 16-1
Drawing to the screen and elsewhere 16-9
Where QuickDraw II draws 16-9
  Coordinate plane 16-10
  Pixel images and the coordinate plane 16-12
  GrafPort, port rectangle, and clipping 16-14
  Global and local coordinate systems 16-16
How QuickDraw II draws 16-18
  Drawing pen 16-18
  Basic drawing functions 16-20
What QuickDraw II draws 16-21
  Points and lines 16-21
  Rectangles 16-22
  Circles, ovals, arcs, and wedges 16-23
  Polygons 16-24
  Regions 16-25
  Pictures 16-25
Drawing text 16-26
  Simple text manipulation 16-26
Drawing in color 16-31
  Color tables and palettes 16-32
  Scan line control bytes 16-34
  Standard color palette in 320 mode 16-35
  Dithered colors in 640 mode 16-35
Cursors 16-37
Using QuickDraw II 16-39
Contents

Chapter 17  QuickDraw II Auxiliary  17-1
  A preview of the QuickDraw II Auxiliary routines  17-1
  About pictures  17-2
  Style modification support  17-3
  QuickDraw II Auxiliary icon record  17-3
  Using QuickDraw II Auxiliary  17-5
  QuickDraw II Auxiliary housekeeping routines  17-6
  QuickDraw II Auxiliary routines  17-9
  QuickDraw II Auxiliary summary  17-16

Chapter 18  SANE Tool Set  18-1
  A preview of the SANE Tool Set routines  18-2
  Using the SANE Tool Set  18-3
    Performance characteristics and limitations  18-6
  Differences between 65C816 and 6502 SANE  18-7
  SANE Tool Set housekeeping routines  18-11
  SANE Tool Set routines  18-15
  SANE Tool Set summary  18-15
Chapter 19  Scheduler  19-1
A preview of the Scheduler routines  19-1
Using the Scheduler  19-2
Scheduler housekeeping routines  19-4
Scheduler routines  19-7
Scheduler summary  19-8

Chapter 20  Scrap Manager  20-1
A preview of the Scrap Manager routines  20-2
Memory and the desk scrap  20-3
Desk scrap data types  20-3
Using the Scrap Manager  20-4
Setting up a private scrap  20-5
Scrap Manager housekeeping routines  20-7
Scrap Manager routines  20-10
Scrap Manager summary  20-19

Chapter 21  Sound Tool Set  21-1
A preview of the Sound Tool Set routines  21-1
Sound hardware  21-3
Oscillators and generators  21-5
Using the Sound Tool Set  21-6
Sound Tool Set housekeeping routines  21-7
Sound Tool Set routines  21-11
Sound Tool Set low-level routines  21-29
Sound Tool Set summary  21-36

Chapter 22  Standard File Operations Tool Set  22-1
A preview of the Standard File Operations Tool Set routines  22-1
Standard dialog boxes  22-3
Standard File dialog templates  22-4
  Templates for the standard Open File dialog box  22-4
    640 mode  22-5
    320 mode  22-6
  Templates for the standard Save File dialog box  22-8
    640 mode  22-8
    320 mode  22-10
Using the Standard File Operations Tool Set  22-13
Standard File Operations Tool Set housekeeping routines  22-15
Standard File Operations Tool Set routines  22-20
Standard File Operations Tool Set summary  22-32
Chapter 23  Text Tool Set  23-1
A preview of the Text Tool Set routines 23-1
Using the I/O directing routines 23-3
Using the text routines 23-4
Using the Text Tool Set 23-9
Text Tool Set housekeeping routines 23-10
Text Tool Set routines 23-15
Text Tool Set summary 23-46

Chapter 24  Tool Locator  24-1
A preview of the Tool Locator routines 24-1
Using the Tool Locator 24-3
Tool Locator housekeeping routines 24-4
Tool Locator routines 24-7
Tool Locator summary 24-26

Chapter 25  Window Manager  25-1
A preview of the Window Manager routines 25-2
Window frames and controls 25-6
Window regions 25-9
Data and content areas and scroll bars 25-9
Using the Window Manager 25-10
Using TaskMaster 25-12
Window Manager icon font 25-15
Window record 25-15
Windows and GrafPorts 25-17
Window frame colors and patterns 25-17
How a window is drawn 25-20
  Draw content routine 25-21
  Draw information bar routine 25-21
Making a window active: activate events 25-24
Defining your own windows 25-25
  wDraw: draw a window frame 25-27
  wHit: find what region a point is in 25-28
  wCalcRgns: calculate a window's regions 25-28
  wNew: perform additional initialization 25-28
  wDispose: remove a window 25-29
  wGrow: draw the outline of a window 25-29
Origin movement 25-29
Window Manager housekeeping routines 25-32
Window Manager routines 25-35
Window Manager summary 25-139
Appendix A  Writing Your Own Tool Set  A-1
   Structure of the Tool Locator  A-2
   Tool set numbers and function numbers  A-3
   Obtaining memory  A-4
   Tool Locator system initialization  A-5
   Disk and RAM structure of tool sets  A-5
   Installing your tool set  A-6
   Function execution environment  A-10

Appendix B  Tool Set Error Codes  B-1

Appendix C  Tool Set Dependencies and Startup Order  C-1
   Tool set dependencies  C-1
   Tool set startup order  C-6

Appendix D  List of Routines by Tool Set Number
       and Routine Number  D-1

Glossary (Volumes 1 and 2)  G-1
Index (Volumes 1 and 2)  I-1
Figures and tables

Chapter 16  QuickDraw II  16-1

Figure 16-1  QuickDraw II coordinate plane  16-11
Figure 16-2  Grid lines, a point, and a pixel
             on the coordinate plane  16-12
Figure 16-3  The locInfo record  16-13
Figure 16-4  Pixel image and boundary rectangle  16-14
Figure 16-5  Boundary rectangle/port rectangle
             intersection  16-15
Figure 16-6  Drawing different parts of a document
             by changing local coordinates  16-17
Figure 16-7  Drawing with pattern and mask  16-19
Figure 16-8  How pen mode affects drawing  16-19
Figure 16-9  What QuickDraw II draws  16-21
Figure 16-10 Drawing a line  16-22
Figure 16-11 Rectangle  16-23
Figure 16-12 Rounded-corner rectangle  16-23
Figure 16-13 Oval  16-23
Figure 16-14 Arc  16-24
Figure 16-15 Polygon  16-24
Figure 16-16 Region  16-25
Figure 16-17 Character  16-27
Figure 16-18 Character bounds rectangle  16-28
Figure 16-19 Master color value  16-31
Figure 16-20 Accessing the color table in 320 mode  16-32
Figure 16-21 Accessing the color table in 640 mode  16-33
Figure 16-22 Scan line control byte  16-34
Figure 16-23 Fill mode example  16-34
Figure 16-24 Cursor record  16-37
Figure 16-25 Font definition  16-42
Figure 16-26 Character with no kerning  16-46
Figure 16-27 Character kerning left  16-46
Figure 16-28 Font rectangle (simulated)  16-47
Figure 16-29 Part of a font strike  16-48
Figure 16-30 Character rectangle in font rectangle  16-51
Figure 16-31 Character bounds rectangle  16-52
Figure 16-32 Font bounds rectangle  16-53
Figure 16-33 ROM font record  16-142
Figure 16-34 Standard SCB  16-145
Figure 16-35 PaintPixels parameter block  16-191
Figure 16-36 BufDimRec  16-206
Figure 16-37 Font flags word  16-226
Figure 16-38 Pen state record  16-238
Figure 16-39 Text face flag  160-258
Table 16-1  QuickDraw II routines and their functions  16-2
Table 16-2  Pen modes  16-20
Table 16-3  Text modes  16-30
Table 16-4  Standard palette in 320 mode  16-35
Table 16-5  Standard palette in 640 mode  16-36
Table 16-6  QuickDraw II—other tool sets required  16-39
Table 16-7  Standard color tables  16-159
Table 16-8  Standard pen state  16-196
Table 16-9  Pen modes  16-235
Table 16-10  Text-only modes  16-260
Table 16-11  QuickDraw II constants  16-274
Table 16-12  QuickDraw II data structures  16-276
Table 16-13  QuickDraw II error codes  16-278

Chapter 17  QuickDraw II Auxiliary  17-1
Figure 17-1  QuickDraw II Auxiliary icon record  17-3
Figure 17-2  The displayMode word  17-4
Table 17-1  QuickDraw II Auxiliary routines and their functions  17-2
Table 17-2  QuickDraw II Auxiliary—other tool sets required  17-5

Chapter 18  SANE Tool Set  18-1
Figure 18-1  SANE return information  18-7
Figure 18-2  SANE direct page on halt  18-9
Table 18-1  SANE Tool Set routines and their functions  18-2
Table 18-2  SANE Tool Set—other tool sets required  18-3

Chapter 19  Scheduler  19-1
Table 19-1  Scheduler routines and their functions  19-2
Table 19-2  Scheduler—other tool sets required  19-2

Chapter 20  Scrap Manager  20-1
Table 20-1  Scrap Manager routines and their functions  20-2
Table 20-2  Public scrap types  20-4
Table 20-3  Scrap Manager—other tool sets required  20-4
Table 20-4  Scrap Manager constants  20-19
Table 20-5  Scrap Manager error codes  20-19
Chapter 21  Sound Tool Set 21-1

Figure 21-1  Sound hardware block diagram 21-3
Figure 21-2  Sound GLU registers 21-4
Figure 21-3  Generator status word 21-12
Figure 21-4  Channel-generator-type word 21-16
Figure 21-5  Sound parameter block 21-17
Figure 21-6  Stop-sound mask 21-19
Table 21-1  Sound Tool Set routines and their functions 21-2
Table 21-2  DOC register allocation 21-4
Table 21-3  Oscillator registers 21-5
Table 21-4  Sound Tool Set—other tool sets required 21-6
Table 21-5  Jump table addresses for Sound Tool Set
  low-level routines 21-23
Table 21-6  Sound Tool Set constants 21-36
Table 21-7  Sound Tool Set data structures 21-37
Table 21-8  Sound Tool Set error codes 21-37

Chapter 22  Standard File Operations Tool Set 22-1

Figure 22-1  Standard Open File dialog box 22-3
Figure 22-2  Standard Save File dialog box 22-3
Figure 22-3  File directory entry 22-23
Figure 22-4  Typelist record 22-23
Figure 22-5  Reply record 22-24
Table 22-1  Standard File Operations Tool Set routines
  and their functions 22-2
Table 22-2  Standard File Operations Tool Set—
  other tool sets required 22-13
Table 22-3  Filter procedure results 22-22
Table 22-4  Standard File Operations Tool Set
  constants 22-32
Table 22-5  Standard File Operations Tool Set
  data structures 22-32

Chapter 23  Text Tool Set 23-1

Figure 23-1  Character echo-flag word 23-29
Table 23-1  Text Tool Set routines and their functions 23-1
Table 23-2  Character device driver types 23-3
Table 23-3  Text Tool Set—other tool sets required 23-9
Table 23-4  Text Tool Set constants 23-46
Table 23-5  Text Tool Set error codes 23-47
Chapter 24 Tool Locator 24-1

- Figure 24-1 Tool table 24-12
- Table 24-1 Tool Locator routines and their functions 24-2
- Table 24-2 Tool set numbers 24-13
- Table 24-3 MessageCenter message types 24-15
- Table 24-4 State record 24-18
- Table 24-5 Tool Locator constants 24-26
- Table 24-6 Tool Locator error codes 24-26

Chapter 25 Window Manager 25-1

- Figure 25-1 Window 25-1
- Figure 25-2 Window frames 25-6
- Figure 25-3 Standard window controls 25-7
- Figure 25-4 Sample document windows 25-7
- Figure 25-5 Proportional scroll bars 25-10
- Figure 25-6 TaskMaster's TaskRec (task record) 25-13
- Figure 25-7 The wmtaskMask bit flag 25-14
- Figure 25-8 Window record 25-16
- Figure 25-9 Document and alert window color table 25-17
- Figure 25-10 Window frame color (frameColor) 25-18
- Figure 25-11 Window title color (titleColor) 25-18
- Figure 25-12 Window title bar color (tBarColor) 25-19
- Figure 25-13 Window size box and alert window's middle outline color (growColor) 25-19
- Figure 25-14 Window information bar and alert window's inside outline color (infoColor) 25-20
- Figure 25-15 Window origin 25-29
- Figure 25-16 Window moving and origins 25-30
- Figure 25-17 Scrolling and window origins 25-31
- Figure 25-18 Grow image of a window 25-76
- Figure 25-19 NewWindow window frame type 25-85
- Figure 25-20 SetFrameColor newColorPtr and theWindowPtr values 25-100
Table 25-1  Window Manager routines and their functions  25-2
Table 25-2  Window Manager—other tool sets required  25-10
Table 25-3  Window Manager icon font  25-15
Table 25-4  The varCode parameters for custom windows  25-25
Table 25-5  Window Manager Desktop routine operations and parameters  25-40
Table 25-6  Desktop patterns  25-43
Table 25-7  DragWindow grid values  25-45
Table 25-8  FindWindow constants  25-49
Table 25-9  NewWindow parameter list  25-84
Table 29-10  Window global mask values  25-137
Table 29-11  Window global flag values  25-137
Table 25-12  Window Manager constants  25-139
Table 25-13  Window Manager data structures  25-142
Table 25-14  Window Manager error codes  25-144

Appendix A  Writing Your Own Tool Set A-1
Table A-1  Structure of a TPT (tool pointer table)  A-2
Table A-2  Structure of an FPT (function pointer table)  A-2
Table A-3  Standard tool set routine numbers  A-3
Table A-4  Tool Locator permanent RAM space  A-4

Appendix B  Tool Set Error Codes B-1
Table B-1  Tool set error codes  B-1

Appendix C  Tool Set Dependencies and Startup Order C-1
Table C-1  Tool set dependencies  C-2
Table C-2  Tool set startup order  C-6

Appendix D  List of Routines by Tool Set Number and Routine Number D-1
Table D-1  Routines by tool set/routine number  D-2
Any time your desktop application needs to draw something, it uses **QuickDraw II** (and its extension, QuickDraw II Auxiliary). QuickDraw II is an adaptation and extension of the Macintosh toolbox component QuickDraw—it performs similar operations but has been enhanced to support Apple IIGS color.

QuickDraw II allows your application to

- Perform graphic operations easily and quickly
- Draw lines and shapes of various sizes and patterns
- Draw items in a variety of colors or gray scales
- Draw text in different fonts and with styling variations, such as italics and boldface

---

**A preview of the QuickDraw II routines**

To introduce you to the capabilities of QuickDraw II, all QuickDraw II routines are grouped by function and briefly described in Table 16-1. These routines are described in detail later in this chapter, where they are separated into housekeeping routines (discussed in routine number order) and the rest of the QuickDraw II routines (discussed in alphabetical order).
### Table 16-1

QuickDraw II routines and their functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Housekeeping routines</strong></td>
<td></td>
</tr>
<tr>
<td>QDBootInit</td>
<td>Initializes QuickDraw II; called only by the Tool Locator—must not be called by an application</td>
</tr>
<tr>
<td>QDStartUp</td>
<td>Starts up QuickDraw II for use by an application</td>
</tr>
<tr>
<td>QDShutDown</td>
<td>Shuts down QuickDraw II when an application quits</td>
</tr>
<tr>
<td>QDVersion</td>
<td>Returns the version number of QuickDraw II</td>
</tr>
<tr>
<td>QDReset</td>
<td>Resets QuickDraw II; called only when the system is reset—must not be called by an application</td>
</tr>
<tr>
<td>QDStatus</td>
<td>Indicates whether QuickDraw II is active</td>
</tr>
<tr>
<td><strong>Global environment routines</strong></td>
<td></td>
</tr>
<tr>
<td>GetStandardSCB</td>
<td>Returns a copy of the standard scan line control byte (SCB)</td>
</tr>
<tr>
<td>SetMasterSCB</td>
<td>Sets the master SCB to a specified value</td>
</tr>
<tr>
<td>GetMasterSCB</td>
<td>Returns a copy of the master SCB</td>
</tr>
<tr>
<td>InitColorTable</td>
<td>Returns a copy of the standard color table for the current mode</td>
</tr>
<tr>
<td>SetColorTable</td>
<td>Sets a specified color table to specified values</td>
</tr>
<tr>
<td>GetColorTable</td>
<td>Fills a specified color table with the contents of another color table</td>
</tr>
<tr>
<td>SetColorEntry</td>
<td>Sets the value of a color in a specified color table</td>
</tr>
<tr>
<td>GetColorEntry</td>
<td>Returns the value of a specified color in a specified color table</td>
</tr>
<tr>
<td>SetSCB</td>
<td>Sets the SCB to a specified value</td>
</tr>
<tr>
<td>GetSCB</td>
<td>Returns the value of a specified SCB</td>
</tr>
<tr>
<td>SetAllSCBs</td>
<td>Sets all SCBs to a specified value</td>
</tr>
<tr>
<td>SetSysFont</td>
<td>Sets a specified font as the system font</td>
</tr>
<tr>
<td>GetSysFont</td>
<td>Returns a handle to the current system font</td>
</tr>
<tr>
<td>ClearScreen</td>
<td>Sets the words in screen memory to a specified value</td>
</tr>
<tr>
<td>GrafOn</td>
<td>Turns on Super Hi-Res graphics mode</td>
</tr>
<tr>
<td>GrafOff</td>
<td>Turns off Super Hi-Res graphics mode</td>
</tr>
<tr>
<td>SetBufDims</td>
<td>Sets the size of the QuickDraw II clipping and text buffers, padding the values to permit large values of (chEextra) and (spEextra) and to allow for style modifications</td>
</tr>
<tr>
<td>ForceBufDims</td>
<td>Sets the size of the QuickDraw II clipping and text buffers, but doesn't pad the values in any way</td>
</tr>
<tr>
<td>SaveBufDims</td>
<td>Saves QuickDraw II's buffer-sizing information in an eight-byte record</td>
</tr>
<tr>
<td>RestoreBufDims</td>
<td>Restores QuickDraw II's internal buffers to the sizes described in the eight-byte record created by the SaveBufDims routine</td>
</tr>
</tbody>
</table>
Table 16-1 (continued)
QuickDraw II routines and their functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GrafPort routines</strong></td>
<td></td>
</tr>
<tr>
<td>OpenPort</td>
<td>Initializes specified memory locations as a standard GrafPort, allocates a new visible region and a new clipping region, and makes the GrafPort the current port</td>
</tr>
<tr>
<td>InitPort</td>
<td>Initializes specified memory locations as a standard port</td>
</tr>
<tr>
<td>ClosePort</td>
<td>Deallocates the clipping and visible regions in a port</td>
</tr>
<tr>
<td>SetPort</td>
<td>Makes a specified port the current GrafPort</td>
</tr>
<tr>
<td>GetPort</td>
<td>Returns a pointer to the current GrafPort</td>
</tr>
<tr>
<td>SetPortLoc</td>
<td>Sets the current port's loctinfo record to specified location information</td>
</tr>
<tr>
<td>GetPortLoc</td>
<td>Gets the current port's loctinfo record and puts it at the specified location</td>
</tr>
<tr>
<td>SetPortRect</td>
<td>Sets the current GrafPort's port rectangle to the specified rectangle</td>
</tr>
<tr>
<td>GetPortRect</td>
<td>Returns the current GrafPort's port rectangle</td>
</tr>
<tr>
<td>SetPortSize</td>
<td>Changes the size of the current GrafPort's port rectangle</td>
</tr>
<tr>
<td>MovePortTo</td>
<td>Changes the location of the current GrafPort's port rectangle</td>
</tr>
<tr>
<td>SetOrigin</td>
<td>Adjusts the contents of the port rectangle and the bounds rectangle so the upper left corner of the port rectangle is set to the specified point</td>
</tr>
<tr>
<td>SetClip</td>
<td>Copies a specified region into the clipping region</td>
</tr>
<tr>
<td>GetClip</td>
<td>Copies the clipping region to a specified region</td>
</tr>
<tr>
<td>ClipRect</td>
<td>Changes the clipping region of the current GrafPort to a rectangle that is equivalent to a specified rectangle</td>
</tr>
<tr>
<td><strong>Pen and pattern routines</strong></td>
<td></td>
</tr>
<tr>
<td>HidePen</td>
<td>Decrements the pen level</td>
</tr>
<tr>
<td>ShowPen</td>
<td>Increments the pen level</td>
</tr>
<tr>
<td>GetPen</td>
<td>Returns the pen location</td>
</tr>
<tr>
<td>SetPenState</td>
<td>Sets the pen state in the GrafPort to specified values</td>
</tr>
<tr>
<td>GetPenState</td>
<td>Returns the pen state from the GrafPort to a specified location</td>
</tr>
<tr>
<td>SetPenSize</td>
<td>Sets the current pen size to a specified pen size</td>
</tr>
<tr>
<td>GetPenSize</td>
<td>Returns the current pen size to a specified location</td>
</tr>
<tr>
<td>SetPenMode</td>
<td>Sets the current pen mode to a specified pen mode</td>
</tr>
<tr>
<td>GetPenMode</td>
<td>Returns the pen mode from the current GrafPort</td>
</tr>
<tr>
<td>SetPenPat</td>
<td>Sets the current pen pattern to a specified pen pattern</td>
</tr>
<tr>
<td>GetPenPat</td>
<td>Copies the current pen pattern from the current GrafPort to a specified location</td>
</tr>
<tr>
<td>SetSolidPenPat</td>
<td>Sets the pen pattern to a solid pattern using the specified color</td>
</tr>
<tr>
<td>SetPenMask</td>
<td>Sets the pen mask to a specified mask</td>
</tr>
<tr>
<td>GetPenMask</td>
<td>Returns the pen mask to a specified location</td>
</tr>
<tr>
<td>SetBackPat</td>
<td>Sets the background pattern to a specified pattern</td>
</tr>
<tr>
<td>GetBackPat</td>
<td>Copies the current background pen pattern from the current GrafPort to a specified location</td>
</tr>
<tr>
<td>SetSolidBackPat</td>
<td>Sets the background pattern to a solid pattern using a specified color</td>
</tr>
<tr>
<td>SolidPattern</td>
<td>Sets a specified pattern to a solid pattern using a specified color</td>
</tr>
<tr>
<td>PenNormal</td>
<td>Sets the pen state to the standard state; pen location is not changed</td>
</tr>
<tr>
<td>MoveTo</td>
<td>Moves the current pen location to a specified point</td>
</tr>
<tr>
<td>Move</td>
<td>Moves the current pen location by specified horizontal and vertical displacements</td>
</tr>
</tbody>
</table>

A preview of the QuickDraw II routines
Table 16-1 (continued)
QuickDraw II routines and their functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Font routines</strong></td>
<td></td>
</tr>
<tr>
<td>SetFont</td>
<td>Sets the current font to a specified font</td>
</tr>
<tr>
<td>GetFont</td>
<td>Returns a handle to the current font</td>
</tr>
<tr>
<td>GetRomFont</td>
<td>Fills a specified buffer with information about the font in ROM</td>
</tr>
<tr>
<td>SetFontID</td>
<td>Sets the fontID field in the GrafPort</td>
</tr>
<tr>
<td>GetFontID</td>
<td>Returns the fontID field of the GrafPort</td>
</tr>
<tr>
<td>GetFontInfo</td>
<td>Returns information about the current font in a specified record</td>
</tr>
<tr>
<td>GetFSize</td>
<td>Returns the size of the font globals record</td>
</tr>
<tr>
<td>GetFontGlobals</td>
<td>Returns information about the current font in a specified record</td>
</tr>
<tr>
<td>SetFontFlags</td>
<td>Sets the font flags word to a specified value</td>
</tr>
<tr>
<td>GetFontFlags</td>
<td>Returns the current font flags word</td>
</tr>
<tr>
<td>SetTextFace</td>
<td>Sets the text face to a specified value</td>
</tr>
<tr>
<td>GetTextFace</td>
<td>Returns the current text face</td>
</tr>
<tr>
<td>SetTextMode</td>
<td>Sets the text mode to a specified value</td>
</tr>
<tr>
<td>GetTextMode</td>
<td>Returns the current text mode</td>
</tr>
<tr>
<td>SetSpaceExtra</td>
<td>Sets the spExtra field in the GrafPort to a specified value</td>
</tr>
<tr>
<td>GetSpaceExtra</td>
<td>Returns the value of the spExtra field from the GrafPort</td>
</tr>
<tr>
<td>SetCharExtra</td>
<td>Sets the chExtra field in the GrafPort to a specified value</td>
</tr>
<tr>
<td>GetCharExtra</td>
<td>Returns the chExtra field from the GrafPort</td>
</tr>
<tr>
<td>SetForeColor</td>
<td>Sets the fgColor field (foreground color) in the GrafPort to a specified value</td>
</tr>
<tr>
<td>GetForeColor</td>
<td>Returns the value of the current fgColor field (foreground color) from the GrafPort</td>
</tr>
<tr>
<td>SetBackColor</td>
<td>Sets the bgColor field (background color) in the GrafPort to a specified value</td>
</tr>
<tr>
<td>GetBackColor</td>
<td>Returns the value of the current bgColor field (background color) from the GrafPort</td>
</tr>
<tr>
<td><strong>Miscellaneous GrafPort routines</strong></td>
<td></td>
</tr>
<tr>
<td>SetClipHandle</td>
<td>Sets the clipRgn handle field in the GrafPort to a specified value</td>
</tr>
<tr>
<td>GetClipHandle</td>
<td>Returns a copy of the handle to the clipping region</td>
</tr>
<tr>
<td>SetVisRgn</td>
<td>Copies a specified region into the visible region (but does not change the visRgn field of the GrafPort)</td>
</tr>
<tr>
<td>GetVisRgn</td>
<td>Copies the contents of the visible region into a specified region</td>
</tr>
<tr>
<td>SetVisHandle</td>
<td>Sets the visRgn field in the GrafPort to a specified value</td>
</tr>
<tr>
<td>GetVisHandle</td>
<td>Returns a copy of the handle to the visible region</td>
</tr>
<tr>
<td>SetPicSave</td>
<td>Sets the picSave field in the GrafPort to a specified value</td>
</tr>
<tr>
<td>GetPicSave</td>
<td>Returns the value of the picSave field of the GrafPort</td>
</tr>
<tr>
<td>SetRgnSave</td>
<td>Sets the rgnSave field in the GrafPort to a specified value</td>
</tr>
<tr>
<td>GetRgnSave</td>
<td>Returns the value of the rgnSave field of the GrafPort</td>
</tr>
<tr>
<td>SetPolySave</td>
<td>Sets the polysave field in the GrafPort to a specified value</td>
</tr>
<tr>
<td>GetPolySave</td>
<td>Returns the value of the polysave field of the GrafPort</td>
</tr>
<tr>
<td>SetGrafProcs</td>
<td>Sets the grafProcs field of the current GrafPort to a specified value</td>
</tr>
<tr>
<td>GetGrafProcs</td>
<td>Returns the pointer to the grafProcs record associated with the GrafPort</td>
</tr>
<tr>
<td>SetUserField</td>
<td>Sets the userField field in the GrafPort to a specified value</td>
</tr>
<tr>
<td>GetUserField</td>
<td>Returns the value of the userField field of the GrafPort</td>
</tr>
<tr>
<td>SetSysField</td>
<td>Sets the sysField field in the GrafPort to a specified value—must not be called by an application</td>
</tr>
<tr>
<td>GetSysField</td>
<td>Returns the value of the sysField field of the GrafPort</td>
</tr>
</tbody>
</table>
Table 16-1 (continued)
QuickDraw II routines and their functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Line drawing routines</strong></td>
<td></td>
</tr>
<tr>
<td>LineTo</td>
<td>Draws a line from the current pen location to a specified point</td>
</tr>
<tr>
<td>Line</td>
<td>Draws a line from the current pen location to a new point specified by the</td>
</tr>
<tr>
<td></td>
<td>horizontal and vertical displacements</td>
</tr>
<tr>
<td><strong>Rectangle drawing routines</strong></td>
<td></td>
</tr>
<tr>
<td>FrameRect</td>
<td>Draws the frame of a specified rectangle using the current pen mode, pen</td>
</tr>
<tr>
<td></td>
<td>pattern, and pen size</td>
</tr>
<tr>
<td>PaintRect</td>
<td>Paints the interior of a specified rectangle using the current pen mode and</td>
</tr>
<tr>
<td></td>
<td>pen pattern</td>
</tr>
<tr>
<td>EraseRect</td>
<td>Erases the interior of a specified rectangle by filling it with the</td>
</tr>
<tr>
<td></td>
<td>background pattern</td>
</tr>
<tr>
<td>InvertRect</td>
<td>Inverts the pixels in the interior of a specified rectangle</td>
</tr>
<tr>
<td>FillRect</td>
<td>Fills the interior of a specified rectangle with a specified pen pattern</td>
</tr>
<tr>
<td><strong>Round rectangle drawing routines</strong></td>
<td></td>
</tr>
<tr>
<td>FrameRRect</td>
<td>Draws the frame of a specified round rectangle using the current pen mode,</td>
</tr>
<tr>
<td></td>
<td>pen pattern, and pen size</td>
</tr>
<tr>
<td>PaintRRect</td>
<td>Paints the interior of a specified round rectangle using the current pen</td>
</tr>
<tr>
<td></td>
<td>mode and pen pattern</td>
</tr>
<tr>
<td>EraseRRect</td>
<td>Erases the interior of a specified round rectangle by filling it with the</td>
</tr>
<tr>
<td></td>
<td>background pattern</td>
</tr>
<tr>
<td>InvertRRect</td>
<td>Inverts the pixels in the interior of a specified round rectangle</td>
</tr>
<tr>
<td>FillRRect</td>
<td>Fills the interior of a specified round rectangle with a specified pen</td>
</tr>
<tr>
<td><strong>Region drawing routines</strong></td>
<td></td>
</tr>
<tr>
<td>FrameRgn</td>
<td>Draws the frame of a specified region using the current pen mode, pen</td>
</tr>
<tr>
<td></td>
<td>pattern, and pen size</td>
</tr>
<tr>
<td>PaintRgn</td>
<td>Paints the interior of a specified region using the current pen mode and</td>
</tr>
<tr>
<td></td>
<td>pen pattern</td>
</tr>
<tr>
<td>EraseRgn</td>
<td>Erases the interior of a specified region by filling it with the background</td>
</tr>
<tr>
<td></td>
<td>pattern</td>
</tr>
<tr>
<td>InvertRgn</td>
<td>Inverts the pixels in the interior of a specified region</td>
</tr>
<tr>
<td>FillRgn</td>
<td>Fills the interior of a specified region with a specified pen pattern</td>
</tr>
<tr>
<td><strong>Polygon drawing routines</strong></td>
<td></td>
</tr>
<tr>
<td>FramePoly</td>
<td>Draws the frame of a specified polygon using the current pen mode, pen</td>
</tr>
<tr>
<td></td>
<td>pattern, and pen size</td>
</tr>
<tr>
<td>PaintPoly</td>
<td>Paints the interior of a specified polygon using the current pen mode and</td>
</tr>
<tr>
<td></td>
<td>pen pattern</td>
</tr>
<tr>
<td>ErasePoly</td>
<td>Erases the interior of a specified polygon by filling it with the background</td>
</tr>
<tr>
<td></td>
<td>pattern</td>
</tr>
<tr>
<td>InvertPoly</td>
<td>Inverts the pixels in the interior of a specified polygon</td>
</tr>
<tr>
<td>FillPoly</td>
<td>Fills the interior of a specified polygon with a specified pen pattern</td>
</tr>
</tbody>
</table>

A preview of the QuickDraw II routines 16-5
### Table 16-1 (continued)
QuickDraw II routines and their functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Oval drawing routines</strong></td>
<td></td>
</tr>
<tr>
<td>FrameOval</td>
<td>Draws the frame of a specified oval using the current pen mode, pen pattern, and pen size</td>
</tr>
<tr>
<td>PaintOval</td>
<td>Paints the interior of a specified oval using the current pen mode and pen pattern</td>
</tr>
<tr>
<td>EraseOval</td>
<td>Erases the interior of a specified oval by filling it with the background pattern</td>
</tr>
<tr>
<td>InvertOval</td>
<td>Inverts the pixels in the interior of a specified oval</td>
</tr>
<tr>
<td>FillOval</td>
<td>Fills the interior of a specified oval with a specified pen pattern</td>
</tr>
<tr>
<td><strong>Arc drawing routines</strong></td>
<td></td>
</tr>
<tr>
<td>FrameArc</td>
<td>Draws the frame of a specified arc using the current pen mode, pen pattern, and pen size</td>
</tr>
<tr>
<td>PaintArc</td>
<td>Paints the interior of a specified arc using the current pen mode and pen pattern</td>
</tr>
<tr>
<td>EraseArc</td>
<td>Erases the interior of a specified arc by filling it with the background pattern</td>
</tr>
<tr>
<td>InvertArc</td>
<td>Inverts the pixels in the interior of a specified arc</td>
</tr>
<tr>
<td>FillArc</td>
<td>Fills the interior of a specified arc with a specified pen pattern</td>
</tr>
<tr>
<td><strong>Pixel transfer routines</strong></td>
<td></td>
</tr>
<tr>
<td>ScrollRect</td>
<td>Shifts the pixels inside the intersection of a specified rectangle, visible region, clipping region, port rectangle, and bounds rectangle</td>
</tr>
<tr>
<td>PaintPixels</td>
<td>Transfers a region of pixels</td>
</tr>
<tr>
<td>PPToPort</td>
<td>Transfers pixels from a source pixel map to the current port and clips the pixels to the current visible region and clipping region</td>
</tr>
<tr>
<td><strong>Text drawing and measuring routines</strong></td>
<td></td>
</tr>
<tr>
<td>DrawChar</td>
<td>Draws a specified character at the current pen location and updates the pen location</td>
</tr>
<tr>
<td>DrawText</td>
<td>Draws specified text at the current pen location and updates the pen location</td>
</tr>
<tr>
<td>DrawString</td>
<td>Draws a specified Pascal-type string at the current pen location and updates the pen location</td>
</tr>
<tr>
<td>DrawCString</td>
<td>Draws a specified C string at the current pen location and updates the pen location</td>
</tr>
<tr>
<td>CharWidth</td>
<td>Returns the character width, in pixels (pen displacement), of a specified character</td>
</tr>
<tr>
<td>TextWidth</td>
<td>Returns the character width, in pixels (pen displacement), of specified text</td>
</tr>
<tr>
<td>StringWidth</td>
<td>Returns the sum of all character widths, in pixels (pen displacements), of a specified Pascal-type string</td>
</tr>
<tr>
<td>CStringWidth</td>
<td>Returns the sum of all the character widths, in pixels (pen displacements), of a specified C string</td>
</tr>
<tr>
<td>CharBounds</td>
<td>Puts the character bounds rectangle of a specified character into a specified buffer</td>
</tr>
<tr>
<td>TextBounds</td>
<td>Puts the character bounds rectangle of specified text into a specified buffer</td>
</tr>
<tr>
<td>StringBounds</td>
<td>Puts the character bounds rectangle of a specified Pascal-type string into a specified buffer</td>
</tr>
<tr>
<td>CStringBounds</td>
<td>Puts the character bounds rectangle of a specified C string into a specified buffer</td>
</tr>
<tr>
<td>Routine</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Calculations with rectangles</strong></td>
<td></td>
</tr>
<tr>
<td>SetRect</td>
<td>Sets a specified rectangle to specified values</td>
</tr>
<tr>
<td>OffsetRect</td>
<td>Offsets a specified rectangle by specified displacements</td>
</tr>
<tr>
<td>InsetRect</td>
<td>Insets a specified rectangle by specified displacements</td>
</tr>
<tr>
<td>SectRect</td>
<td>Calculates the intersection of two rectangles and places the intersection in a destination rectangle</td>
</tr>
<tr>
<td>UnionRect</td>
<td>Calculates the smallest rectangle that contains both source rectangles and places the result in a destination rectangle</td>
</tr>
<tr>
<td>PtInRect</td>
<td>Detects whether the pixel below and to the right of a specified point is in a specified rectangle</td>
</tr>
<tr>
<td>Pt2Rect</td>
<td>Copies a specified point to the upper left corner of a specified rectangle and another point to the lower right corner of the rectangle</td>
</tr>
<tr>
<td>EqualRect</td>
<td>Indicates whether two rectangles are equal</td>
</tr>
<tr>
<td>NotEmptyRect</td>
<td>Indicates whether a specified rectangle is not empty</td>
</tr>
<tr>
<td><strong>Calculations with points</strong></td>
<td></td>
</tr>
<tr>
<td>AddPt</td>
<td>Adds two specified points together and leaves the result in the destination point</td>
</tr>
<tr>
<td>SubPt</td>
<td>Subtracts the source point from the destination point and leaves the result in the destination point</td>
</tr>
<tr>
<td>SetPt</td>
<td>Sets a point to specified horizontal and vertical values</td>
</tr>
<tr>
<td>EqualPt</td>
<td>Indicates whether two points are equal</td>
</tr>
<tr>
<td>LocalToGlobal</td>
<td>Converts a point from local coordinates to global coordinates</td>
</tr>
<tr>
<td>GlobalToLocal</td>
<td>Converts a point from global coordinates to local coordinates</td>
</tr>
<tr>
<td><strong>Calculations with regions</strong></td>
<td></td>
</tr>
<tr>
<td>NewRgn</td>
<td>Allocates space for a new region and initializes it to an empty region—this is the only way to create a new region</td>
</tr>
<tr>
<td>DisposeRgn</td>
<td>Deallocates the memory for a specified region</td>
</tr>
<tr>
<td>CopyRgn</td>
<td>Copies the region definition from one region to another</td>
</tr>
<tr>
<td>SetEmptyRgn</td>
<td>Destroys previous region information by setting a specified region to an empty region</td>
</tr>
<tr>
<td>SetRectRgn</td>
<td>Destroys previous region information by setting a specified region to a specified rectangle</td>
</tr>
<tr>
<td>RectRgn</td>
<td>Destroys previous region information by setting a specified region to a specified rectangle</td>
</tr>
<tr>
<td>OpenRgn</td>
<td>Allocates temporary space and starts saving lines and framed shapes for later processing as a region definition</td>
</tr>
<tr>
<td>CloseRgn</td>
<td>Completes the region definition process started by an OpenRgn call</td>
</tr>
<tr>
<td>OffsetRgn</td>
<td>Moves a region on the coordinate plane a specified distance</td>
</tr>
<tr>
<td>InsetRgn</td>
<td>Shrinks or expands a specified region</td>
</tr>
<tr>
<td>SectRgn</td>
<td>Calculates the intersection of two regions and places the intersection in a destination region</td>
</tr>
</tbody>
</table>
### Table 16-1 (continued)
QuickDraw II routines and their functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Calculations with regions</strong></td>
<td></td>
</tr>
<tr>
<td>UnionRgn</td>
<td>Calculates the smallest region that contains every point that is in either source region and places the result in a destination region</td>
</tr>
<tr>
<td>DiffRgn</td>
<td>Calculates the difference of two regions and places the difference in a destination region</td>
</tr>
<tr>
<td>XorRgn</td>
<td>Calculates the difference between the union and the intersection of two regions and places the result in a destination region</td>
</tr>
<tr>
<td>PtInRgn</td>
<td>Checks to see whether the pixel below and to the right of a specified point is within a specified region</td>
</tr>
<tr>
<td>RectInRgn</td>
<td>Checks whether a specified rectangle intersects a specified region</td>
</tr>
<tr>
<td>EqualRgn</td>
<td>Indicates whether two regions are equal</td>
</tr>
<tr>
<td>EmptyRgn</td>
<td>Indicates whether a specified region is empty</td>
</tr>
<tr>
<td><strong>Calculations with polygons</strong></td>
<td></td>
</tr>
<tr>
<td>OpenPoly</td>
<td>Returns a handle to a polygon data structure that will be updated by future LineTo calls</td>
</tr>
<tr>
<td>ClosePoly</td>
<td>Completes the polygon definition process started with an OpenPoly call</td>
</tr>
<tr>
<td>KillPoly</td>
<td>Disposes of a specified polygon</td>
</tr>
<tr>
<td>OffsetPoly</td>
<td>Offsets a specified polygon by specified horizontal and vertical displacements</td>
</tr>
<tr>
<td><strong>Mapping and scaling utilities</strong></td>
<td></td>
</tr>
<tr>
<td>MapPt</td>
<td>Maps a specified point from a source rectangle to a destination rectangle</td>
</tr>
<tr>
<td>MapRect</td>
<td>Maps a specified rectangle from a source rectangle to a destination rectangle</td>
</tr>
<tr>
<td>MapRgn</td>
<td>Maps a specified region from a source rectangle to a destination rectangle</td>
</tr>
<tr>
<td>MapPoly</td>
<td>Maps a specified polygon from a source rectangle to a destination rectangle</td>
</tr>
<tr>
<td>ScalePt</td>
<td>Scales a specified point from a source rectangle to a destination rectangle</td>
</tr>
<tr>
<td><strong>Cursor-handling routines</strong></td>
<td></td>
</tr>
<tr>
<td>SetCursor</td>
<td>Sets the cursor to an image passed in a specified cursor record</td>
</tr>
<tr>
<td>GetCursorAdr</td>
<td>Returns a pointer to the current cursor record</td>
</tr>
<tr>
<td>HideCursor</td>
<td>Hides the cursor by decrementing the cursor level</td>
</tr>
<tr>
<td>ShowCursor</td>
<td>Shows the cursor by incrementing the cursor level</td>
</tr>
<tr>
<td>ObscureCursor</td>
<td>Hides the cursor until the mouse moves</td>
</tr>
<tr>
<td>InitCursor</td>
<td>Reinitializes the cursor</td>
</tr>
<tr>
<td><strong>Miscellaneous QuickDraw II utilities</strong></td>
<td></td>
</tr>
<tr>
<td>Random</td>
<td>Returns a pseudorandom number in the range -32768 to 32767</td>
</tr>
<tr>
<td>SetRandSeed</td>
<td>Sets the seed value for the random number generator</td>
</tr>
<tr>
<td>GetPixel</td>
<td>Returns the pixel below and to the right of a specified point</td>
</tr>
<tr>
<td>GetAddress</td>
<td>Returns a pointer to a specified table</td>
</tr>
<tr>
<td>SetIntUse</td>
<td>Indicates to the cursor drawing code whether the code should use scan line interrupts</td>
</tr>
<tr>
<td>SetStdProcs</td>
<td>Sets up a specified record of pointers for customizing QuickDraw II operations</td>
</tr>
</tbody>
</table>
QuickDraw II can draw to the screen or to other parts of Apple II GS memory. In fact, printing a document with the Print Manager involves using QuickDraw II to "draw" your document into a memory buffer used by the Print Manager (see Chapter 15, "Print Manager," in Volume 1).

To get our bearings, we'll first consider where QuickDraw II draws. Then we'll briefly discuss how it draws, and finally look at what it draws.

Where QuickDraw II draws

The question of where QuickDraw II draws involves consideration of Apple II GS memory (including screen memory) as well as QuickDraw II's own internal representation of its drawing universe. These are the main concepts for you to remember:

- Drawings are stored in Apple II GS memory as pixel images, ordered collections of bytes that represent rectangular arrays of pixels. Screen memory contains a special pixel image—its contents are displayed on the computer's monitor.

- QuickDraw II draws its text and graphic objects on an abstract two-dimensional mathematical surface called the coordinate plane. Points on a plane are much easier to visualize and manipulate than addresses in memory. Locations on the QuickDraw II coordinate plane are related to pixel-image locations by specific location information supplied to QuickDraw II.

- QuickDraw II draws most objects within the context of graphic ports. A port is a complete drawing environment and defines, among other things, a specific part of memory and a specific rectangular area on the coordinate plane where drawing can occur. There can be many ports open at a time—some for drawing to the screen, some for drawing to other parts of memory. Different ports' drawing spaces may be separate from each other, or they may overlap.

- QuickDraw II can be made to clip, or constrain its drawing, to within limits of arbitrary size, shape, and location.

- By manipulating two independent sets of coordinates (global coordinates and local coordinates), an application can easily control both what gets drawn inside a port's drawing space and where on the screen or other pixel image that drawing space appears.
QuickDraw II locates every action it takes in terms of coordinates on a two-dimensional grid, as shown in Figure 16-1. The grid is QuickDraw II’s **coordinate plane**; coordinates on the plane are integers ranging from −16K to +16K in both the X and Y directions. The point (0,0), therefore, is in the middle of the grid. Note also that grid values increase to the right and **downward** on the plane; this is different from what you might be used to, but it is the direction and order in which video scan lines are drawn.

Units on the grid are in terms of **pixels**. Thus a 10 x 10 square on the coordinate plane is equivalent to a rectangle 10 pixels by 10 pixels on the display screen (this would not be a square, of course, because Apple IIGS pixels are not square). Only a very small portion of the coordinate plane can be displayed on the screen at any one time—the plane is 32,000 pixels on a side, and the screen can show a maximum of 640 pixels by 200 pixels at a time. The small black square near the center of the grid in Figure 16-1 shows the approximate size of the screen (in 320 mode) compared with the coordinate plane.

---

**Warning**

QuickDraw II must not be asked to draw outside the coordinate plane. Commands to draw outside this space will produce unpredictable results. They won’t generate errors.

*Macintosh programmers:* This conceptual drawing space is not the same as that for QuickDraw on the Macintosh. On the Macintosh, the drawing space is 64K by 64K pixels centered around 0,0, which makes the boundary coordinates −32K,−32K and 32K,32K.
To understand how QuickDraw II does its drawing, we need to consider how it represents some basic graphic elements. On the coordinate plane, grid lines are considered to be infinitely thin. A point is defined as the intersection of two grid lines, so it also has no dimensions. Pixels, on the other hand, have a definite size; they are thought of as falling between the lines of the grid. The smallest element QuickDraw II can draw is a pixel, so if it were to draw a point at the location (3,3) on the coordinate plane, it would have to draw a single pixel. But which one? The point is between four pixels.
QuickDraw defines the pixel corresponding to each point on the plane as the pixel immediately *below and to the right* of the point, as shown in Figure 16-2.

![Figure 16-2](image)

**Grid lines, a point, and a pixel on the coordinate plane**

**Pixel images and the coordinate plane**

A **pixel image** is an area of memory that contains a graphic image. The image is organized as a rectangular grid of pixels occupying contiguous memory locations. Each pixel has a value that determines what color in the graphic image is associated with that pixel.

*Macintosh programmers*: QuickDraw II's pixel images are similar to Macintosh QuickDraw's bit images. The major difference is that a pixel is described by more than a single bit.

QuickDraw II draws to the coordinate plane. However, the coordinate plane is really just an abstract concept. Inside the Apple II GS, drawing actually occurs by modifying pixel images—that is, by modifying the contents of certain memory locations. In particular, drawing something visible on the screen involves modifying the pixel image corresponding to screen memory.
The data structure that ties the coordinate plane to memory is the *locInfo* (for location information) record. The *locInfo* record tells QuickDraw II where in memory to draw, how the pixel image in that part of memory is arranged, and what that image's position on the coordinate plane is. The structure of the *locInfo* record definition is shown in Figure 16-3, and each field is described in more detail following the figure.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td><em>portSCB</em></td>
<td>Word—Replica of scan line control byte</td>
</tr>
<tr>
<td>1</td>
<td><em>ptrToPixImage</em></td>
<td><em>Long—POINTER to first byte of pixel image</em></td>
</tr>
<tr>
<td>2</td>
<td><em>width</em></td>
<td><em>Word—INTEGER; width, in bytes, of each line in pixel image</em> (must be even multiple of 8)</td>
</tr>
<tr>
<td>6</td>
<td><em>boundsRect</em></td>
<td><em>Four words—RECT data structure defining boundary rectangle</em></td>
</tr>
</tbody>
</table>

**Figure 16-3**
The *locInfo* record

The fields are as follows:

- The *portSCB* (a replica of the scan line control byte) tells QuickDraw II how many bits per pixel there are in this image—two for 640 mode, four for 320 mode. The scan line control byte and the differences between 640 mode and 320 mode are discussed further in the section “Drawing in Color” in this chapter.

- The *ptrToPixImage* field contains the image pointer; that is, the memory address of the image. It points to the first byte of the pixel image, which contains the first (upper leftmost) pixel.

- The *width* field specifies the image width; that is, the width (in bytes, not pixels) of each line in the pixel image. QuickDraw II needs to know this so it can tell where each new row in the image starts. (The image width must be an even multiple of eight bytes.)

- The *boundsRect* (for boundary rectangle) is a RECT data structure defining the rectangle that maps the pixel image onto the coordinate plane. The top left point in the rectangle corresponds to the first pixel in the image. The bottom right of the rectangle describes the extent of the pixel image (as far as QuickDraw II is concerned).
The image pointer, the image width, and the boundary rectangle defined by the `boundsRect` are shown in Figure 16-4.

![Figure 16-4](image)

**Figure 16-4**
Pixel image and boundary rectangle

- *Note:* Remember that what separates one pixel image from another is where in memory it is stored, not where on the QuickDraw II coordinate plane its boundary rectangle happens to be. You can think of each pixel image as having its own private copy of the entire coordinate plane to play with, so even if two pixel images have overlapping coordinate plane locations, there won’t be any conflict between them if they occupy completely different parts of computer memory.

**GrafPort, port rectangle, and clipping**

Most drawing takes place in conjunction with a data structure called a **GrafPort** (for **graphic port**). Each GrafPort contains a complete specification of a drawing environment, including the location information (`locInfo` record) described earlier. In addition to the location information, a GrafPort contains three other fields that restrict where drawing in a pixel image can take place: the port rectangle, clipping region, and visible region.

The **port rectangle** (as specified by the `portRect` field in the GrafPort) is a rectangle on the coordinate plane. Any drawing through a GrafPort occurs only inside its port rectangle. When you look at a window on the screen in a desktop application, its interior (everything but its frame) corresponds to a port rectangle. Windows are described further in Chapter 25, “Window Manager.”
The port rectangle can coincide with the boundary rectangle, or it can be different. You can think of it as a movable opening that allows access to all or part of the pixel image. As Figure 16-5 shows, QuickDraw II can draw only where the boundary rectangle and port rectangle overlap.

![Figure 16-5](image)

**Boundary rectangle/port rectangle intersection**

The clipping region (as specified by the *clipRgn* field in the GrafPort) is provided for an application to use. When a GrafPort is opened or initialized, the clipping region is set to the entire coordinate plane (effectively preventing any clipping from occurring). The program can use the clipping region any way it wants. Drawing to a pixel image through a GrafPort occurs only inside the clipping region.

The visible region (as specified by the *visRgn* field in the GrafPort) is normally maintained by the Window Manager. An application can have multiple windows on the screen, each one associated with a GrafPort. Windows can overlap, and each port's visible region represents the parts of the window that are visible.

*Note:* When using GrafPorts that are not associated with windows, it is up to your application to maintain the visible region. When a GrafPort is opened or initialized, its *visRgn* field is set to be equal to the *portRect* field. It will stay that way until your program changes it.

In summary, drawing occurs in a pixel image only in the intersection of the boundary rectangle, port rectangle, clipping region, and visible region.
Global and local coordinate systems

In QuickDraw II's universe, everything is positioned in terms of coordinates on the plane. However, if you think of multiple open windows on the screen, you can see that there are at least two different ways in which you might want to locate objects:

- You may want to specify where windows appear on the screen (when they are moved, for example).
- You may want to specify where objects appear within windows (when scrolling, for example) independently of where on the screen the windows may be.

The toolbox needs global coordinates whenever more than one GrafPort share the same pixel image; the global coordinates tell QuickDraw II exactly where every port rectangle is compared with every other one. The global coordinate system for each GrafPort is that in which the boundary rectangle for its pixel image has its origin at \((0,0)\) on the coordinate plane. In QuickDraw II, the origin of a rectangle is its upper left corner. For drawing to the screen, you can think of global coordinates as screen coordinates, where the top left corner of the screen is the point \((0,0)\).

However, each port also has its own local coordinate system. For example, when drawing into a port, you can think in terms of distance from the port rectangle's origin rather than the boundary rectangle's origin. By defining the port rectangle as starting at \((0,0)\), you can base all your drawing commands on distance in from the left edge and down from the top of the port rectangle.

That's convenient for drawing in a window, but local coordinates offer more convenience than that. They aren't constrained to a value of \((0,0)\) for the port rectangle origin—you can set them to any coordinate-plane value. Why would you want to? Because of the way drawing commands work.

Suppose you are using a window to display portions of a document that is larger than the port rectangle in size—a fairly common occurrence. You are using drawing commands that draw the entire document, and you know that's no problem because the drawing will be automatically clipped to the port rectangle. But how do you control which part of the document shows through in your window? You do it by adjusting local coordinates.
For example, consider a document that has (0,0) as its origin. All QuickDraw II's drawing commands are based on the current port's local coordinate system. So if location (0,0) in your GrafPort's local coordinates correspond to the port rectangle's upper left corner, any time you draw your document into that port, its upper left corner will be displayed. If you define your local coordinates differently, different parts of your document will appear in the window. Thus, you can think of local coordinates as document coordinates—the upper left corner of the document that the port displays is the local coordinate origin—as shown in Figure 16-6.

![Figure 16-6](image)

**Figure 16-6**
Drawing different parts of a document by changing local coordinates

*Note:* When the local coordinates of a GrafPort are changed, the coordinates of the GrafPort's boundary rectangle and visible region are similarly recalculated, so the port will not change its relative position on the screen or its relation to other open ports on the screen.

However, when the local coordinates are changed, the GrafPort's clipping region and pen location are not changed—that is, they would appear to shift right along with the image that is being viewed in the port. It makes sense to have the pen (which is used to modify the image being viewed) and the clipping region (which is used to mask off parts of the image being viewed) stay with the image.
How QuickDraw II draws

The way QuickDraw II draws any of its objects depends on the drawing environment specified in the current GrafPort. Each GrafPort record includes location and clipping information (described earlier), information about the graphics pen (described next), information about any text that will be drawn (described in the section “Drawing Text” in this chapter), and other information, such as the patterns to draw with.

Drawing pen

Each open port has its own drawing pen. The pen controls where and how drawing (of both text and graphics) occurs. It has several characteristics (which can be set by the application) that control this.

**Pen location:** The pen has a coordinate-plane location (in local coordinates). The pen location is used only for drawing lines and text—other shapes are drawn independently of pen location.

**Pen size:** The pen is a rectangle that can have almost any width or height. Its default size is 1 x 1 (pixels). If either the width or the height is set to 0, the pen will not draw.

**Pen pattern:** The pen pattern is a repeating array (eight pixels by eight pixels) that is used like ink in the pen. Wherever the pen draws, the pen pattern is drawn in the image. The pattern is always aligned with the coordinate plane so that adjacent areas of the same pattern drawn at different times will blend in a continuous manner.

**Background pattern:** The background pattern is an array similar to the pen pattern. The process of erasing is that of drawing with the background pattern.

**Drawing mask:** The drawing mask is an eight-bit by eight-bit pattern that is used to mask, or screen off, parts of the pattern as it is drawn. Only those pixels in the pattern aligned with an on (1) bit in the mask are drawn. Figure 16-7 shows how a mask affects drawing with a pattern.
Note that drawing with a mask in which every bit has the value 1 is like drawing with no mask at all—all pen pixels are passed through to the image. Likewise, drawing with a mask that is all 0's is like not drawing at all—all pen pixels are blocked.

**Pen mode:** The pen mode specifies one of eight Boolean operations (modeCopy, notCopy, modeOr, notOr, modeXOR, notXOR, modeBIC, and notBIC) that determine how the pen pattern is to affect an existing image. When the pen draws, QuickDraw II compares pixels in the existing image with their corresponding pixels in the pattern and then uses the pen mode to determine the value of the resulting pixels. For example, with a pen mode of modeCopy, the existing pixels' values are ignored—a solid black line is black regardless of the image already on the plane. With a pen mode of notXOR, the bits in each pen pixel are inverted, then combined in an exclusive-OR operation with the bits in each corresponding existing pixel. Figure 16-8 shows a filled rectangle drawn over an existing circle, in both modeCopy and notXOR mode.

**Figure 16-7**
Drawing with pattern and mask

**Figure 16-8**
How pen mode affects drawing
The uses for the different pen modes are shown in Table 16-2. For more detail on how to use the modes, see the section "SetPenMode" in this chapter.

**Table 16-2**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modeCopy, notCopy</td>
<td>Copy source (or NOT source) to destination. The <strong>modeCopy</strong> mode is the typical drawing mode. For text, the fully colored text pixels (both foreground and background) are copied into the destination.</td>
</tr>
<tr>
<td>modeOR, notOR</td>
<td>Overlay (OR) source (or NOT source) and destination. Use <strong>modeOR</strong> to nondestructively overlay new images on top of existing images and <strong>notOR</strong> to overlay inverted images. For text, the fully colored text pixels (both foreground and background) are ORed with the destination.</td>
</tr>
<tr>
<td>modeXOR, notXOR</td>
<td>Exclusive-or (XOR) pen with destination. Use these modes for cursor drawing and rubber-banding. If an image is drawn using <strong>modeXOR</strong>, the appearance of the destination at the image location can be restored merely by drawing the image again in <strong>modeXOR</strong>. For text, the fully colored text pixels (both foreground and background) are XORed with the destination.</td>
</tr>
<tr>
<td>modeBIC, notBIC</td>
<td>Bit Clear (BIC) pen with destination ((NOT pen) AND destination). Use this mode to explicitly erase (turn off) pixels, often prior to overlaying another image. You can use <strong>notBIC</strong> to display the intersection of two images. For text, the fully colored text pixels (both foreground and background) are BICed with the destination.</td>
</tr>
</tbody>
</table>

**Basic drawing functions**

QuickDraw II draws lines using the current pen size, pen pattern, drawing mask, and pen mode. It draws other shapes (rectangles, rounded-corner rectangles, ovals, arcs, polygons, and regions) in five different ways:

- **Framing** uses the current pen size, pen pattern, drawing mask and pen mode to draw an outline of the shape.
- **Painting** uses the current pen pattern, drawing mask, and pen mode to fill the interior of the shape.
- **Erasing** uses the current background pattern and drawing mask to fill the interior of the shape.
- **Inverting** uses the drawing mask to invert the pixels in the interior of the shape.
- **Filling** uses a specified pattern and the drawing mask to fill the interior of the shape.

QuickDraw II draws text as described in the section "Drawing Text" in this chapter.
**What QuickDraw II draws**

QuickDraw II can draw a number of graphic objects into a pixel image. It draws text characters in a variety of monospaced and proportional fonts, with styling variations that include italics, boldface, underlining, outlining, and shadowing. It draws straight lines of any length, width, and pattern. It draws hollow or pattern-filled rectangles, circles, and polygons. It draws elliptical arcs and filled wedges, irregular shapes, and collections of shapes. It also draws pictures—combinations of these simple shapes. Figure 16-9 summarizes QuickDraw II’s graphic objects.

![Figure 16-9](image)

**Points and lines**

A point is represented mathematically by its Y and X coordinates—these are two integers.

**Important**

QuickDraw II’s data structure that defines a point has the vertical coordinate first: (y,x) rather than (x,y).
A line is represented by its ends—two points, or four integers. Like a point, a line is infinitely thin. When drawing a line, QuickDraw II moves the top left corner of the pen along the mathematical trajectory from the current pen location to the destination location. The pen hangs below and to the right of the trajectory, as illustrated in Figure 16-10.

Figure 16-10
Drawing a line

Before drawing a line, you can use QuickDraw II calls to set the current pen location and other characteristics, such as pen size, mode, and pattern.

Rectangles

A rectangle is also represented by two points—its upper left and lower right corners. The borders of a rectangle are infinitely thin. Rectangles are fundamental to QuickDraw II; there are many functions for moving, sizing, and otherwise manipulating rectangles.

The pixels associated with a rectangle are only those within the rectangle's bounding lines. Thus, the pixels immediately below and to the right of the bottom and right-hand lines of the rectangle are not part of it.

Important

QuickDraw II's RECT data structure has coordinates in the following order: top, left, bottom, right. Thus, the defining coordinates for the rectangle in Figure 16-11 are (1,2,7,6). This order may be different from what you are used to, but it is consistent with the (y,x) ordering of points.

16-22 Chapter 16: QuickDraw II
Rectangles may have square or rounded corners. The corners of **rounded-corner rectangles** (Figure 16-12) are sections of ovals (discussed next); they are specified by an oval height and an oval width.

**Circles, ovals, arcs, and wedges**

Ellipses and portions of ellipses form another class of shapes drawn by QuickDraw II. An **oval** (Figure 16-13) is an ellipse; it is defined in the same way as a rectangle, with the exception that QuickDraw II is told to draw the ellipse inscribed within the rectangle rather than the rectangle itself. If the enclosing rectangle is a square, the resulting oval is a circle.
• **Pixel shape:** Remember, Apple II GS pixels are not square. A perfect circle (or square) on the screen will have unequal horizontal and vertical dimensions in terms of pixels.

An **arc** (Figure 16-14) is a portion of an oval defined by the oval's enclosing rectangle and by two angles (the beginning and the end of the arc) and measured clockwise from vertical.

If an arc is painted, filled, inverted, or erased, it becomes a **wedge;** its fill pattern extends to the center of the enclosing rectangle, within the area defined by the beginning and ending angle lines.

![Figure 16-14](image)

**Arc**

**Polygons**

A **polygon** (Figure 16-15) is any sequence of connected lines. You define a polygon by moving the pen to the starting point of the polygon and drawing lines from there to the next point, from that point to the next, and so on.

![Figure 16-15](image)

**Polygon**
Polygons are not treated exactly the same as other closed shapes, such as rectangles. For example, when QuickDraw II draws a polygon, it draws outside the actual boundary of the polygon because the line-drawing routines draw below and to the right of the pen locations. When it paints, fills, inverts, or erases a polygon, however, the fill pattern stays within the boundary of the polygon. If the polygon's ending point isn't the same as its starting point, QuickDraw II adds a line between them to complete the shape.

Regions

A region is another fundamental element of QuickDraw II, one that can be considerably more complex than a line or a rectangle. A **region** (Figure 16-16) is defined as a collection of shapes or lines (or other regions) whose outline is one or more closed loops. Your application can draw, erase, move, or manipulate regions the way it does any other QuickDraw II shapes.

You can define regions by drawing lines, framing shapes, manipulating existing regions, and equating regions to rectangles or other regions.

Regions are particularly important to the Window Manager, which must keep track of often irregularly shaped, noncontiguous portions of windows in order to know when to activate the windows or what parts of them to update.

![Figure 16-16 Region](image)

Pictures

A **picture** is a collection of QuickDraw II drawing commands. Its data structure consists of little more than the stored commands. QuickDraw II plays the commands back when the picture is reconstructed with a DrawPicture call. A complex mechanical drawing produced from an Apple II GS drafting program might be saved as a single QuickDraw II picture.

Pictures can be used to transfer data between applications via the Clipboard. See Chapter 20, "Scrap Manager."
Drawing text

QuickDraw II doesn't draw only graphic images—it also does all text drawing for desktop applications. As an application writer, you can easily control the placement, size, style, font, and color of display text with QuickDraw II calls.

Your program can provide QuickDraw II with text in the following formats:

- **Character**: A single ASCII character at a time
- **Pascal string**: A length byte followed by a sequence of ASCII characters
- **C string**: A sequence of ASCII characters terminated by a 0 byte ($00)
- **Text block**: A number of ASCII characters in a buffer, with the number specified separately

QuickDraw II draws the text in the same format in which it receives it. It draws each character at the current pen location, with the current font, using the current text mode, with the current character style, and using the current foreground and background colors. After text is drawn, the pen position is updated.

Simple text manipulation

This section introduces the text concepts you will need to know about for most applications. For most applications, you won't need to know anything more about fonts than is presented in this section. If you're writing an application that lets the user choose from a selection of fonts, or if you're developing an application that requires a specific font, you'll also need to know about the Font Manager. See Chapter 8, "Font Manager," in Volume 1.

A font is a collection of graphical and numerical information representing a set of characters. The graphical part of the font is called the font strike and consists of all the images of the characters, placed one after another. (The font strike in a IIGS font is stored in a one-bit-per-pixel format.) By convention, no blank space is left between the character images in the font strike; when text is drawn, both the space left between characters and the positioning of characters are determined by several tables of numerical information that are also part of the font. For the precise format of a IIGS font, see the section "Font Definition" in this chapter.

QuickDraw II always displays and measures text using the current font (whose handle is found in the fontHandle field of the current GrafPort).

A font has a base line (a horizontal line that runs through the font strike), an ascent (the number of rows of pixels of the font above the base line), and a descent (the number of rows below). Each character in a font has a character image (the piece of the font strike that represents the character, using a bit set to 1 to represent the character's foreground pixels), a character origin (a point on the base line used to position the character with respect to the current pen position), and a character width (the number of pixels QuickDraw II will advance the pen position after it draws the character).
These concepts are illustrated in Figure 16-17.

When a character is drawn, it's placed so its character origin coincides with the current pen position. The character image's 1 bits, as mentioned, determine the **foreground pixels**. (The 0 bits of the character image are background pixels, but they are not the character's only background pixels. The precise definition of background pixels is provided later in this section.)

After the character is drawn, the pen is automatically advanced by the character width. The next character drawn will have its character origin at this new pen position (if the pen hasn't been moved first). Characters drawn one after the other are thus strung out horizontally in the expected manner.

For most characters in most fonts, the character image will lie between the old pen position and the new one. In fact, the new pen position will usually be several pixels to the right of the rightmost pixels of the character image, this supplies the small amount of blank space between characters. However, some characters in some fonts may have foreground pixels that lie to the left of the old pen position or to the right of the new pen position (or both). This is called **kerning**. When kerning occurs, the character images of adjacent characters (that is, characters drawn one after another) may possibly overlap.
The **character bounds rectangle** determines the extent of the background pixels of a character. The character bounds rectangle, relative to a current pen position and starting from the character origin, extends as follows:

- As far up as the font's ascent
- As far down as the font's descent
- As far left as the current (old) pen position, or as far as the character's leftmost foreground pixel, whichever is farther left (it is the leftmost foreground pixel if the character kerns to the left)
- As far right as the subsequent (new) pen position, or as far as the character's rightmost foreground pixel, whichever is farther right (it is the rightmost foreground pixel if the character kerns to the right)

Because pen positions are points, not pixels, the phrase *as far left as the current pen position* means that it includes the pixels immediately to the right of the current pen position. Similarly, *as far right as the subsequent pen position* means that it includes the pixels immediately to the left of the subsequent pen position.

This defines the character bounds rectangle, as shown in Figure 16-18.

---

Figure 16-18
Character bounds rectangle

The character bounds rectangle contains all the foreground pixels of the character; that was the point of extending it as far as any kerning the character does in either direction. The **background pixels** of a character are defined to be all pixels in the character bounds rectangle that are not foreground pixels. Thus, the background pixels include all the pixels corresponding to 0 bits in the character image; in addition, they generally include those pixels extending from the old pen position to the new.
When QuickDraw II draws a character—say, using the pen mode modeCopy, with a foreground color of red and a background color of green—the foreground pixels are colored red and the background pixels are colored green. Our definition ensures that the background will at least go from pen position to pen position and that it will go far enough so no kerning foreground pixels will extend beyond the background.

The QuickDraw II routine CharBounds calculates the character bounds rectangle of a specified character in local coordinates based on the current pen position. Every pixel that could be affected by drawing the character is located inside the character bounds rectangle. This is different from the QuickDraw II call CharWidth, which simply returns the character width of a specified character—that is, the amount by which the pen position would be advanced if the character were drawn. The width of the character bounds rectangle is not the same as the character width; either one may be larger than the other, or one may even be 0 and the other nonzero. It is not necessary, however, that the widths be different, and for some characters, they may be the same.

\[ \text{Note: Neither CharBounds nor CharWidth actually draws the character.} \]

When QuickDraw II draws a string (whether a Pascal-type string, a C-type string, or a text block), it draws the individual characters of the string into an internal text buffer, advancing the position in the text buffer by the character width after each character and ORing the character images together whenever they overlap (as they may with kerning). Then the entire string is drawn into the destination pixel image, using the current text mode, foreground and background colors, and so on. The pen position is advanced by the sum of the character widths of all the characters in the string.

The QuickDraw II routines StringWidth, CStringWidth, and TextWidth return the amount the pen would be advanced if the specified string or text were to be drawn; that is, they return the sum of the character widths of all the characters in the string or text.

The QuickDraw II routines StringBounds, CStringBounds, and TextBounds return the smallest rectangle that would enclose all the foreground and background pixels of the string if it were drawn; in effect, they return the string bounds rectangle. This is the same as the UnionRect of all the individual character bounds rectangles (if the characters were drawn one after another).

**Important**

The rectangle is not necessarily the same as the rectangle you would get if you strung out the character bounds rectangles one after another, with the right edge of each touching the left edge of the next. Because of kerning, the character bounds rectangles of characters in a string may overlap.

Note: Neither the bounds calls nor the width calls actually draw anything.
The bounds calls and width calls take into account any active style modifications, chExtra and spExtra values, and fontFlags settings that may affect either the area covered by foreground and background pixels or the amount the pen is advanced after text drawing. The QuickDraw II routine GetFontInfo takes into account the style modifications, but not the values of chExtra, spExtra, or fontFlags. The QuickDraw II routines GetFontLore and GetFontGlobals report on the current font as it exists in memory and do not take into account any of the other values mentioned.

In addition to the pen modes, which can be used for text, text can also be drawn in eight special text-only modes (four modes and their opposites). The uses for the different text modes are shown in Table 16-3. The opposite modes (notForeCopy, notForeOR, notForeXOR, and notForeBIC) work the same way as the original modes, except that the foreground pixels are turned to background pixels and the background pixels are turned to foreground pixels before the operation is performed. For more detail on how to use the modes, see the section "SetTextMode" in this chapter.

Table 16-3
Text modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modeForeCopy, notForeCOPY</td>
<td>Copies only the foreground pixels into the destination—background pixels are not altered</td>
</tr>
<tr>
<td>modeForeOR, notForeOR</td>
<td>ORs only the foreground pixels into the destination—background pixels are not altered</td>
</tr>
<tr>
<td>modeForeXOR, notForeXOR</td>
<td>XORs only the foreground pixels into the destination—background pixels are not altered</td>
</tr>
<tr>
<td>modeForeBIC, notForeBIC</td>
<td>BICs only the foreground pixels into the destination; that is, inverts the source pixels and ANDs them with the destination—background pixels are not altered</td>
</tr>
</tbody>
</table>

If you need to know more about how fonts are drawn and constructed (if, for example, you want to write a font-editing application), see the section "Fonts and Text in QuickDraw II" in this chapter.
**Drawing in color**

The video display hardware of the Apple IIGS includes advanced color capabilities. Although tool calls make it unnecessary for you to manipulate the hardware directly, knowledge of a few background concepts will help you understand the way QuickDraw II manipulates the colors on the screen.

The Apple IIGS offers two Super-Hi-Res graphics modes. Both modes have 200 scan lines, but the scan lines differ in horizontal resolution—one mode has 320 pixels (the color of each specified by four bits), and the other has 640 pixels (the color of each specified by two bits). In changing from 320 mode to 640 mode, the horizontal resolution is doubled at the expense of dividing the color resolution by 4.

Both modes use a **chunky pixel** organization (in which the bits for a given pixel are contained in adjacent bits within one byte), as opposed to **bit planes** (in which adjacent bits in memory affect adjacent pixels on the screen). Therefore, the four bits of a pixel in 320 mode are in the same memory locations as the four bits of a pair of adjacent two-bit pixels in 640 mode.

Colors on the Apple IIGS are determined from **master color values**, which are mathematical combinations of the primary red, blue, and green hues available on a color monitor. A master color value is a two-byte number, formatted as shown in Figure 16-19.

![Figure 16-19: Master color value](image)

A three-digit hexadecimal number can describe each master color, with one digit ($0$–$F$) for each primary color. Thus, a master color value of $000$ denotes black, $FFF$ is white, $00F$ is the brightest possible blue, $080$ is a medium-dark green, and so on. Because each primary color has 16 possible values, $4,096$ colors are possible.

At any one time, the Apple IIGS uses only a small subset of all possible colors. An application does so by constructing one or more **color tables**, short lists of the available colors for any one pixel.
Color tables and palettes

Pixels contain only two or four bits, and it takes 12 bits to specify a master color value. Thus, applications cannot use master color values to directly specify pixel colors. Instead, the pixel value is a two- or four-bit offset into a color table.

A color table is a table of 16 two-byte entries. Each entry in the table is a master color value; any of the 4,096 possible color values may appear in any position in the color table. The colors available to the application, as specified in its color tables, constitute its palette.

Pixels in 320 mode are represented in memory by four-bit integers. For each pixel, that four-bit value is used as an offset into a color table. With four bits, there are 16 possible pixel values, so the available colors for each pixel in 320 mode equals 16—the entire color table—as shown in Figure 16-20.

![Figure 16-20](image)

Accessing the color table in 320 mode
Pixels in 640 mode are represented in memory by two-bit integers. With two bits, there are only four possible pixel values to offset into the color table. To avoid limiting 640 mode to only four colors, however, each four adjacent pixels in 640 mode use four different parts of the same color table; a color table, then, consists of four minipallettes, which needn't have the same sets of master colors. Therefore, although each individual pixel in 640 mode can have one of only four colors, groups of four pixels can have a total of 16 colors from which to choose, as shown in Figure 16-21.

![Figure 16-21](image)

Accessing the color table in 640 mode

How to use this ability to create a large variety of colors is described in the section "Dithered Colors in 640 Mode" in this chapter.
Scan line control bytes

An application may construct as many as 16 different color tables to choose from. Each of the 200 scan lines in Super Hi-Res graphics can use any one of the 16 tables. For each scan line, a scan line control byte (SCB) decides which color table is active, as shown in Figure 16-22.

The SCB also controls screen display mode (320 or 640), interrupt mode, and fill mode.

**Interrupt mode:** Interrupts can be used to synchronize drawing with vertical blanking so pixels are not changed as they are being drawn (a pixel is drawn once every 1/60 of a second). Interrupts can also be used to change the color table before a screen is completely drawn. This allows a program to show more than 256 colors on the screen at once but costs the overhead of servicing the interrupt.

**Fill mode:** When fill mode is active, pixel values of 0 can be used to fill areas of color in 320 mode.

*Note:* Fill mode works only in 320 mode.

A pixel with a numeric value of 0 serves as a placeholder indicating that the pixel should be displayed as the same color last displayed, as shown in Figure 16-23.

![Scan line control byte diagram](chart)

**Figure 16-22**
Scan line control byte

**Figure 16-23**
Fill mode example
Standard color palette in 320 mode

The standard palette (the default color table) for 320 mode is shown in Table 16-4.

Table 16-4
Standard palette in 320 mode

<table>
<thead>
<tr>
<th>Offset</th>
<th>Color</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Black</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>Dark gray</td>
<td>777</td>
</tr>
<tr>
<td>2</td>
<td>Brown</td>
<td>841</td>
</tr>
<tr>
<td>3</td>
<td>Purple</td>
<td>72C</td>
</tr>
<tr>
<td>4</td>
<td>Blue</td>
<td>00F</td>
</tr>
<tr>
<td>5</td>
<td>Dark green</td>
<td>080</td>
</tr>
<tr>
<td>6</td>
<td>Orange</td>
<td>F70</td>
</tr>
<tr>
<td>7</td>
<td>Red</td>
<td>D00</td>
</tr>
<tr>
<td>8</td>
<td>Beige</td>
<td>FA9</td>
</tr>
<tr>
<td>9</td>
<td>Yellow</td>
<td>FFO</td>
</tr>
<tr>
<td>10</td>
<td>Green</td>
<td>0E0</td>
</tr>
<tr>
<td>11</td>
<td>Light blue</td>
<td>4DF</td>
</tr>
<tr>
<td>12</td>
<td>Lilac</td>
<td>DAF</td>
</tr>
<tr>
<td>13</td>
<td>Periwinkle blue</td>
<td>78F</td>
</tr>
<tr>
<td>14</td>
<td>Light gray</td>
<td>C CC</td>
</tr>
<tr>
<td>15</td>
<td>White</td>
<td>FFF</td>
</tr>
</tbody>
</table>

Note: "Offset" means position in the color table and "Value" means master color value, the hexadecimal value controlling the fundamental red-green-blue intensities.

Dithered colors in 640 mode

Only four colors are available for each pixel in 640 mode. But when small pixels of different colors are next to each other on the screen, their colors blend. For example, a black pixel next to a white pixel appears to the eye as a larger gray pixel. By cleverly choosing the entries in the color table, we can make more colors appear on the screen. This process is called **dithering**.

At the same time, to preserve the maximum resolution for displaying text, both black and white must be available for each pixel. This leaves only two remaining colors per pixel to choose from, which seems like a severe restriction. But with dithering, you can have 640-mode resolution for text and still display 16 or more colors if you are willing to resort to a few simple tricks.

Consider the following byte with four pixels in it:

<table>
<thead>
<tr>
<th>Bit value</th>
<th>0:1 0:1 0:1 0:1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel number</td>
<td>1 2 3 4</td>
</tr>
</tbody>
</table>

Each pixel has the value 1, which is an index into the second place in each of the color table's minipalletees (see Figure 16-21). So pixel 1's color is determined by entry 1 in minipalette 1, pixel 2's color is determined by entry 1 in minipalette 2, and so on.
If we use the standard 640-mode color table (shown in Table 16-5), pixels 1 and 3 will appear blue ($00F$) and pixels 2 and 4 will appear red ($0D0$). The eye will average these colors and see violet.

Table 16-5
Standard palette in 640 mode

<table>
<thead>
<tr>
<th>Offset</th>
<th>Color</th>
<th>Value</th>
<th>Minipalette offset</th>
<th>Offset</th>
<th>Color</th>
<th>Value</th>
<th>Minipalette offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Black</td>
<td>000</td>
<td>0</td>
<td>8</td>
<td>Black</td>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Blue</td>
<td>00F</td>
<td>1</td>
<td>9</td>
<td>Blue</td>
<td>00F</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Yellow</td>
<td>FF0</td>
<td>2</td>
<td>10</td>
<td>Yellow</td>
<td>FF0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>White</td>
<td>FFF</td>
<td>3</td>
<td>11</td>
<td>White</td>
<td>FFF</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Black</td>
<td>000</td>
<td>0</td>
<td>12</td>
<td>Black</td>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>Red</td>
<td>D00</td>
<td>1</td>
<td>13</td>
<td>Red</td>
<td>D00</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Green</td>
<td>0E0</td>
<td>2</td>
<td>14</td>
<td>Green</td>
<td>0E0</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>White</td>
<td>FFF</td>
<td>3</td>
<td>15</td>
<td>White</td>
<td>FFF</td>
<td>3</td>
</tr>
</tbody>
</table>

Note: The entries in the minipalletes for the standard 640-mode color table are set up so black and white appear in the same positions in each palette. This provides pure black and white at full 640 resolution, allowing crisper text display.

There are 16 different combinations of values a pair of pixels can assume in 640 mode, meaning that you can obtain 16 colors by dithering. To implement it, just make sure that the pattern you use for drawing or filling consists of a repeating array of four-bit (two-pixel) values.
Cursors

A cursor is a small image that appears on the screen and is controlled by a mouse. (The cursor appears only on the screen, never in an off-screen pixel image.) The cursor record contains the height and width of the cursor, the cursor image of the cursor, the mask controlling the appearance of the cursor, and a hot spot defining where the image of the cursor will be placed by the mouse, as shown in Figure 16-24.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>cursorHeight</td>
<td>Word—INTEGER: total number of horizontal slices in cursor</td>
</tr>
<tr>
<td>1</td>
<td>cursorWidth</td>
<td>Word—INTEGER: number of words wide in single horizontal slice of cursor</td>
</tr>
<tr>
<td>2</td>
<td>cursorImage</td>
<td>x Bytes—Array of words; cursor image (last word in each slice must be 0)</td>
</tr>
<tr>
<td>3</td>
<td>cursorMask</td>
<td>x Bytes—Array of words; cursor mask (last word in each slice must be 0)</td>
</tr>
<tr>
<td>4</td>
<td>hotSpotY</td>
<td>Word—INTEGER: Y coordinate of hot spot</td>
</tr>
<tr>
<td></td>
<td>hotSpotX</td>
<td>Word—INTEGER: X coordinate of hot spot</td>
</tr>
</tbody>
</table>

Figure 16-24
Cursor record

Note: Because of its variable size, the cursor record is not provided in the APW interface file.

The cursor appears on the screen as the size defined by the cursorHeight and cursorWidth fields in the cursor record. The appearance of each pixel is determined by the corresponding bits in the data mask and by the pixel under the cursor (that is, by the pixel already on the screen in the same position as this bit of the cursor). The image on the screen is obtained by ORing the mask with the destination and XORing that result with the cursor image.

The hot spot aligns a pixel in the cursor image with the mouse location. Thus, a hot spot of (0,0) is at the top left of the image, and a hot spot of (8,8) would be in the center of a cursor defined as 16 pixels wide and 16 pixels high.
The arrow cursor in 320 mode is defined as shown in the following assembly-language fragment:

```
  dc i'11,4'                    ; Eleven slices by 4 words
  dc h'0000000000000000'        ; Cursor image
  dc h'0f00000000000000'        
  dc h'0FF0000000000000'       
  dc h'0FFFF00000000000'       
  dc h'0000OFF000000000'       
  dc h'0000000000000000'       
  dc h'FF00000000000000'       ; Mask image
  dc h'FFF0000000000000'        
  dc h'FFFF000000000000'       
  dc h'FFFFF000000000000'      
  dc h'FFFF0000000000000'      
  dc h'FFFFF0000000000000'     
  dc h'FFFF0000000000000'      
  dc h'FFFFF0000000000000'     
  dc h'FFFF0000000000000'      
  dc h'FFFFF0000000000000'     
  dc h'FFFF0000000000000'      
  dc h'FFFFF0000000000000'     
  dc h'FFFFF0000000000000'     
  dc h'FFFFF0000000000000'     
  dc h'FFFFF0000000000000'     
  dc h'FFFFF0000000000000'     
  dc h'FFFFF0000000000000'     
  dc h'FFFFF0000000000000'     
  dc i'1,1'                    ; Hot spot
```
Using QuickDraw II

This section discusses how the QuickDraw II routines fit into the general flow of an application and gives you an idea of which routines you'll need to use under normal circumstances. Each routine is described in detail later in this chapter.

QuickDraw II depends on the presence of the tool sets shown in Table 16-6 and requires that at least the indicated version of the tool set be present.

Table 16-6
QuickDraw II—other tool sets required

<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
<th>Minimum version needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01</td>
<td>#01 Tool Locator</td>
<td>1.0</td>
</tr>
<tr>
<td>$02</td>
<td>#02 Memory Manager</td>
<td>1.0</td>
</tr>
<tr>
<td>$03</td>
<td>#03 Miscellaneous Tool Set</td>
<td>1.0</td>
</tr>
</tbody>
</table>

The first QuickDraw II call your application must make is QDStartUp. Conversely, when you quit your application, you must make the QDShutDown call.

All graphic operations are performed in GrafPorts. Before a GrafPort can be used, it must be opened with the OpenPort routine. Normally, you don't call OpenPort yourself—in most cases, your application will draw into a window you've created with Window Manager routines, and these routines call OpenPort to create the window's GrafPort. Similarly, a GrafPort's regions can be disposed of by the ClosePort routine. When you call the Window Manager to close or dispose of a window, it calls these routines for you.

In an application that uses multiple windows, each is a separate GrafPort. If your application draws into more than one GrafPort, you can call SetPort to set the GrafPort you want to draw in. At times, you may need to preserve the current GrafPort; you can do this by calling GetPort to save the current port, SetPort to set the port you want to draw in, and then SetPort again when you need to restore the previous port.

Some toolbox routines return or expect points that are expressed in a common, global coordinate system; others use local coordinates. For example, when the Event Manager reports an event, it gives the mouse location in global coordinates, but your application may need to know where the mouse location is in the window's local coordinates. The GlobalToLocal routine lets you convert global coordinates to local coordinates, and the LocalToGlobal routine lets you do the reverse.

The SetOrigin routine will adjust a GrafPort's local coordinate system. If your application performs scrolling, you'll use ScrollRect to shift the pixels of the image and then use SetOrigin to readjust the coordinate system after the shift.
You can redefine a GrafPort's clipping region with the SetClip or ClipRect routine. Just as GetPort and SetPort preserve the current GrafPort, GetClip and SetClip are useful for saving the GrafPort's clipping region while you temporarily perform other clipping functions. This is useful, for example, when you want to reset the clipping region to redraw the newly displayed portion of a document that's been scrolled.

The LineTo routine draws a line from the current pen location to a given point, and the Line routine draws a line as an offset from the current position. You can set the pen location with the MoveTo or Move routine; you can set other pen characteristics with SetPenSize, SetPenMode, and SetPenPat.

In addition to drawing text and lines, you can use QuickDraw II to draw a variety of shapes. Most of them are defined by a rectangle that encloses the shape. The following require you to call a series of routines to define them:

- To define a region, call the NewRgn routine to allocate space for it, then call OpenRgn, and then specify the outline of the region by calling routines that draw lines and shapes. End the region definition by calling CloseRgn. When you're completely done with the region, call DisposeRgn to release the memory the region occupies.
- To define a polygon, call the OpenPoly routine and then form the polygon by calling procedures that draw lines. Call ClosePoly when you're finished defining the polygon; call KillPoly when you're completely done with it.

You can perform the following graphic operations on rectangles, rounded-corner rectangles, ovals, arcs and wedges, regions, and polygons: framing, painting, erasing, inverting, and filling. These operations are described in the section "Basic Drawing Functions" in this chapter.

You'll use points, rectangles, and regions not only when drawing with QuickDraw II but also when using other parts of the toolbox. At times, you may find it useful to perform calculations on these entities. You can, for example, add and subtract points and perform a number of calculations on rectangles and regions, such as offsetting them, rescaling them, calculating their union or intersection, and so on.

Note: When performing a calculation on entities in different GrafPorts, you need to adjust to a common coordinate system first by calling LocalToGlobal to convert to global coordinates.

Warning

QuickDraw II doesn't forgive certain kinds of programming errors. Results will be unpredictable if you make any QuickDraw II calls before initializing QuickDraw II, pass any bad handles to QuickDraw II, or make any QuickDraw calls with a bad port. Even an error code probably won't be returned if one of these kinds of errors is made, and your program may never get control again after one of these errors. Your application may not fail immediately; it may work for a while and then later fail for no apparent reason. You must be certain these types of errors can't occur in your program.
Fonts and text in QuickDraw II

This section contains a detailed description of the handling of fonts and text in QuickDraw II, including the definition of an Apple IIGS font.

•:•

Note: Most application writers, most of the time, will not need any more information than is included in the previous sections about text handling in QuickDraw II. But if you are designing a font, writing a font editor, or using unusual fonts or an unusually large variety of fonts, you'll need the information presented in this section.

•:•

Macintosh programmers: The treatment of text drawing and text measurement on the Apple IIGS is similar to their treatment on the Macintosh. The IIGS font definition is similar to that of the Macintosh, and a simple conversion algorithm allows the IIGS to use any font developed for the Macintosh. Most Macintosh QuickDraw text calls are duplicated precisely in QuickDraw II. Any differences are due to one or more of the following:

• Some information has been added to the beginning of the font definition.

• Because Macintosh-like resources do not exist, the IIGS Font Manager performs differently from the Macintosh Font Manager.

• Some bounding box calls (TextBounds and its siblings) missing from Macintosh QuickDraw have been added.

• Some calls—DrawCString, CStringWidth, and the like—have been added to handle the C string data type (a sequence of characters terminated by a 0 byte).

• The Font Manager is not closely integrated with QuickDraw II. (The interaction between QuickDraw II and the Font Manager is quite different).

• QuickDraw II does not scale text. However, the Font Manager can scale fonts as required.

Font definition

An Apple IIGS font consists of a variable-length header, followed by a Macintosh font record (this embedded Macintosh font is referred to as the MF part of the IIGS font). The header is of variable length to allow extra information to be added in the future.

The MF part of a IIGS font is exactly like a Macintosh font, except for one thing—the high-order and low-order bytes of integers. The Macintosh's 68000 microprocessor stores integers with the high-order byte first (that is, high-order byte at lower memory location); the IIGS's 65816 microprocessor stores them with low-order byte first. In converting a Macintosh font to a IIGS font, the high-order and low-order bytes of each integer are swapped. This does not apply to the font strike (called bitImage in the font definition), which can be used as is.

Apple IIGS font definition

The IIGS font record is shown in Figure 16-25.
Figure 16-25
Font definition
Apple IIGS font header fields

Some information about the font is contained in the header. Because fonts designed at a later time may include additional information that could be utilized by later versions of QuickDraw II, the header is of variable length. For upward and downward compatibility of QuickDraw II and IIGS fonts, the following two fields are particularly useful:

**offseToMF**: Offset, in words, from this field to the Macintosh font (MF) part included in the IIGS font (specifically, to the `fontType` field). The header is therefore $2 \times \text{offseToMF}$ bytes long. In the version of QuickDraw II current at the time of this manual’s publication, $\text{offseToMF} = 6$; thus, the header is 12 bytes. Future fonts may have longer headers that contain font information that can be utilized by future versions of QuickDraw II. To ensure that these improved fonts can be used by older versions of QuickDraw II, the `offseToMF` field provides a reliable jump over this extra font information to the start of the Macintosh part of the font. An older QuickDraw II will not be able to make use of new header fields added since it was implemented, but at least it will be able to find the information it can use.

**version**: Version number of the font definition under which the font was designed. By checking this field, later versions of QuickDraw II can avoid trying to find and use information not included in an older font. (Presumably a newer QuickDraw II, alerted by the version number to the lack of such information, would use some default or calculated values.) The font definition described in this manual is 1.1 ($0101$).

Examples of extra information that may be included in later fonts and used by later versions of QuickDraw II include thickness of underline, slope of italicized letters, smearing extent of boldface, and the like (on the Macintosh, these are determined by the Font Manager).

The other header fields are

**family**: Integer identifying the font, regardless of size or style. This can be thought of as corresponding to the font’s name—Courier or Geneva, for example.

**style**: Style in which the font was designed. For example, application writers and graphic designers may design italic or bold fonts for reasons of aesthetics or time performance. When QuickDraw II is asked to apply a certain style when drawing a character or string, it first checks this field. If the field indicates that the requested style is already part of the font, the drawing call will not apply the styling algorithm. This would, for example, prevent a preitalicized font from being realitalicized.

**size**: Relative measure of the size of fonts. The measure is analogous to the Macintosh point size; however, the actual font size is different from the true point size in a typographic sense.

**jbrExtent**: Maximum horizontal distance, in pixels, from the character origin to the far edge of any foreground or background pixel of any character in the font. (See the section “Font Bounds Rectangle” in this chapter for a more precise definition.)
Macintosh font part of an Apple IIGS font

A Macintosh font, or in this case the Macintosh font part of an IIGS font, consists of four sections:

1. A fixed-length record containing general information, such as font height and maximum character width.

2. The font strike (named bitimage in the font record definition), which is a pixel image containing the image of every character defined in the font, strung one after another. The pixel image is in a one-bit-per-pixel form. Its width, measured in words, is given by the rowWords field of the font record; its height, measured in pixels, is given by fRectHeight.

3. The location table (locTable), an array of integers that indicates for each defined character where its image in the font strike begins.

4. The offset/width table (owTable), an array of integers. For each character, the low-order byte of its entry in the offset/width table (the character offset) indicates how the character image to be drawn should be positioned with respect to the current pen location; the high-order byte indicates how far the pen should be advanced after the character is drawn (character width).

This table is also used to identify characters not defined in a font. An owTable value of -1 ($FFFF) marks a missing character, which must be handled specially by the text calls.

A detailed description of the meanings and uses of these various fields and arrays is given in the sections that follow.

Characters

A character image is a rectangular array of bits, representing pixels. The on, or 1, bits are called the character foreground pixels.

The number of columns in a character image is called the character image width, or just the image width. Note that a character can have an image width of 0. For example, the space has a 0 image width; its character image consists of no pixels at all.

The character rectangle is a rectangle that encloses the character image. Its width is the image width of the character, which may vary from character to character in a font; its height is the character height, which is the same for all characters in a font.
Each character has a number associated with it. This number, called the **character width**, and found in the offset/width table, is the number of pixels the pen position is to be advanced after the character is drawn. This is different from the image width, and the distinction between the two is important. For example, the space character has 0 image width but does have some positive character width, which determines the size of the space. Some characters have a nonzero image width but a 0 character width; an example of this is an umlaut, which is meant to be typed over a vowel. The umlaut is drawn first, and then the vowel is drawn with the same pen location. Characters with 0 character width are called **dead characters**.

Also associated with every character in a font are its **base line** and its character origin. The **base line** is a horizontal line that separates the image into two sets of rows, one set above and one below. (Remember that in QuickDraw II, as in QuickDraw, horizontal and vertical lines fall between pixels rather than running through them.) The position of the base line depends on the font’s **ascent** and **descent** fields; it is chosen so there are ascent rows above it and descent rows below. The base line will be in the same horizontal position for every character in the font.

Any foreground pixels of a character image that lie below the base line are collectively called the character’s **descender**. Most characters don’t have a descender, but in an average font, characters like q and y do.

The **ascent line** is the horizontal line just above the top row of a character; the **descent line** is the line just below the bottom row. These will be the same for every character in the font.

The **character origin** of a character is a point on the base line used to position the character for drawing. This point may be between pixels of the character image, to the right of them, or to the left. (Here, note that points lie between pixels, not on them.) Its location relative to the character image can be calculated by the character offset in the offset/width table, as detailed in later sections of this chapter. When the character is drawn, it is placed in the destination pixel image so that its character origin coincides with the current pen location.

For many letters, the character origin is located on the left edge of the character image so that, when the character is drawn, its leftmost foreground pixels fall just to the right of the pen. Sometimes the character origin is between pixels of the character image (or, rarely, entirely to the right of the image). When such a character is drawn, some of its pixels will fall to the left of the pen position. This is called kerning to the left. In such a case, the distance, in pixels, from the character origin to the left edge of the character is called the character’s **leftward kern**.

When character-image pixels fall to the right of the new pen position after the character is drawn, the character is said to kern to the right. The **kernMax** field in a font is concerned only with kerning to the left. Kerning in either direction can cause letters to overlap each other. See Figures 16-26 and 16-27.
Figure 16-26
Character with no kerning

Figure 16-27
Character kerning left
**Fonts**

**Font rectangle**

Imagine all the defined characters of a font drawn so their character origins coincide. The result would be a black mess of foreground pixels. The smallest rectangle completely enclosing this mess is called the **font rectangle** (see Figure 16-28).

![Font rectangle diagram](image)

**Figure 16-28**

Font rectangle (simulated)

The fields of the font record that measure aspects of the font rectangle are described in the following list:

- **kernMax**: Distance, in pixels, from the font rectangle's common character origin to the left edge of the font rectangle. If any character in the font actually kerns to the left, **kernMax** is represented as a negative number. If the character origin lies on the left edge of the font rectangle, **kernMax** is 0.

Most fonts fall into these two categories. However, in some fonts, the left edge of the font rectangle is one or more pixels to the right of the character origin. In such a case, **kernMax** is assigned a positive value, even though this bends the terminology a bit, for example, people do not usually say of a character that leaves two columns of blank pixels between the pen position and its image that it kerns to the left 2 pixels, or -2 pixels, or anything at all.

- **fRectWidth**: Width, in pixels, of the font rectangle. Note that this may be more than the maximum character image width because the font rectangle's left and right extremes may come from different characters.

- **fRectHeight**: Height, in pixels, of the font rectangle.
ascent: Number of pixel rows above the common base line in the font rectangle.

descent: Number of pixel rows below the base line in the font rectangle. Note that
\( fRectHeight = ascent + descent \).

nDescent: Negative of descent.

\* Note: For typical fonts—those in which the font rectangle at least touches its character origin—\( ascent \) and \( descent \) will be non-negative, and \( kernMax \) and \( nDescent \) will be nonpositive. However, fonts can be designed without these restrictions.

fontType: QuickDraw II ignores this field.

firstChar: ASCII code of the first defined character in the font.

lastChar: ASCII code of the last defined character of the font.

wMax: Maximum character width (pen displacement) of any character in the font, measured in pixels.

owTLoc: Offset, in words, from this field to the font offset/width table (owTable). By adding \( 2 \times owTLoc \) to the memory address of this field, you get a pointer to the owTable. There is no corresponding field for the location table in the font record; to get a pointer to the locTable, you must subtract \( 2 \times (lastChar - firstChar + 3) \) from the owTable pointer.

leading: Recommended number of blank pixel rows between the descent row of one line of text and the ascent row of the next. Applications may use this or not, as they please.

rowWords: Width of the font strike, in words. This is discussed further in the next section, "Font Strike."

Font strike

The font strike (called builtImage in the font definition) is a one-bit-per-pixel pixel image consisting of the character images of every defined character in the font, placed sequentially in order of increasing ASCII code from firstChar to lastChar +1. The character images in the font strike abut each other; no blank columns are left between them; see Figure 16-29.

![Font strike image](image-url)
Because all the characters of a font have the same height, the font strike is just one long pixel image with no jumps or undefined stretches and with a height of $jRectHeight$. The strike is padded on the right, if necessary, with enough extra pixels on each row to make the row width a multiple of 16—that is, to make each row an integral number of words. This width, measured in words, is found in the $rowWords$ field of the font record.

**Defined versus undefined characters**

Not every possible ASCII code must have a character image in the font strike. The font may leave some characters undefined; these are called **missing characters**. Every character with a code less than $firstChar$ or greater than $lastChar + 1$ is undefined. There may be other undefined characters as well. The offset/width table ($owTable$) has an entry for every code from $firstChar$ to $lastChar + 2$, inclusive. If a character’s entry in the offset/width table is $-1$ ($FFFF$), the character is undefined or missing.

Character code $lastChar + 1$ is a special case. Immediately following $lastChar$ in the font strike is a character (known as the **missing symbol**) that is to be used in place of any missing character. This character must be present in the font strike. It has entries in the $locTable$ and the $owTable$, and its entry in the $owTable$ must not be $-1$. For all purposes, the missing symbol is a defined character with ASCII code $lastChar + 1$. In many fonts, the missing symbol is a hollow rectangle; in the current system font, it is a white-on-black question mark.

Whenever the QuickDraw II text-handling routines encounter a missing character—less than $firstChar$, greater than $lastChar + 1$, or having an $owTable$ entry of $-1$—the routines immediately substitute the missing symbol for the character, using the missing symbol’s character image, $locTable$ entry, and $owTable$ entry wherever needed.

**Location table**

The $locTable$ is an array of integers with an entry for each character code from $firstChar$ to $lastChar + 2$. It is used to find character images in the font strike. For each defined character, the $locTable$ entry gives the distance, in pixels, from the beginning of the font strike to the beginning of the character’s image in the font strike ("beginning," here, means left edge). This indicates where the character image starts. To see where it ends, take the next $locTable$ entry (the beginning of the next character image) and subtract 1. Because the character images abut each other, this will give you the precise limits of the character image. The image width of a defined character with code $C$ is $locTable[C + 1] - locTable[C]$. This may be 0.
For this scheme to work, two conditions must hold:

1. The \textit{locTable} entry for an undefined character must be the same as the entry for the next defined character. This prevents undefined characters, which have no image in the strike, from interfering with the hunt for images of defined characters.

   Note that there always will be a next defined character because the missing symbol, which serves as a defined character, is tacked on at the end of the strike.

2. To get the character image for the missing symbol, there has to be an entry in the \textit{locTable} following the missing symbol entry. For this reason \textit{locTable[lastChar + 2]} is included and is set equal to the length of the font strike in pixels, ignoring the padding that is added to the font strike to align it to word boundaries.

\textbf{Offset/width table}

The offset/width table (\textit{owTable}) is an array of integers with an entry for each character code from \texttt{firstChar} to \texttt{lastChar + 2}. If a character's entry is \texttt{-1}, the character is undefined (missing). Otherwise the entry's low-order and high-order bytes are the character width and character offset, respectively. Both are interpreted as numbers in the range 0–254 (255 is ruled out to avoid the case where both bytes are 255, giving an entry of \texttt{-1}, which would mark a missing character).

The character offset is used to calculate the position of character origin relative to the image in the following way: The offset is added to the font's \texttt{kernMax}. The result is the (horizontal) distance, in pixels, from the character origin to the left edge of the image. If the result is negative, then the origin is to the right of the image's left edge (the character kerns leftward). If the result is positive, the origin is to the left of the image's left edge. (A result of 0 means that the character origin sits on the left edge of the image). Because we already know that the character origin must lie on the base line (whose position is determined from ascent and descent), this precisely locates the origin.
If you draw the font rectangle and look at a particular character's character rectangle within it, the character offset is seen to be the offset, in pixels, between the left edge of the font rectangle and the left edge of the character rectangle. See Figure 16-30.

![Character rectangle in font rectangle](image)

**Figure 16-30**
Character rectangle in font rectangle

The low-order byte of the *owTable* entry gives the character width, which is the distance, in pixels, the pen should be advanced to the right after the character is drawn. In applications, this distance can be affected by a number of calls, particularly *SetCharExtra* and *SetSpaceExtra*. There is, however, a general rule in QuickDraw II: any character whose character width (taken unmodified from the *owTable*) is 0 will not have that width changed by *chExtra*, *spExtra*, style modifications, nonproportionality, or any other effect. We assume that characters are given 0 width only for some very good reason.

---

**Warning**

Any modification, or any combination of modifications, that results in a character width of less than 0 or greater than 255 will wreak havoc with the drawing routines and is not allowed. This includes *chExtra*, *spExtra*, and style modifications, among others. QuickDraw II does not check for this condition. That is up to you.

---

The lastChar + 2 entry of the *owTable* is set to -1.
Character backgrounds

A character's foreground consists of all the on pixels (1 bits) in its image. The off pixels (0 bits) are part of the background. In QuickDraw II, the background is extended on the left to include any pixels that are to the left of the image's left edge but to the right of the character origin (and between the ascent and descent lines). On the right side, the background is extended to include any pixels (between ascent and descent) that are to the right of the image's right edge but to the left of the character origin of the next character (that is, to the left of the new pen position). Any new pixels added in this way are also considered background pixels. In other words, the foreground of a character consists of all 1 bits in its character image. The background consists of all 0 bits in the image plus all nonforeground pixels that are to the right of the character origin, to the left of the subsequent character origin (character origin + character width), above the descent line, and below the ascent line. In some cases, no extending is needed. If the character kerns to the left, no left extension is necessary; if it kerns to the right, no right extension is needed.

This is a very natural definition of background. If you're going to draw a green character that doesn't stretch entirely from the old pen position to the new and you have a red background, the red background will usually extend a little to the left and/or right of the character's image. This is what people generally want for a background. But in addition to this, when characters do kern, the background extends as far left or right as the kerning, so the kerned part of the character doesn't jut out past the character's background.

This brings us to the definition of the **character bounds rectangle** (Figure 16-31). It is the smallest rectangle enclosing all the foreground and background pixels of a character. It may be somewhat larger than the character rectangle, which encloses the image, because the bounds rectangle takes into account the character width (pen positions) as well as the image width. The width of a character's bounds rectangle is called the **character bounds width**. QuickDraw II includes calls for measuring character bounds rectangles and corresponding routines for strings, C strings, and text.
To get some new, useful measures for the width of a font, we define the font bounds rectangle. Imagine that, for all of a font's characters, the characters' bounds rectangles were drawn so all the character origins coincided. The resulting rectangle (more precisely, the rectangle that is the union of all these rectangles) is called the **font bounds rectangle**. This rectangle, illustrated in Figure 16-32, includes all pixels, foreground and background, of every character in the font. (Consequently, it may be bigger than the font rectangle, which is only guaranteed to include all the foreground pixels.)

We define \( \text{jbrWidth} \) to be the width of the font bounds rectangle, \( \text{jbrRightF.xtent} \) to be the distance from the common character origin to the right edge of the font bounds rectangle, and \( \text{jbrLeftF.xtent} \) to be the distance from the origin to the left edge (all distances measured in pixels and as positive numbers). Finally, we define \( \text{jbrExtent} \) to be the maximum of \( \text{jbrLeftF.xtent} \) and \( \text{jbrRightF.xtent} \).

The \( \text{jbrExtent} \) value is the farthest possible horizontal distance from the pen location to the far edge of any pixel that can be altered by drawing any character in the font. In many ways, this is a more precise measure of the width of a font than \( \text{widMax} \) or \( \text{fRectWidth} \).

It would seem from Figure 16-32 that \( \text{jbrLeftF.xtent} \) and \( \text{kernMax} \) are the same, or rather that \( \text{jbrLeftF.xtent} = -\text{kernMax} \). This is true if, and only if, \( \text{kernMax} \) is 0 or negative; if \( \text{kernMax} \) is positive (that is, if every character image starts at least 1 pixel to the right of the character origin) then \( \text{jbrLeftF.xtent} \) is 0. This makes \( \text{jbrLeftF.xtent} \) easy to calculate. It would also seem as if \( \text{jbrRightF.xtent} \) is the same as \( \text{widMax} \), but if any character kerns to the right beyond the reach of \( \text{widMax} \), \( \text{jbrRightF.xtent} \) will be bigger than \( \text{widMax} \).
Macintosh programmers: The *jbrExtent* field is needed for safe handling of the text buffer. It is not included in the Macintosh font definition. If you are converting a Macintosh font to the II GS, *jbrExtent* can be calculated by using the CharBounds call on character codes 0-255 and doing some simple arithmetic (the CharBounds call itself doesn't need a valid value for *jbrExtent*, so it can be called for the calculation). This has to be done only once for each font.

### Drawing and the text buffer

Whenever a character or string is to be drawn, it is first drawn into the text buffer, a one-bit-per-pixel pixel image reserved for the private use of the QuickDraw II text-drawing calls. For strings, only those characters that have a chance of making it into the destination pixel image are actually drawn; the others, both to the left and to the right, contribute only to the cumulative pen displacement. Thus, there is no reason for an application to try to clip characters out of long strings unless it has a very fast way of doing so.

The text buffer is empty at the beginning of each drawing call. Successive characters of a string are drawn into it, with an internal text buffer pen incremented by the character width each time. Regardless of the ultimate text mode (*txMode* in the GrafPort), characters are drawn into the text buffer in OR mode. Thus, characters that kern into each other do not interfere destructively. (For this reason, with certain text modes, such as modeForeXor, the results you get if you put up a string one character at a time with DrawChar can be different from those you get if you put up the whole string with DrawString. In the case of DrawChar, overlapping characters may cancel out some pixels.)

Once the character or string is safely in the text buffer, any requested style modifications (underlining, bolding, and the like) are applied to it. Then the text buffer is transferred to the destination. Individual bits are replaced with two or four bits, depending on the chunkiness of the destination; the bit patterns used are the Grafport's *fgColor* for the 1 (foreground) bits and *bgColor* for the 0 (background) bits. During the transfer, the text image is clipped to the current clipping region. The surviving pixels are combined with the destination's pixels, according to whatever text mode is in use. If and when the result makes it to the screen, the bit patterns will be translated into colored pixels according to the current color map(s).
Controlling text display

Various QuickDraw II calls affect text display. Generally, the calls set some field of the current GrafPort that is used in the text-drawing process. Matching the calls that set the fields are corresponding calls that return the value of the fields.

SetForeColor, GetForeColor, SetBackColor, GetBackColor, SetTextMode, and GetTextMode deal with the GrafPort fields fgColor, bgColor, and txMode, whose effects were described earlier. The other display control calls deal with character spacing, style modifications, boldfacing, underlining, and font flag options, as described in the following sections.

Character spacing calls

SetCharExtra, GetCharExtra, SetSpaceExtra, GetSpaceExtra: These calls set and/or get the chExtra and spExtra fields in the GrafPort; these fields can alter the character widths (pen displacement) when characters are drawn. Each is a fixed number with a one-word integer part and a one-word fractional part.

The chExtra field was included because some fonts that look fine in 320 mode appear too closely spaced in 640 mode. Putting an extra pixel between letters seems to help in these cases.

The chExtra value is added to the character width of each character, except a dead character, as it is drawn. The chExtra value is not added to dead characters, which have a character width of 0.

Adding chExtra to a character width will give us character origin positions that have a fractional part. During any text-drawing call, QuickDraw II keeps track of this fractional part and, when drawing a character, rounds its character-origin position to the nearest integer (1/2—that is, $8000—is rounded up). The fractional part is not remembered after the call has been completed.

Commonly used to help in justifying text, spExtra works the same way as chExtra except that it is only applied to the space character. Note that here “space character” means ASCII $20 and nothing else. In particular, the nonbreaking space included in many fonts is unaffected by the spExtra field. The spExtra value is cumulative with chExtra.

These values are set by the QuickDraw II routines SetCharExtra and SetSpaceExtra (and can be fetched by GetCharExtra and GetSpaceExtra). In theory, the application can set chExtra and spExtra to any fixed value, even a negative one. However, any values that cause a character to have a character width of less than 0 or greater than 255 pixels will cause no end of trouble.
**Style modification calls**

**SetTextFace, GetTextFace:** These calls set and get the `textFace` field of the GrafPort, which determines the style to be applied to the text. At the time of publication, the following bits were defined:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Bold</td>
</tr>
<tr>
<td>1</td>
<td>Italic—available only if QuickDraw II Auxiliary is loaded and started up</td>
</tr>
<tr>
<td>2</td>
<td>Underline</td>
</tr>
<tr>
<td>3</td>
<td>Outline—available only if QuickDraw II Auxiliary is loaded and started up</td>
</tr>
<tr>
<td>4</td>
<td>Shadow—available only if QuickDraw II Auxiliary is loaded and started up</td>
</tr>
</tbody>
</table>

These styles may affect the character width (pen displacement), image width, ascent, and descent associated with a character or string. For example, boldfacing spaces characters farther apart and makes the characters thicker. These changes are reflected in the results returned by the width routines (CharWidth, StringWidth, and so on), the bounds routines (CharBounds, StringBounds, and so on), and the GetFontInfo routine.

**Font flags option calls**

**SetFontFlags, GetFontFlags:** The `fontFlags` field of the GrafPort is set by `SetFontFlags` and fetched by `GetFontFlags`. At the time of publication, only the last three bits (bits 2-0) of this word are defined; bits 15-3 are reserved and should be set to 0.

**Bit 0:** If bit 0 is set, the font is used as a nonproportional font; every character, except characters with character width 0, is given the same character width, namely `widMax` (the maximum character width field from the font definition).

When nonproportionality is in effect, `chExtra`, `spExtra`, style modifications, and so forth, are applied to the character width after it has been set to `widMax`.

**Bit 1:** As of Version 2.0, if bit 1 is set and bit 0 is not, every character in the font is given the same character width, just as occurs with the nonproportionality setting.

However, in this case, the width used is the character width of the font's digit 0 (ASCII $30$).

This feature makes it easier to line up columns of figures. It makes all digits, spaces, periods, and the like, the same width. "Width," here, means character width; that is, the pen displacement after the character is drawn. (The image width of the characters remains unchanged.) Of course, standard nonproportionality would also make everything line up, but in most fonts, `widMax` is a good deal more than the width of a digit, which causes numbers to end up spaced too far apart.

Because the width used in numeric spacing is usually less than `widMax`, some characters—for instance, W's and M's—end up overlapping other characters. Consequently, numeric spacing is useful with the characters most commonly used with numbers—space, period, and so on—but is not appropriate for general text.
If you want absolute control over the width of characters, you can use numeric spacing or the standard nonproportionality and then adjust it to your tastes using the SetCharExtra routine.

When numeric spacing is in effect, chExtra, spExtra, style modifications, and the like, are applied to the character width after it has been set to the width of the digit 0.

Bits 1 and 0 should not both be set to 1.

**Bit 2:** Bit 2 controls how the foreground and background colors in the GrafPort are applied to text when it is drawn. If bit 2 is 0, the foreground and background colors are treated as pixel values (two- or four-bit numbers depending upon the GrafPort's SCB), with all other bits in the word ignored. Each foreground pixel is given the value of the foreground color value, and each background pixel is given the value of the background color value. For example, in 640 mode with a foreground color word of 0110011001100110 and bit 2 set to 0, each pixel will have a value of 10.

If bit 2 is set to 1, the foreground and background colors are treated as a word's worth of pixel values. This feature is useful when you are trying to draw text in 640 mode using dithered colors. Each foreground pixel in a destination word is given the value of the corresponding pixel in the foreground color word. Each background pixel in a destination word is given the value of the corresponding pixel in the background color word. For example, in 640 mode with a foreground color word of 0110011001100110 and bit 2 set to 1, odd-numbered pixels will have a value of 10 and even-numbered pixels will have a value of 01.

---

**Using the QuickDraw II font calls**

**Text drawing calls**

**DrawChar, DrawText, DrawString, DrawCString:** These calls are used when the specified character or string of characters is drawn, using all the current information—font, style, mode, and so forth. The current pen position is used as the character origin of the first character. The pen is advanced by the sum of the character widths. Note that, although the text image is clipped to the current clip region, the pen is not clipped in any way; the new pen position can be outside the current GrafPort bounds.

---

**Warning**

Near the edges of its drawing space (±16K, ±16K), QuickDraw II is unreliable; this applies to text drawing as well as to any other kind. Calls that would draw outside the space can cause catastrophic results.

---

Fonts and text in QuickDraw II 16-57
Text width calls

**CharWidth, TextWidth, StringWidth, CStringWidth:** These calls return the total pen displacement that would result if the character or sequence of characters were to be drawn. Nothing is actually drawn, however. The width calls take into account current styles, *chExtra, spExtra*, font flags, and the like. But they do not take kerning (which is independent of pen displacement) into account; that's a job for the text bounds calls. Note that the width calls only return a pen displacement, not a new pen location. They make no use of the current pen location, and they don't change it.

Text bounds calls

**CharBounds, TextBounds, StringBounds, CStringBounds:** These calls return the smallest rectangle that would enclose all the foreground and background pixels of the character or string (or text block or C string) of characters if they were to be drawn, starting at the current pen location. The rectangle is given in the local coordinates of the current GrafPort.

Unlike the text width calls, these calls take kerning, as well as pen movement, into account. The bounds rectangle extends to the left as far as the starting pen position or the leftmost kerning pixel (if any) of the text image, whichever is farther to the left. Similarly, it extends as far right as the new pen position or the rightmost kerning pixel, whichever is farther to the right. But at the least, the bounds rectangle is reliable; any pixel that might be changed by a text-drawing call is inside the corresponding bounds rectangle.

The rectangle extends up (from the current pen location) to the ascent line and down to the descent line. It is not clipped to any clipping region. It takes into account style modifications, *chExtra, spExtra*, and so forth. Note that the bounds rectangle is not actually drawn by these calls; its coordinates are simply returned to the application.

Some strings (or text or C strings), or possibly even some characters, may have no foreground or background pixels. Such a character would have to have 0 image width and 0 character width—a space with no length. A string may have 0 length (no characters) or be composed entirely of the spaceless spaces just described. In these cases, the text bounds calls return a degenerate rectangle; that is, one whose right and left edges are the same (namely, the current pen location's X coordinate). The upper and lower edges of the rectangle will be the ascent and descent lines (relative to the pen's Y coordinate), as usual.

Text buffer management calls

**SetBufDims, ForceBufDims, SaveBufDims, RestoreBufDims:** These calls affect the size of the text buffer and the way it is used.

*Note:* If you are using the Font Manager, it takes care of all this text buffer management for you.
Important

These calls affect the QuickDraw II clip buffer as well as the text buffer!

When a string (or text block or C string) is to be drawn into a pixel image, it is first drawn into the text buffer. Characters of the string that fall far outside the destination's left or right boundaries are not actually drawn into the text buffer; only their character widths are used—to determine where the string actually enters the destination (on the right) and/or what the final pen location should be (on the left).

For the text-drawing calls to handle this safely and efficiently, QuickDraw II must have certain information about the largest pixel image sizes and character sizes it will have to deal with. For one thing, the text buffer must be at least as wide (in pixels) as the widest destination pixel image that may be used (actually, it must be a little wider to avoid disaster when drawing characters that fall partly in and partly out of the destination), and it has to be as high as the highest font. For another thing, to decide if a pen location is so far outside the destination that a character drawn with that origin can't possibly impinge on the destination, QuickDraw II needs to know the width of the widest possible character. “Widest,” here, includes not only image width and character width, but also any elongations due to chExtra, spExtra, style modifications, and so forth. Any pixel that can be touched by a character's foreground or background must be considered.

This is what forExtent was created for. It describes how far away from the current pen location any pixel that can be altered can be. But forExtent depends only on the font and does not take into account style modifications and the like. This is why we have two calls: SetBufDims, which provides generous defaults for any character elongations, and ForceBufDims, which puts things more under the application's control.

When QDStartUp is called, it creates a text buffer that is twice as high as the system font, wide enough to support the maxWidth parameter of QDStartUp, and capable of handling characters twice as wide as the system font characters ("wide" in the sense of forExtent). It also permits the use, with any font, of any chExtra <= forExtent (of that font); spExtra <= forExtent; and it allocates up to 36 extra pixels per character to accommodate style modifications (bolding, for example, adds 1 pixel to a character, and italicizing a large font can stretch its horizontal extent significantly). If your application is going to deal only with fonts and text display parameters that fall within those limits, you can trust to the defaults and never call SetBufDims or ForceBufDims.

The SetBufDims routine takes three parameters:

- maxWidth: INTEGER
- maxFontHeight: INTEGER
- maxFBRExtent: INTEGER
The `maxWidth` value is the width in bytes (not pixels) of the largest pixel image the application will draw into (a value of 0 indicates screen width). It will override the value supplied to QDStartUp. The `maxFontHeight` value is the height, in pixels, of the tallest font the application will have to work with. The `maxFBRExtent` value is the `fbExtent` of the widest (that is, greatest `fbExtent`) font the application will work with. The call resizes the clip buffer and the text buffer to accommodate these sizes.

In addition, SetBufDims pads the text buffer to allow for (1) values of `chExtra` and `spExtra` ≤ the `fbExtent` of the font in use at any given time and (2) an extra 36 pixels of style modification added to the width of any character.

SetBufDims's three parameters are used to size QuickDraw II's internal buffers. When it's time to actually draw a string, and QuickDraw II must decide which characters might make it into the destination, it uses the `fbExtent` of the current font (which may be way smaller than `maxFBRExtent`), the current values of `spExtra`, `chExtra`, `txtFace`, and so on, and for a destination pixel image width, the width of the active portion of the current GrafPort's pixel image (its `minRect`, to be specific). Therefore, large values for SetBufDims's parameters may soak up some memory for the text buffer size but will not cost much in time lost drawing into the text buffer characters that will never make it into the destination. This also means that, once the text buffer is sized, the `maxFBRExtent` parameter can be forgotten. (This is not true for ForceBufDims.)

ForceBufDims takes the same parameters as SetBufDims and performs the same functions; however, it does not pad the text buffer at all. Any extra pixels that might be added to a character bounds width due to `chExtra`, `spExtra`, style modifications, or whatever, should be added into the `maxFBRExtent` parameter by the application making the call.

ForceBufDims, like SetBufDims, sizes the buffer(s) on the basis of its parameters, and when a string is actually drawn, only the width of the current GrafPort's pixel image is considered, not all of `maxWidth`. But, unlike SetBufDims, ForceBufDims forces QuickDraw II to use the `maxFBRExtent` parameter to decide which characters are in and which out, rather than trying to calculate a current `fbExtent` value. ForceBufDims is for those times when you're going to do something so unusual that QuickDraw II won't be able to anticipate your actions (such as using very large `chExtra` or `spExtra` values). Consequently, when ForceBufDims is called, its `maxFBRExtent` value must be remembered for subsequent drawing calls. In the SaveBufDims and RestoreBufDims, there is an asymmetry in the parameters handed back, depending on whether the text buffer was originally set (`maxFBRExtent` no longer needed) or forced (`maxFBRExtent` must be remembered). Precise calculations of `maxFBRExtent` for the ForceBufDims call are not necessary; upper limits will do.
You can of course call SetBufDims or ForceBufDims every time you change fonts or even every time you call SetCharExtra and SetTextFace. This is not recommended, however, because sizing (and clearing) buffers can be quite time-consuming. The routines should probably be called only once (if at all), with the maximum realistic values for each of the parameters, and never again.

InflateTextBuffer takes two parameters: newWidth, a font width (that is, fbRExtent in pixels), and newHeight, a font height (in pixels). It then calculates whether the current dimensions of the text buffer are large enough to accommodate a font with that width and height and, if they are not, the routine enlarges the text buffer so that it can handle fonts of that size. The routine will never shrink the size of the text buffer. InflateTextBuffer always pads the value of newWidth to allow for style modifications and for reasonable values of chExtra and spExtra.

† Note: If the current text buffer size was set by a SetBufDims call, then when InflateTextBuffer enlarges the text buffer, it makes an internal call to SetBufDims, so the new width is also considered a "set" value. However, if the text buffer currently has a "forced" width, as set by ForceBufDims, InflateTextBuffer will enlarge the buffer by first padding the newWidth value and then calling ForceBufDims with this padded value as the "forced" font width.

SaveBufDims and RestoreBufDims are included for orderly context-switching between subprograms. SaveBufDims saves the state of the clip buffer and text buffer sizes in the form of an eight-byte record:

maxWidth INTEGER
textBufHeight INTEGER
textBufferWords INTEGER
fontWidth INTEGER

The maxWidth value is the current value of the application-set maximum pixel image width in bytes. The textBufHeight value is the current text buffer height in pixels, and textBufferWords is the current width of the text buffer in words. The fontWidth value serves two purposes: if it is 0, it means the buffer was set up with a call to SetBufDims; if it is nonzero, the buffer was set up with a call to ForceBufDims, and the value of fontWidth is equal to the maxFBRExtent parameter used in that call.

RestoreBufDims restores the buffer dimensions on the basis of the record it is given.

Regardless of which call—QDStartUp, SetBufDims, ForceBufDims, or RestoreBufDims—sizes or resizes the text buffer, the application is not responsible for clearing it. The calls take care of clearing the text buffer automatically. Also note that SaveBufDims and RestoreBufDims do not save and restore the contents of the text buffer, they restore only the parameters related to its size.
Font information calls

GetFontInfo, GetFontGlobals, GetFGSize, GetFontLore: These calls are included for gathering information on the current font. GetFontInfo returns information in the following font info record:

- ascent: INTEGER
- descent: INTEGER
- widMax: INTEGER
- leading: INTEGER

These values have been modified, if necessary, to reflect style modifications currently in effect.

GetFontGlobals returns a variable-length font globals record as follows:

- fgFontID: INTEGER
- fgStyle: INTEGER
- fgSize: INTEGER
- fgVersion: INTEGER
- fgWidMax: INTEGER
- fgFbrExtent: INTEGER

(Additional fields may be present at the end of the record.)

The fgWidMax value is taken from the embedded Macintosh font; all the others are from the Apple II GS header. They are taken directly from the font and are not modified, regardless of any style modifications in effect.

In the future, more information will probably be added to the font globals record. The GetFGSize routine exists to warn the application about the added information. The routine returns the length in bytes of the font globals record. Future versions of QuickDraw II may add extra information at the ends of their font globals records, but for compatibility, those versions will maintain the documented fields and ordering of earlier versions.

GetFontLore returns the same information as GetFontGlobals in the same order. However, the application can use GetFontLore and specify the maximum number of bytes the application will accept; GetFontLore returns no more than that many bytes in the form of an initial segment of the font globals record. Thus, the application can avoid calling the Memory Manager to size a buffer on the basis of GetFGSize; instead, it can just receive the font globals information that the application knows how to handle. The GetFontLore routine is recommended; GetFontGlobals is maintained for compatibility reasons.
$0104  QDBootInit

Initializes QuickDraw II; called only by the Tool Locator.

Warning
An application must never make this call.

This routine puts the address of the cursor update routine into the bank $E1 vectors.

Parameters
The stack is not affected by this call. There are no input or output parameters.

Errors
None

C
Call must not be made by an application.
$0204  **QDStartUp**

Starts up QuickDraw II for use by an application.

---

**Important**

Your application must make this call before it makes any other QuickDraw II calls.

---

The routine sets the current port to the standard port and clears the screen.

QuickDraw II uses three consecutive pages of bank zero for its direct page space, starting at dPageAddr. The maxWidth parameter specifies the size in bytes of the largest pixel map that will be drawn to (a value of 0 indicates screen width). Knowing this maxWidth allows QuickDraw II to allocate certain buffers only once and keep them throughout the life of the application.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>dPageAddr</td>
</tr>
<tr>
<td>masterSCB</td>
</tr>
<tr>
<td>maxWidth</td>
</tr>
<tr>
<td>userID</td>
</tr>
</tbody>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

|$0401  alreadyInitialized |
| Attempt made to start up QuickDraw II a second time without first shutting it down |

|$0410  screenReserved   |
| Memory Manager reported screen memory (bank $E1 from $2000 to $9FFF) is already owned by someone else |

Memory Manager errors

Retumed unchanged

16-64  QuickDraw II housekeeping routines
extern pascal void QDStartUp(dPageAddr, masterSCB, maxWidth, userID)

Word dPageAddr;
Word masterSCB;
Word maxWidth;
Word userID;
$0304 QDShutDown
Shuts down QuickDraw II when an application quits.

Important
If your application has started up QuickDraw II, the application must make this call before it quits.

Parameters
The stack is not affected by this call. There are no input or output parameters.

Errors
Memory Manager errors Returned unchanged

C
eextern pascal void QDShutDown()

$0404 QDVersion
Returns the version number of QuickDraw II.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
</tbody>
</table>

Word—Space for result

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>versionInfo</td>
</tr>
</tbody>
</table>

Word—Version number of QuickDraw II

← SP

Errors
None

C
eextern pascal Word QDVersion()

16-66 QuickDraw II housekeeping routines
$0504  **QDReset**
Resets QuickDraw II; called only when the system is reset.

---

**Warning**
An application must never make this call.

---

**Parameters**
The stack is not affected by this call. There are no input or output parameters.

**Errors**
None

C
Call must not be made by an application.

---

$0604  **QDStatus**
Indicates whether QuickDraw II is active.

---

**Parameters**

---

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>wordspace</th>
</tr>
</thead>
</table>

Word—Space for result
← SP

---

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>activeFlag</th>
</tr>
</thead>
</table>

Word—BOOLEAN; TRUE if QuickDraw II active, FALSE if inactive
← SP

---

**Errors**
None

C
extern pascal Boolean QDStatus()

---

QuickDraw II housekeeping routines 16-67
AddPt

Adds two specified points together and leaves the result in the destination point. For example, two source points of (10,20) and (1,2) result in a destination point of (11,22).

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcPtPtr</td>
</tr>
<tr>
<td>destPtPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to POINT

Long—POINTER to POINT used as source and destination

← SP

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

← SP

**Errors**

None

C

extern pascal void AddPt(srcPtPtr,destPtPtr)

Point *srcPtPtr;

Point *destPtPtr;

16-68 QuickDraw II routines
$AC04  CharBounds

Places the character bounds rectangle of a specified character into a specified buffer.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>theChar</td>
</tr>
<tr>
<td>resultPtr</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

Errors

None

C

extern pascal void CharBounds(theChar, resultPtr)

Word theChar;

Rect *resultPtr;
CharWidth

Returns the character width, in pixels (pen displacement), of a specified character.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
<tr>
<td>theChar</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Word—Space for result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word—Character to be measured</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>charWidth</td>
</tr>
</tbody>
</table>

| Word—INTEGER; width of character in pixels |

Errors

None

C

extern pascal Integer CharWidth(theChar)
Word theChar;
ClearScreen

Sets the words in screen memory to a specified value. The value is stuffed into each word of screen memory. The colorWord value represents a group of adjacent pixels (4 in 320 mode; 8 in 640 mode). See the section "Drawing in Color" in this chapter. ClearScreen is usually used to clear the screen to a solid color.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>colorWord</th>
<th>Word—Color as offset into current color table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>← SP</th>
</tr>
</thead>
</table>

Errors

None

C

extern pascal void ClearScreen(colorWord)

Word colorWord;
$2604 \textbf{ClipRect}

Changes the clipping region of the current GrafPort to a rectangle that is equivalent to a specified rectangle.

\textbf{Parameters}

\textbf{Stack before call}

\begin{tabular}{|c|c|}
\hline
\textit{previous contents} & \textit{Long}—POINTER to RECT defining rectangle \\
\hline
\textit{rectPtr} & \leftarrow SP \\
\hline
\end{tabular}

\textbf{Stack after call}

\begin{tabular}{|c|c|}
\hline
\textit{previous contents} & \leftarrow SP \\
\hline
\end{tabular}

\textbf{Errors}

Memory Manager errors \hspace{1cm} Returned unchanged

\textbf{C}

\texttt{extern pascal void ClipRect(rectPtr)}

\texttt{Rect *rectPtr;}

\hline

$C204 \textbf{ClosePoly}

Completes the polygon definition process started with an OpenPoly call.

\textbf{Parameters}

The stack is not affected by this call. There are no input or output parameters.

\textbf{Errors}

$0441 \hspace{1cm} \texttt{PolyNotOpen} \hspace{1cm} \text{No polygon open in current GrafPort}$

Memory Manager errors \hspace{1cm} Returned unchanged

\textbf{C}

\texttt{extern pascal void ClosePoly()}

\hline

16-72 \hspace{0.5cm} \textit{QuickDraw II routines}
$1A04 \textbf{ClosePort}

Deallocates the clipping and visible regions in a port. If the application disposes of the memory containing the port without first calling ClosePort, the memory associated with the handles is lost and cannot be reclaimed.

\textbf{Warning}

Never close the current port.

\textbf{Parameters}

\textbf{Stack before call}

\begin{verbatim}
previous contents

Long–POINTER to port

← SP
\end{verbatim}

\textbf{Stack after call}

\begin{verbatim}
previous contents

← SP
\end{verbatim}

\textbf{Errors}

Memory Manager errors Returned unchanged

\textbf{C}

extern pascal void ClosePort(portPtr)

GrafPortPtr portPtr;
CloseRgn

Completes the region-definition process started by an OpenRgn call. The region must have already been created by a NewRgn call, which supplies `rgnHandle`.

Parameters

Stack before call

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>rgnHandle</td>
</tr>
</tbody>
</table>
```

Long—HANDLE to region being set to a collection of points

Stack after call

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>SP</td>
</tr>
</tbody>
</table>
```

Errors

- $0431 rgnNotOpen: No region open in current GrafPort
- Memory Manager errors: Returned unchanged

C

```
extern pascal void CloseRgn(rgnHandle)
RgnHandle rgnHandle;
```
CopyRgn

Copies the region definition from one region to another. The \textit{srcRgnHandle} and \textit{destRgnHandle} must have already been created; in particular, this routine does not allocate the \textit{destRgnHandle}.

Parameters

Stack before call

\begin{tabular}{|c|}
\hline
\textit{previous contents}  \\
\hline
\textit{srcRgnHandle}  \\
\hline
\textit{destRgnHandle}  \\
\hline
\end{tabular}

- \textit{srcRgnHandle}  \hspace{1cm} \textbf{Long-HANDLE to source region}
- \textit{destRgnHandle}  \hspace{1cm} \textbf{Long-HANDLE to destination region}

Stack after call

\begin{tabular}{|c|}
\hline
\textit{previous contents}  \\
\hline
\end{tabular}

\textbf{← SP}

Errors

- Memory Manager errors  \hspace{1cm} Returned unchanged

C

\begin{verbatim}
extern pascal void CopyRgn(srcRgnHandle, destRgnHandle)
RgnHandle srcRgnHandle;
RgnHandle destRgnHandle;
\end{verbatim}
$AE04 CStringBounds

Places the character bounds rectangle of a specified C string into a specified buffer.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>cStringPtr</td>
</tr>
<tr>
<td>resultPtr</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
</tr>
</tbody>
</table>

Errors

None

C

extern pascal void CStringBounds(cStringPtr, resultPtr)

Pointer cStringPtr;
Rect *resultPtr;
**$AA04 CStringWidth**

Returns the sum of all the character widths, in pixels (pen displacements), in a specified C string. This would be the pen displacement if the string were to be drawn.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Word—Space for result</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
<td></td>
</tr>
<tr>
<td>cStringPtr</td>
<td>Long—POINTER to C string</td>
</tr>
</tbody>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Word—INTEGER; width of C string in pixels</th>
</tr>
</thead>
</table>
| cStringWidth      | SP                                      

**Errors**

None

**C**

```c
extern pascal Integer CStringWidth(cStringPtr)
Pointer cStringPtr;
```
Calculates the difference of the areas enclosed by two regions and places the region definition of the enclosing area in a destination region. The destination region, which may be one of the source regions, must already exist; this routine does not allocate it.

If the `rgn1Handle` is empty, the destination is set to an empty region.

### Parameters

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rgn1Handle</code></td>
</tr>
<tr>
<td><code>rgn2Handle</code></td>
</tr>
<tr>
<td><code>diffRgnHandle</code></td>
</tr>
</tbody>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

Memory Manager errors Returned unchanged

```c
extern pascal void DiffRgn(rgn1Handle, rgn2Handle, diffRgnHandle)
RgnHandle rgn1Handle;
RgnHandle rgn2Handle;
RgnHandle diffRgnHandle;
```
$6804 \textbf{DisposeRgn}

Deallocates the memory for a specified region. For more information about how memory is allocated and deallocated, see Chapter 12, "Memory Manager," in Volume 1.

\textbf{Parameters}

\textbf{Stack before call}

\begin{itemize}
  \item \textit{previous contents}
  \item \textit{rgnHandle} \texttt{Long}—HANDLE of region being disposed
  \item \texttt{SP} \leftarrow
\end{itemize}

\textbf{Stack after call}

\begin{itemize}
  \item \textit{previous contents}
  \item \texttt{SP} \leftarrow
\end{itemize}

\textbf{Errors}

Memory Manager errors Returned unchanged

\textbf{C}

\begin{verbatim}
extern pascal void DisposeRgn(rgnHandle)
RgnHandle rgnHandle;
\end{verbatim}
**$A404 DrawChar**

Draws a specified character at the current pen location and updates the pen location.

**Parameters**

**Stack before call**

```
previous contents
theChar
```

**Word**—ASCII code of character to be drawn (0–255)

← SP

**Stack after call**

```
previous contents
```

← SP

**Errors**

None

**C**

```c
extern pascal void DrawChar(theChar);
Word theChar;
```
$A604 \textbf{DrawCString} \\

Draws a specified C string at the current pen location and updates the pen location.

\textbf{Parameters}

\textbf{Stack before call}

\begin{align*}
\text{previous contents} & \\
\downarrow \text{CStringPtr} & \quad \text{Long—POINTER to C string} \\
\downarrow & \quad \leftarrow \text{SP}
\end{align*}

\textbf{Stack after call}

\begin{align*}
\text{previous contents} & \\
\downarrow & \quad \leftarrow \text{SP}
\end{align*}

\textbf{Errors} \quad \text{None}

\textbf{C}

\begin{Verbatim}
\texttt{extern pascal void DrawCString(cStringPtr)}
\texttt{Pointer cStringPtr;}
\end{Verbatim}
$A504  DrawString

Draws a specified Pascal-type string at the current pen location and updates the pen location.

Parameters

Stack before call

```
previous contents
--- stringPtr ---
```

Long—POINTER to Pascal-type string to be drawn

← SP

Stack after call

```
previous contents
← SP
```

Errors

None

C

```c
extern pascal void DrawString(stringPtr)

Pointer stringPtr;
```
DrawText

Draws specified text at the current pen location and updates the pen location.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>textPtr</td>
</tr>
<tr>
<td>textLength</td>
</tr>
</tbody>
</table>

- **Long**—POINTER to text to be drawn
- **Word**—INTEGER; number of bytes in text to be drawn

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

Errors

None

C

```
extern pascal void DrawText(textPtr, textLength)

Pointer     textPtr;
Word        textLength;
```
$7804 \textbf{EmptyRgn}

Indicates whether a specified region is empty.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{wordspace}</td>
</tr>
<tr>
<td>\textit{rgnHandle}</td>
</tr>
</tbody>
</table>

| ← SP |

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{emptyFlag}</td>
</tr>
</tbody>
</table>

| ← SP |

**Errors**

None

**C**

\begin{verbatim}
extern pascal Boolean EmptyRgn(rgnHandle)
RgnHandle  rgnHandle;
\end{verbatim}
EqualPt

Indicates whether two points are equal (two equal points have the same Y and X coordinates).

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>wordspace</th>
<th>Word—Space for result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- point1Ptr</td>
<td></td>
<td>Long—POINTER to first POINT</td>
</tr>
<tr>
<td>-- point2Ptr</td>
<td></td>
<td>Long—POINTER to second POINT</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>equalFlag</th>
<th>Word—BOOLEAN; TRUE if points are equal, FALSE if not</th>
</tr>
</thead>
</table>

Errors

None

C

extern pascal Boolean EqualPt(point1Ptr, point2Ptr)

Point *point1Ptr;
Point *point2Ptr;
EqualRect

Indicates whether two rectangles are equal. The two rectangles must have identical sizes and locations to be considered equal. Any two empty rectangles are always equal.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
<tr>
<td>rect1Ptr</td>
</tr>
<tr>
<td>rect2Ptr</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack after call</th>
</tr>
</thead>
<tbody>
<tr>
<td>equalFlag</td>
</tr>
</tbody>
</table>

Errors

None

C

```c
extern pascal Boolean EqualRect(rect1Ptr, rect2Ptr)
Rect *rect1Ptr;
Rect *rect2Ptr;
```
EqualRgn

Indicates whether two regions are equal. The two regions must have identical sizes, shapes, and locations to be considered equal. Any two empty regions are always equal.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
<tr>
<td>rgn1Handle</td>
</tr>
<tr>
<td>rgn2Handle</td>
</tr>
<tr>
<td>Word—Space for result</td>
</tr>
<tr>
<td>Long—HANDLE to one region</td>
</tr>
<tr>
<td>Long—HANDLE to other region</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>equalFlag</td>
</tr>
<tr>
<td>Word—BOOLEAN; TRUE if regions are equal, FALSE if not</td>
</tr>
</tbody>
</table>

Errors

Memory Manager errors Returned unchanged

C

extern pascal Boolean EqualRgn(rgn1Handle, rgn2Handle)

RgnHandle rgn1Handle;
RgnHandle rgn2Handle;
EraseArc

Erases the interior of a specified arc by filling it with the background pattern.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rectPtr</td>
</tr>
<tr>
<td>startAngle</td>
</tr>
<tr>
<td>arcAngle</td>
</tr>
</tbody>
</table>

- **rectPtr** — Long—POINTER to RECT defining enclosing rectangle
- **startAngle** — Word—INTEGER; starting angle in degrees
- **arcAngle** — Word—INTEGER; arc angle in degrees

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

Errors

None

C

```c
extern pascal void EraseArc(rectPtr, startAngle, arcAngle)

Rect *rectPtr;
Integer startAngle;
Integer arcAngle;
```
EraseOval

Erasers the interior of a specified oval by filling it with the background pattern.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rectPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to RECT defining enclosing rectangle

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

Errors

None

C

extern pascal void EraseOval(rectPtr)

Rect *rectPtr;
ErasePoly

Erasers the interior of a specified polygon by filling it with the background pattern. Because polygons are treated differently than other closed shapes, the frame of the polygon (if drawn) is not completely erased. See the section “Polygons” in this chapter for more information.

Important
Because this call allocates and deallocates some temporary memory space, Memory Manager errors can occur.

Parameters

Stack before call

previous contents

polyHandle

Long—HANDLE to polygon

← SP

Stack after call

previous contents

← SP

Errors

Memory Manager errors Returned unchanged

C

extern pascal void ErasePoly(polyHandle)

Handle polyHandle;
$5504 \textbf{EraseRect} \newline Erases the interior of a specified rectangle by filling it with the background pattern.

\textbf{Parameters}

\textbf{Stack before call}

\begin{center}
\begin{tabular}{|c|}
\hline
\textit{previous contents} \\
\hline
\hline
\textit{rectPtr} \\
\hline
\leftarrow \text{SP} \\
\hline
\end{tabular}
\end{center}

Long—POINTER to RECT defining rectangle

\textbf{Stack after call}

\begin{center}
\begin{tabular}{|c|}
\hline
\textit{previous contents} \\
\hline
\hline
\leftarrow \text{SP} \\
\hline
\end{tabular}
\end{center}

\textbf{Errors}

None

\begin{verbatim}
C
extern pascal void EraseRect(rectPtr)
    Rect *rectPtr;
\end{verbatim}

\textbf{QuickDraw II routines} 16-91
$7B04 \textbf{EraseRgn}

Erases the interior of a specified region by filling it with the background pattern.

**Parameters**

**Stack before call**

```
previous contents

-- rgnHandle --

Long-HANDLE to region

<- SP
```

**Stack after call**

```
previous contents

<- SP
```

**Errors**

None

**C**

```c
extern pascal void EraseRgn(rgnHandle)
RgnHandle rgnHandle;
```
EraseRRect

Erasers the interior of a specified round rectangle by filling it with the background pattern.

The corners of the round rectangle are sections of an oval defined by ovalHeight and ovalWidth. For more information, see Figure 16-12 in the section "Rectangles" in this chapter.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rectPtr</td>
</tr>
<tr>
<td>ovalWidth</td>
</tr>
<tr>
<td>ovalHeight</td>
</tr>
</tbody>
</table>

Long—POINTER to RECT defining enclosing rectangle
Word—INTEGER; width, in pixels, of oval defining rounded corners
Word—INTEGER; height, in pixels, of oval defining rounded corners

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>
| SP

Errors

None

C

extern pascal void EraseRRect(rectPtr, ovalWidth, ovalHeight)

Rect *rectPtr;
Word   ovalWidth;
Word   ovalHeight;
$6604 \text { FillArc}

Fills the interior of a specified arc with a specified pen pattern.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rectPtr</td>
</tr>
<tr>
<td>startAngle</td>
</tr>
<tr>
<td>arcAngle</td>
</tr>
<tr>
<td>patternPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to RECT defining rectangle

Word—INTEGER; starting angle in degrees

Word—INTEGER; arc angle in degrees

Long—POINTER to pattern

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Errors

None

C

extern pascal void FillArc(rectPtr, startAngle, arcAngle, patternPtr)

Rect *rectPtr;

Integer startAngle;

Integer arcAngle;

Pattern patternPtr;
$5C04 \textbf{ FillOval}

Fills the interior of a specified oval with a specified pen pattern.

\textbf{Parameters}

\textbf{Stack before call}

\begin{center}
\begin{tabular}{|c|}
\hline
\textit{previous contents}  \\
\hline
\textit{rectPtr}  \\
\hline
\textit{patternPtr}  \\
\hline
\end{tabular}
\end{center}

\begin{itemize}
\item \textbf{Long}—POINTER to \texttt{RECT} defining enclosing rectangle
\item \textbf{Long}—POINTER to pattern
\end{itemize}

\textbf{Stack after call}

\begin{center}
\begin{tabular}{|c|}
\hline
\textit{previous contents}  \\
\hline
\end{tabular}
\end{center}

\begin{itemize}
\item \textbf{SP}
\end{itemize}

\textbf{Errors}

None

\textbf{C}

\begin{verbatim}
extern pascal void FillOval(rectPtr, patternPtr)
Rect *rectPtr;
Pattern patternPtr;
\end{verbatim}
$C004  

**FillPoly**

Fills the interior of a specified polygon with a specified pen pattern. Because polygons are treated differently than other closed shapes, the frame of the polygon (if drawn) is not completely filled. See the section "Polygons" in this chapter for more information.

---

**Important**

Because this call allocates and deallocates some temporary memory space, Memory Manager errors can occur.

---

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>polyHandle</td>
</tr>
<tr>
<td>patternPtr</td>
</tr>
</tbody>
</table>

Long—HANDLE to polygon

Long—POINTER to pattern

← SP

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

← SP

**Errors**

Memory Manager errors Returned unchanged

**C**

```c
extern pascal void FillPoly(polyHandle, patternPtr)
Handle    polyHandle;
Pattern    patternPtr;
```
FillRect

Fills the interior of a specified rectangle with a specified pen pattern.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rectPtr</td>
</tr>
<tr>
<td>patternPtr</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Errors

None

C

```c
extern pascal void FillRect(rectPtr, patternPtr)
Rect *rectPtr;
Pattern patternPtr;
```
$7D04 \textbf{FillRgn}

Fills the interior of a specified region with a specified pen pattern.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{rgnHandle} \quad \textit{patternPtr}</td>
</tr>
</tbody>
</table>

\textbf{Long—HANDLE} to region

\textbf{Long—POINTER} to pattern

\textbf{SP}

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{SP}</td>
</tr>
</tbody>
</table>

**Errors**

None

**C**

\begin{verbatim}
extern pascal void FillRgn(rgnHandle, patternPtr)
RgnHandle rgnHandle;
Pattern patternPtr;
\end{verbatim}
$6104 \textbf{FillRRect}

Fills the interior of a specified round rectangle with a specified pen pattern.

The corners of the round rectangle are sections of an oval defined by $\texttt{ovalHeight}$ and $\texttt{ovalWidth}$. For more information, see Figure 16-12 in the section "Rectangles" in this chapter.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- rectPtr</td>
</tr>
<tr>
<td>ovalWidth</td>
</tr>
<tr>
<td>ovalHeight</td>
</tr>
</tbody>
</table>
| -- patternPtr     | Long—POINTER to RECT defining enclosing rectangle
|                   | Word—INTEGER; width, in pixels, of oval defining rounded corners
|                   | Word—INTEGER; height, in pixels, of oval defining rounded corners
|                   | Long—POINTER to pattern

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

None

**C**

```c
extern pascal void FillRRect(rectPtr,ovalWidth,ovalHeight,patternPtr)

Rect *rectPtr;
Word ovalWidth;
Word ovalHeight;
Pattern patternPtr;
```
**ForceBufDims**

Sets the size of the QuickDraw II clipping and text buffers, but does not pad
maxFBRExtent in any way. The maxFBRExtent value must include not only the
greatest width of the character but also any extra width necessary because of style
modifications, chExtra, and spExtra.

*Note:* You need to make this call only if your application is going to use, or allow
the user to choose, fonts that have unusually large values of chExtra and spExtra.
See the section "Fonts and Text in QuickDraw II" in this chapter for more
information.

Although SetBufDims and ForceBufDims may be called at any time, it is usually best
to call them once with reasonable maximum values early in the application (if at all),
because claiming and clearing a buffer can take lots of time.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxWidth</td>
</tr>
<tr>
<td>maxFontHeight</td>
</tr>
<tr>
<td>maxFBRExtent</td>
</tr>
</tbody>
</table>

Word—INTEGER; width, in bytes, of widest pixel map to be used
Word—INTEGER; height, in pixels, of tallest font the application will use
Word—INTEGER; greatest fbrExtent in pixels of any font to be used

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

← SP

**Errors**

Memory Manager errors Returned unchanged

**C**

extern pascal void ForceBufDims(maxWidth,maxFontHeight,maxFBRExtent)

Word maxWidth;
Word maxFontHeight;
Word maxFBRExtent;
FrameArc

Draws the frame of a specified arc using the current pen mode, pen pattern, and pen size. Only pixels entirely within the rectangle are affected.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rectPtr</td>
</tr>
<tr>
<td>startAngle</td>
</tr>
<tr>
<td>arcAngle</td>
</tr>
</tbody>
</table>

- **Long**—POINTER to RECT defining enclosing rectangle
- **Word**—INTEGER; starting angle in degrees
- **Word**—INTEGER; arc angle in degrees

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
</tr>
</tbody>
</table>

Errors

None

C

```
extern pascal void FrameArc(rectPtr, startAngle, arcAngle)

Rect *rectPtr;
Integer startAngle;
Integer arcAngle;
```
$5804  **FrameOval**

Draws the frame of a specified oval using the current pen mode, pen pattern, and pen size. Only pixels entirely within the rectangle are affected.

---

**Important**

If a region is open, this command contributes to the region definition; this can cause Memory Manager errors to occur.

---

**Parameters**

**Stack before call**

| previous contents |
| rectPtr — Long—POINTER to RECT defining enclosing rectangle |

**Stack after call**

| previous contents |

**Errors**

- Memory Manager errors
- Returned unchanged

**C**

```c
extern pascal void FrameOval(rectPtr)
Rect *rectPtr;
```
FramePoly

Draws the frame of a specified polygon using the current pen mode, pen pattern, and pen size. The polygon is framed with a series of LineTo calls.

Important
If this call is used, parts of the frame are not affected by ErasePoly and FillPoly calls. You can, for example, erase the frame by using another FramePoly call using the background pattern. In addition, if a region is open, this command contributes to the region definition; this can cause Memory Manager errors to occur.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>polyHandle</td>
</tr>
</tbody>
</table>

Long—HANDLE to polygon

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

← SP

Errors

Memory Manager errors    Returned unchanged

C

extern pascal void FramePoly(polyHandle)

Handle polyHandle;
FrameRect

Draws the frame of a specified rectangle using the current pen mode, pen pattern, and pen size. Only pixels entirely within the rectangle are affected.

Important
If a region is open, this command contributes to the region definition; this can cause Memory Manager errors to occur.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rectPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to RECT defining rectangle

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Errors

Memory Manager errors Returned unchanged

C

extern pascal void FrameRect(rectPtr)

Rect *rectPtr;
FrameRgn

Draws the frame of a specified region using the current pen mode, pen pattern, and pen size. Only pixels entirely inside the region are affected.

If a region is open and being formed, the outline of the region being framed is added to the open region’s boundary.

---

**Important**

If a region is open, this command contributes to the region definition; this can cause Memory Manager errors to occur.

---

**Parameters**

**Stack before call**

```
<table>
<thead>
<tr>
<th>Previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgnHandle</td>
</tr>
<tr>
<td>Long—HANDLE to region</td>
</tr>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>
```

**Stack after call**

```
| Previous contents |
| ← SP              |
```

**Errors**

Memory Manager errors Returned unchanged

**C**

```c
extern pascal void FrameRgn(rgnHandle)
RgnHandle  rgnHandle;
```
FrameRRect

Draws the frame of a specified round rectangle using the current pen mode, pen pattern, and pen size. Only pixels entirely within the rectangle are affected.

Important
If a region is open, this command contributes to the region definition; this can cause Memory Manager errors to occur.

The corners of the round rectangle are sections of an oval defined by ovalHeight and ovalWidth. For more information, see Figure 16-12 in the section "Rectangles" in this chapter.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- rectPtr --</td>
</tr>
<tr>
<td>ovalWidth</td>
</tr>
<tr>
<td>ovalHeight</td>
</tr>
</tbody>
</table>

| Long—POINTER to RECT defining enclosing rectangle |
| Word—INTEGER; width, in pixels, of oval defining rounded corners |
| Word—INTEGER; height, in pixels, of oval defining rounded corners |

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Errors

Memory Manager errors Returned unchanged

C

extern pascal void FrameRRect(rectPtr, ovalWidth, ovalHeight)

Rect *rectPtr;
Word ovalWidth;
Word ovalHeight;

16-106 QuickDraw II routines
GetAddress

Returns a pointer to a specified table. QuickDraw II contains a number of tables that may be useful. The GetAddress call allows you to access these tables.

The current tableIDs are as follows:

$0001 screenTable
$0002 conTable320
$0003 conTable640

Important
If your application is using one of these tables, make sure the application obtains the correct pointer every time it runs. These tables will move as the ROM version changes.

The screen table has 200 two-byte entries. Each entry is the address of the start of a scan line in the display buffer. The zeroth entry is $2000, which is the address of scan line 0; entry 1 is $20A0, which is the address of scan line 1; and so on.

The conTable320 and conTable640 tables are used to convert from bytes that are one bit per pixel to bytes that are four and two bits per pixel respectively. The conTable320 table has 256 four-byte entries; the conTable640 table has 256 two-byte entries. These entries are the two- and four-bit-per-pixel representations of one-bit-per-pixel bytes.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
<tr>
<td>tableID</td>
</tr>
</tbody>
</table>

Long—Space for result

Word—INTEGER; ID of table whose pointer will be retrieved

←SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablePtr</td>
</tr>
</tbody>
</table>

Long—POINTER to table in ROM

←SP

QuickDraw II routines 16-107
extern pascal Pointer GetAddress(tableID)
Word tableID;

**Assembly-language example**

The byte containing $37$ appears as follows in one-, two-, and four-bit-per-pixel mode:

<table>
<thead>
<tr>
<th>One-bit</th>
<th>Two-bit</th>
<th>Four-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>%00110111</td>
<td>%00 00 11 11 00 11 11 11</td>
<td>$00FF OFFF</td>
</tr>
</tbody>
</table>

The two- and four-bit versions would be obtained from the table as follows:

- **One-bit**
  - `lda OneBit`
  - `and #$00FF`
  - `asl a`
  - `tay`

- **Two-bit**
  - `lda OneBit`
  - `and #$00FF`
  - `asl a`
  - `asl a`
  - `tay`

- **Four-bit**
  - `lda [TwoBitTable], y`
  - `tay`
  - `lda [FourBitTable], y`
  - `tax`
  - `iny`

In both cases, the addresses obtained from GetAddress are already on the direct page.
GetArcRot

Returns the value of the arcRot field in the current GrafPort.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
</tbody>
</table>

Word—Space for result
← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>arcRot</td>
</tr>
</tbody>
</table>

Word—INTEGER; value of arcRot field
← SP

Errors

None

C

extern pascal Integer GetArcRot()
GetBackColor

Returns the value of the bgcolor field from the GrafPort.

Parameters

Stack before call

\[
\begin{array}{c|c}
\text{previous contents} & \text{Word—Space for result} \\
\text{workspace} & \leftarrow \text{SP}
\end{array}
\]

Stack after call

\[
\begin{array}{c|c}
\text{previous contents} & \text{Word—INTEGER; value of bgcolor field} \\
\text{backColor} & \leftarrow \text{SP}
\end{array}
\]

Errors

None

C

\text{extern pascal Word GetBackColor()}

16-110 QuickDraw II routines
GetBackPat

Copies the current background pen pattern from the current GrafPort to a specified location.

Parameters

Stack before call

\[
\begin{array}{l}
\text{previous contents} \\
\hline
\text{patternPtr} \quad \text{Long—POINTER to location for pattern} \\
\hline
\end{array}
\]

\[\leftarrow \text{SP}\]

Stack after call

\[
\begin{array}{l}
\text{previous contents} \\
\hline
\end{array}
\]

\[\leftarrow \text{SP}\]

Errors

None

C

```c
extern pascal void GetBackPat(patternPtr)
Pattern patternPtr;
```
$D504 GetCharExtra

Returns the chExtra field from the GrafPort.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
</tbody>
</table>

Long—Space for result

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>charExtra</td>
</tr>
</tbody>
</table>

Long—FIXED; value of chExtra field

Errors
None

C
extern pascal Fixed GetCharExtra()
GetClip

Copies the clipping region to a specified region. The destination region must have been created earlier with a NewRgn call.

Parameters

Stack before call

| previous contents |
|-------------------|--------------------------|
| rgnHandle          | Long—HANDLE to region    |

Stack after call

| previous contents |
|-------------------|--------------------------|
|                   | SP                       |

Errors

Memory Manager errors Returned unchanged

C

extern pascal void GetClip(rgnHandle)
RgnHandle rgnHandle;
GetClipHandle

Returns a copy of the handle to the clipping region.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
</tbody>
</table>

Long—Space for result

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgnHandle</td>
</tr>
</tbody>
</table>

Long—HANDLE to clipping region

← SP

Errors

None

C

extern pascal RgnHandle GetClipHandle()
$1104 \text{ GetColorEntry}$

Returns the value of a specified color in a specified color table.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
<tr>
<td>tableNumber</td>
</tr>
<tr>
<td>entryNumber</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Word—Space for result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word—INTEGER; number of color table</td>
</tr>
<tr>
<td>Word—INTEGER; number of color to be examined</td>
</tr>
</tbody>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Word—Color of entry</th>
</tr>
</thead>
</table>

**Errors**

$0450 \text{ badTableNum} \quad \text{Invalid table number; 0 to 15 are valid}$

$0451 \text{ badColorNum} \quad \text{Invalid color number; 0 to 15 are valid}$

**C**

```c
extern pascal Word GetColorEntry(tableNumber, entryNumber)
Word    tableNumber;
Word    entryNumber;
```
$0F04  GetColorTable

Fills a specified color table with the contents of another color table.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>tableNumber</td>
</tr>
<tr>
<td>destTablePtr</td>
</tr>
</tbody>
</table>

- **Word**—INTEGER; color table to be copied
- **Long**—POINTER to color table to receive new values

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

$0450  badTableNum  Invalid table number; 0 to 15 are valid

**C**

```c
extern pascal void GetColorTable(tableNumber, destTablePtr)

Word  tableNumber;
ColorTable  destTablePtr;
```
$8F04  GetCursorAdr

Returns a pointer to the current cursor record. See the section "Cursors" in this chapter for the definition of the cursor record.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>___ longspace ___</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Long—Space for result

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>___ cursorPtr ___</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Long—POINTER to current cursor record

Errors

None

C
extern pascal Pointer GetCursorAdr()
GetFGSize

Returns the size of the font globals record. The font globals record, which contains information about the font, may increase in length in future versions of QuickDraw II. The GetFGSize routine tells your application how much space to allocate for the record.

This call is primarily intended to provide backward compatibility. Under normal circumstances, you'll probably prefer to use the GetFontLore routine. See the section "GetFontLore" in this chapter.

*Note:* The information in the record will only increase. Fields that have been defined will not disappear, but additional fields may be added at the end of the font globals record.

**Parameters**

Stack before call

- `previous contents`
- `wordspace`
- `Word—Space for result ← SP`

Stack after call

- `previous contents`
- `fgSize`
- `Word—INTEGER; size of font globals record ← SP`

**Errors**

- None

**C**

```
extern pascal Word GetFGSize();
```
$\text{GetFont}$

Returns a handle to the current font.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- longspace --</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- fontHandle --</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Errors**

None

**C**

```
extern pascal FontHndl GetFont()
```
GetFontFlags

Returns the current font flags word. See Figure 16-37 in the section "SetFontFlags" in this chapter for the possible fontFlags values.

Parameters

Stack before call

| previous contents | wordspace | Word—Space for result | ← SP |

Stack after call

| previous contents | fontFlags | Word—INTEGER; current font flags word (see Figure 16-37) | ← SP |

Errors

None

C

extern pascal Word GetFontFlags()
Returns information about the font globals record into a specified buffer. The size of the font globals record is returned by the GetFGSize routine, as described in the section "GetFGSize" in this chapter. The information represents the GrafPort's current font and does not reflect style modifications, chExtra or spExtra fields, and so on. Future versions of QuickDraw II may add more information at the end of this record, but the current fields and their order will be maintained.

This call is primarily intended to provide backward compatibility. Under normal circumstances, you'll probably prefer to use the GetFontLore routine. See the section "GetFontLore" in this chapter.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$fgRecPtr</td>
</tr>
<tr>
<td>Long—POINTER to space for font globals record</td>
</tr>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

None

**C**

```c
extern pascal void GetFontGlobals($fgRecPtr)
FontGlobalsRecPtr $fgRecPtr;
```
$D104 \textbf{GetFontID}

Returns the \textit{fontID} field of the GrafPort.

\textbf{Parameters}

\textbf{Stack before call}

\begin{itemize}
\item \textit{previous contents}
\item \textit{longspace}
\item \textit{Long}—Space for result \leftarrow SP
\end{itemize}

\textbf{Stack after call}

\begin{itemize}
\item \textit{previous contents}
\item \textit{fontID}
\item \textit{Long}—Value of \textit{fontID} field in GrafPort \leftarrow SP
\end{itemize}

\textbf{Errors} \hspace{1cm} \textbf{None}

\begin{verbatim}
C
 extern pascal LongWord GetFontID()

\textbullet \textit{Note:} C Pascal-type functions do not deal properly with data structures returned on the stack. The Long result returned by this call can be passed to any calls requiring a font ID as a parameter. You cannot use the C dot operator to access the individual font ID fields within the value returned by this call.
\end{verbatim}
$9604 \textbf{GetFontInfo}

Returns information about the current font in a specified buffer. The information in the \textit{fontinfo} record does reflect current style modifications, but not the values of the \textit{chExtra} and \textit{spExtra} fields of the GrafPort. See the section "Font Information Calls" in this chapter for the definition of the font info record.

Your application can use the information returned in the \textit{fontinfo} record to determine the spacing between lines of text. Normal spacing is ascent plus descent plus leading.

\textbf{Parameters}

\textbf{Stack before call}

\begin{itemize}
\item previous contents
\item \textit{fontinfoRecPtr} \hspace{2cm} Long—POINTER to space for \textit{fontinfo} record
\end{itemize}

\textbf{Stack after call}

\begin{itemize}
\item previous contents
\item \hspace{1cm} \leftarrow SP
\end{itemize}

\textbf{Errors}

None

\textbf{C}

\begin{verbatim}
extern pascal void GetFontInfo(fontinfoRecPtr)
FontInfoRecPtr fontInfoRecPtr;
\end{verbatim}
$D904 \textbf{GetFontLore} \\

Returns information, up to a specified number of bytes, about the current font in a specified buffer. The routine returns the same values as the GetFontGlobals call (see the section "Font Information Calls" in this chapter), except that GetFontLore will not return more bytes than are specified in the recordSize parameter. Thus, you can set aside a fixed amount of space for the record.

\underline{Important} \\
This call is available in Version 2.0 or later of QuickDraw II.

You can specify the number of bytes as equal to the maximum number of bytes in the record for a particular version of QuickDraw II. Future versions of QuickDraw II may add more information at the end of this record, but the current fields and their order will be maintained.

The numBytesXfer value may sometimes be less than recordSize—for example, if the recordSize is larger than the number of bytes that GetFontLore has to return.

\underline{Parameters} \\

\underline{Stack before call} \\

\begin{tabular}{|c|c|}
\hline
previous contents & \\
\hline
wordspace & Word—Space for result \\
\hline
recordPtr & Long—POINTER to space for record \\
\hline
recordSize & Word—Maximum number of bytes to transfer \\
\hline
\end{tabular}

← $P$

\underline{Stack after call} \\

\begin{tabular}{|c|c|}
\hline
previous contents & \\
\hline
numBytesXfer & Word—INTEGER; number of bytes transferred \\
\hline
\end{tabular}

← $P$

\underline{Errors} \\
None

\underline{C} \\

\begin{verbatim}
extern pascal Word GetFontLore(recordPtr,recordSize)

FontGlobalsRecPtr recordPtr;
Word recordSize;
\end{verbatim}

16-124 QuickDraw II routines
$A104  GetForeColor

Returns the value of the current $fgColor field from the GrafPort.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
</tbody>
</table>

Word—Space for result

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>foreColor</td>
</tr>
</tbody>
</table>

Word—INTEGER; value of $fgColor field

← SP

Errors

None

C

extern pascal Word GetForeColor()
GetGrafProcs

Returns the pointer to the grafProcs record associated with the GrafPort.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
</tbody>
</table>
| Long—Space for result | ← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>grafProcsPtr</td>
</tr>
</tbody>
</table>
| Long—POINTER to grafProcs record | ← SP

Errors

None

C

extern pascal QDProcsPtr GetGrafProcs()
$1704 \textbf{GetMasterSCB}

Returns a copy of the master SCB.

Parameters

Stack before call

\begin{verbatim}
| previous contents               | previous contents               |
| wordspace                       | wordspace                       |
\end{verbatim}

\textbf{Word}—Space for result

\textbf{← SP}

Stack after call

\begin{verbatim}
| previous contents               | previous contents               |
| masterSCB                       | masterSCB                       |
\end{verbatim}

\textbf{Word}—Master SCB value

\textbf{← SP}

Errors

None

C

\texttt{extern pascal Word GetMasterSCB();}
$2904 \textbf{GetPen}

Returns the pen location.

\textit{Macintosh programmers}: This routine does not pass the point on the stack; instead, it passes a pointer to the point.

\textbf{Parameters}

\textbf{Stack before call}

\begin{align*}
\text{previous contents} & \quad \text{Long—POINTER to POINT} \\
\text{pointPtr} & \quad \leftarrow \text{SP}
\end{align*}

\textbf{Stack after call}

\begin{align*}
\text{previous contents} & \quad \leftarrow \text{SP}
\end{align*}

\textbf{Errors} \quad \text{None}

\textbf{C}

\begin{verbatim}
extern pascal void GetPen(pointPtr)

Point *pointPtr;
\end{verbatim}

16-128 \textbf{QuickDraw II routines}
GetPenMask

Returns the pen mask to a specified location.

Parameters

Stack before call

```
previous contents
-- maskPtr -- Long—POINTER to space for mask
<- SP
```

Stack after call

```
previous contents
---------- <- SP
```

Errors

None

C

```c
extern pascal void GetPenMask(maskPtr);
Mask maskPtr;
```
$2F04 \textbf{GetPenMode} \newline
Returns the pen mode from the current GrafPort. See Table 16-9 in the section "SetPenMode" in this chapter for the pen mode values.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>workspace</th>
</tr>
</thead>
</table>

Word—Space for result


**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>penMode</th>
</tr>
</thead>
</table>

Word—INTEGER; pen mode value, as shown in Table 16-9

**Errors**

None

**C**

extern pascal Word GetPenMode()
$3104 \textbf{GetPenPat}

Copies the current pen pattern from the current GrafPort to a specified location.

**Parameters**

**Stack after call**

\[
\begin{array}{c|c}
\text{previous contents} & \text{Long—POINTER to space for pattern} \\
\hline
\text{patternPtr} & \leftarrow \text{SP}
\end{array}
\]

**Stack after call**

\[
\begin{array}{c|c}
\text{previous contents} & \text{SP} \\
\hline
\end{array}
\]

**Errors**

None

C

```c
extern pascal void GetPenPat(patternPtr)
Pattern patternPtr;
```
$2D04  GetPenSize

Returns the current pen size to a specified location.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>pointPtr</td>
</tr>
<tr>
<td>Long—POINTER to space for POINT</td>
</tr>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Errors

None

C

extern pascal void GetPenSize(pointPtr)
Point *pointPtr;
$2B04 \textbf{GetPenState}

Returns the pen state from the GrafPort to a specified location. See Figure 16-38 in the section “SetPenState” in this chapter for the definition of the pen state record.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>penStatePtr</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Errors

None

C

```c
extern pascal void GetPenState(penStatePtr)
PenStatePtr penStatePtr;
```
Returns the value of the `picSave` field of the GrafPort.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>longspace</code></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
| ← $P$ Long—Space for result

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>picSaveValue</code></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
| ← $P$ Long—Current `picSave` value

**Errors**

None

**C**

```c
extern pascal Longint GetPicSave()
```
GetPixel

Returns the pixel below and to the right of a specified point.

The `thePixel` result is returned in the lower bits of the word. If the current drawing location has a chunkiness of 2, two bits of the word are valid. If the current drawing location has a chunkiness of 4, four bits of the word are valid.

There is no guarantee that the point actually belongs to the port.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wordspace</code></td>
</tr>
<tr>
<td><code>h</code></td>
</tr>
<tr>
<td><code>v</code></td>
</tr>
</tbody>
</table>

- **Word**—Space for result
- **Word**—Horizontal value of point, in global coordinates
- **Word**—Vertical value of point, in global coordinates

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>thePixel</code></td>
</tr>
</tbody>
</table>

- **Word**—Pixel value

**Errors**

None

**C**

```c
extern pascal Word GetPixel(h, v)

Integer  h;
Integer  v;

You can also use the following alternate form of the call:

extern pascal Word GetPixel(point)

Point    point;
```
$4304 \textbf{GetPolySave}

Returns the value of the \textit{polySave} field of the GrafPort.

\textbf{Parameters}

\textbf{Stack before call}

\begin{tabular}{|c|c|c|}
\hline
\textit{previous contents} & \multicolumn{2}{|c|}{\textbf{Long}—Space for result} \\
\hline
\textit{longspace} & \multicolumn{2}{|c|}{\leftarrow SP} \\
\hline
\end{tabular}

\textbf{Stack after call}

\begin{tabular}{|c|c|c|}
\hline
\textit{previous contents} & \multicolumn{2}{|c|}{\textbf{Long}—Current \textit{polySave} value} \\
\hline
\textit{polySaveValue} & \multicolumn{2}{|c|}{\leftarrow SP} \\
\hline
\end{tabular}

\textbf{Errors} None

\textbf{C}

\begin{verbatim}
extern pascal LongWord GetPolySave()
\end{verbatim}
GetPort

Returns a pointer to the current GrafPort.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
</tbody>
</table>

Long—Space for result

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>resultPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to GrafPort

← SP

Errors

None

C

extern pascal GrafPortPtr GetPort()
$1E04 \text{ GetPortLoc} \\
\text{Gets the current port's locInfo record and puts it at the specified location.}

\textbf{Parameters}

\textbf{Stack before call}

\begin{center}
\begin{tabular}{c|c}
\textit{previous contents} & \textbf{Long—POINTER to space for locInfo record} \\
\hline
locInfoPtr & \leftarrow \text{SP}
\end{tabular}
\end{center}

\textbf{Stack after call}

\begin{center}
\begin{tabular}{c|c}
\textit{previous contents} & \leftarrow \text{SP}
\end{tabular}
\end{center}

\textbf{Errors} \hspace{1cm} \text{None}

\textbf{C}

\begin{verbatim}
extern pascal void GetPortLoc(locInfoPtr) 
locInfoPtr locInfoPtr;
\end{verbatim}
$2004 \textbf{GetPortRect}

Returns the current GrafPort's port rectangle.

\textbf{Parameters}

\textbf{Stack before call}

\begin{verbatim}
| previous contents |
|-- rectPtr --|
\end{verbatim}

\text{Long—POINTER to space for RECT defining rectangle}

\text{← SP}

\textbf{Stack after call}

\begin{verbatim}
| previous contents |
\end{verbatim}

\text{← SP}

\textbf{Errors}

\text{None}

\text{C}

\text{extern pascal void GetPortRect(rectPtr)}

\text{Rect *rectPtr;}
GetRgnSave

Returns the value of the rgnSave field of the GrafPort.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Long—Space for result

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgnSaveValue</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Long—Current rgnSave value

← SP

Errors

None

C

extern pascal LongWord GetRgnSave()
GetRomFont

Fills a specified buffer with information about the font in ROM.

**Parameters**

**Stack before call**

```
| previous contents |
|---|---|
| recordPtr |
```

Long—POINTER to space for ROM font record (see Figure 16-33)

**Stack after call**

```
| previous contents |
|---|---|
| |
```

**Errors**

None

**C**

```c
extern pascal void GetROMFont(recordPtr)
RomFontRecPtr recordPtr;
```

(continued)
The record

The record pointed to by recordPtr has the form illustrated in Figure 16-33.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rfFamNum</td>
<td>Word—INTEGER specifying font family number</td>
</tr>
<tr>
<td>1</td>
<td>rfFamStyle</td>
<td>Word—Specifying font style</td>
</tr>
<tr>
<td>2</td>
<td>rfSlze</td>
<td>Word—INTEGER specifying font size in points</td>
</tr>
<tr>
<td>3</td>
<td>rfFontHandle</td>
<td>Long—HANDLE to font</td>
</tr>
<tr>
<td>4</td>
<td>rfNamePtr</td>
<td>Long—POINTER to font name</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Word—INTEGER indicating fbrExtent</td>
</tr>
</tbody>
</table>

Figure 16-33

ROM font record
GetSCB

Returns the value of a specified SCB (scan line control byte).

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Word—Space for result</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>Word—INTEGER; scan line whose SCB is to be examined</td>
</tr>
<tr>
<td>scanLine</td>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

| previous contents | Word—Value of SCB |
|-------------------| ← SP |

Errors

$0452$ badScanLine Invalid scan line number; 0 to 199 are valid

C

extern pascal Word GetSCB(scanLine)

Word scanLine;
GetSpaceExtra

Returns the value of the spExtra field from the GrafPort.

Parameters

Stack before call

| previous contents |
|-- longspace --|

Long—Space for result

← SP

Stack after call

| previous contents |
|-- spaceExtra --|

Long—FIXED; value of spExtra field

← SP

Errors

None

C

extern pascal Fixed GetSpaceExtra()
GetStandardSCB

Returns a copy of the standard SCB in the low-order byte of the word.

Parameters

Stack before call

\[
\begin{array}{c|c}
\text{previous contents} & \text{wordspace} \\
\hline
\end{array}
\]

Word—Space for result

← SP

Stack after call

\[
\begin{array}{c|c}
\text{previous contents} & \text{scb} \\
\hline
\end{array}
\]

Word—Standard SCB (see Figure 16-34)

← SP

Errors

None

C

extern pascal Word GetStandardSCB()

Standard SCB

The standard SCB has values as shown in Figure 16-34.

Figure 16-34

Standard SCB

QuickDraw II routines 16-145
GetSysField

Returns the value of the sysField field of the GrafPort.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- longspace --</td>
</tr>
</tbody>
</table>

Long—Space for result

<- SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- sysField --</td>
</tr>
</tbody>
</table>

Long—Current sysField value

<- SP

Errors

None

C

extern pascal Longint GetSysField()
GetSysFont

Returns a handle to the current system font.

Parameters

Stack before call

\[
\begin{array}{c|c}
\text{previous contents} & \text{Long—Space for result} \\
\text{longspace} & \leftarrow \text{sp} \\
\end{array}
\]

Stack after call

\[
\begin{array}{c|c}
\text{previous contents} & \text{Long—HANDLE to current system for} \\
\text{fontHandle} & \leftarrow \text{sp} \\
\end{array}
\]

Errors

None

C

\text{extern pascal FontHndl GetSysFont()}

QuickDraw II routines 16-147
$9B04  \textbf{GetTextFace}  

Returns the current text face. See Figure 16-39 in the section “SetTextFace” in this chapter for the bit values for the \textit{textFace} parameter.

\textbf{Parameters}

Stack before call

\begin{verbatim}
previous contents
wordspace
\end{verbatim}

\textbf{Word}—Space for result

\leftarrow SP

Stack after call

\begin{verbatim}
previous contents
textFace
\end{verbatim}

\textbf{Word}—INTEGER; text face (see Figure 16-39 in “SetTextFace”)  

\leftarrow SP

\textbf{Errors} \quad \text{None}

\textbf{C}  

\texttt{extern pascal TextStyle GetTextFace()}
$9D04  GetTextMode

Returns the current text mode. See Table 16-10 in the section "SetTextMode" in this chapter for the modes used only for text.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>wordspace</th>
<th>Word—Space for result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SP</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>textMode</th>
<th>Word—INTEGER; text mode (see Table 16-10 in &quot;SetTextMode&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SP</td>
</tr>
</tbody>
</table>

Errors

None

C

extern pascal Word GetTextMode()
GetTextSize

Returns the current value of the txSize field of the GrafPort. This value may not be the same as the point size of the current font; to obtain that value, use the GetFontLore or GetFontGlobals routines.

Parameters

Stack before call

| previous contents | workspace | Word—Space for result | ← SP

Stack after call

| previous contents | txSize | Word—Current value of txSize field | ← SP

Errors

None

C

extern pascal Integer GetTextSize()
$4704 \textbf{GetUserField}

Returns the value of the \textit{userField} field of the GrafPort.

\textbf{Parameters}

\textbf{Stack before call}

\begin{center}
\begin{tabular}{c|c|}
\textit{previous contents} & \textit{Long}—Space for result \\
\hline
\textit{longspace} & \leftarrow \textit{SP} \\
\end{tabular}
\end{center}

\textbf{Stack after call}

\begin{center}
\begin{tabular}{c|c|}
\textit{previous contents} & \textit{Long}—Current \textit{userField} value \\
\hline
\textit{userField} & \leftarrow \textit{SP} \\
\end{tabular}
\end{center}

\textbf{Errors} \quad \text{None}

\textbf{C} \quad \texttt{extern pascal Longint GetUserField()}

QuickDraw II routines 16-161
GetVisHandle

Returns a copy of the handle to the visible region.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
<tr>
<td>Long—Space for result</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>visRgnHandle</td>
</tr>
<tr>
<td>Long—HANDLE to region</td>
</tr>
</tbody>
</table>

Errors

None

C

extern pascal RgnHandle GetVisHandle()
GetVisRgn

Copies the contents of the visible region into a specified region. The region must have already been created with a NewRgn call.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgnHandle</td>
</tr>
<tr>
<td>Long—HANDLE to region</td>
</tr>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

| previous contents |
| ← SP              |

Errors

None

C

extern pascal void GetVisRgn(rgnHandle)

RgnHandle rgnHandle;
GlobalToLocal

Converts a point from global coordinates to local coordinates. Global coordinates have 0,0 as the upper left corner of the pixel image. Local coordinates are based on the current boundary rectangle of the GrafPort.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>pointPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to POINT to be converted

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

← SP

Errors

None

C

extern pascal void GlobalToLocal(pointPtr)

Point *pointPtr;
$0B04  **GrafOff**

Turns off the Super Hi-Res graphics mode. The routine affects only the bit in the New Video register that affects what is displayed. It does not change the linearization bit in the field. See the *Apple IIgs Hardware Reference* for more information.

**Parameters**
The stack is not affected by this call. There are no input or output parameters.

**Errors**
None

**C**
```c
extern pascal void GrafOff()
```

$0A04  **GrafOn**

Turns on the Super Hi-Res graphics mode. The routine affects only the bit in the New Video register that affects what is displayed. It does not change the linearization bit in the field. See the *Apple IIgs Hardware Reference* for more information.

**Parameters**
The stack is not affected by this call. There are no input or output parameters.

**Errors**
None

**C**
```c
extern pascal void GrafOn()
```
**HideCursor**

Hides the cursor by decrementing the cursor level. A cursor level of 0 indicates the cursor is visible; a cursor level less than 0 indicates the cursor is not visible.

**Parameters**
The stack is not affected by this call. There are no input or output parameters.

**Errors**
None

**C**
```
extern pascal void HideCursor();
```

---

**HidePen**

Decrements the pen level. A non-negative pen level indicates that drawing will occur; a negative pen level indicates that drawing will not occur.

**Parameters**
The stack is not affected by this call. There are no input or output parameters.

**Errors**
None

**C**
```
extern pascal void HidePen();
```
**InflateTextBuffer**

Ensures that the text buffer is big enough to handle a font with the specified width and height, increasing it if necessary.

This routine is usually used only by the Font Manager, but you may need it if your application is dealing with fonts without the Font Manager's help.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>newWidth</td>
</tr>
<tr>
<td>newHeight</td>
</tr>
</tbody>
</table>

Word—INTEGER; width of font

Word—INTEGER; height of font

**Stack after call**

| previous contents |

SP

**Errors**

None

**C**

```c
extern pascal void InflateTextBuffer(newWidth, newHeight)

Word    newWidth;
Word    newHeight;
```
$0D04  **InitColorTable**

Returns a copy of the standard color table for the current mode, as shown in Table 16-7.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablePtr</td>
</tr>
</tbody>
</table>

| Long—POINTER to space for standard color table (see Table 16-7) |

← SP

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

← SP

**Errors**

None

**C**

```c
extern pascal void InitColorTable(tablePtr)

ColorTable tablePtr;
```
The standard color tables are shown in Table 16-7.

### Table 16-7

#### Standard color tables

<table>
<thead>
<tr>
<th>Pixel value</th>
<th>Name</th>
<th>Master color</th>
<th>Pixel value</th>
<th>Name</th>
<th>Master color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entries for 320 mode</td>
<td></td>
<td></td>
<td>Entries for 640 mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Black</td>
<td>000</td>
<td>0</td>
<td>Black</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>Dark gray</td>
<td>777</td>
<td>1</td>
<td>Red</td>
<td>F00</td>
</tr>
<tr>
<td>2</td>
<td>Brown</td>
<td>841</td>
<td>2</td>
<td>Green</td>
<td>0F0</td>
</tr>
<tr>
<td>3</td>
<td>Purple</td>
<td>72C</td>
<td>3</td>
<td>White</td>
<td>FFF</td>
</tr>
<tr>
<td>4</td>
<td>Blue</td>
<td>00F</td>
<td>4</td>
<td>Black</td>
<td>000</td>
</tr>
<tr>
<td>5</td>
<td>Dark green</td>
<td>080</td>
<td>5</td>
<td>Blue</td>
<td>00F</td>
</tr>
<tr>
<td>6</td>
<td>Orange</td>
<td>F70</td>
<td>6</td>
<td>Yellow</td>
<td>FF0</td>
</tr>
<tr>
<td>7</td>
<td>Red</td>
<td>D00</td>
<td>7</td>
<td>White</td>
<td>FFF</td>
</tr>
<tr>
<td>8</td>
<td>Beige</td>
<td>FA9</td>
<td>8</td>
<td>Black</td>
<td>000</td>
</tr>
<tr>
<td>9</td>
<td>Yellow</td>
<td>FF0</td>
<td>9</td>
<td>Red</td>
<td>F00</td>
</tr>
<tr>
<td>10</td>
<td>Green</td>
<td>0E0</td>
<td>A</td>
<td>Green</td>
<td>0F0</td>
</tr>
<tr>
<td>11</td>
<td>Light blue</td>
<td>4DF</td>
<td>B</td>
<td>White</td>
<td>FFF</td>
</tr>
<tr>
<td>12</td>
<td>Lilac</td>
<td>DAF</td>
<td>C</td>
<td>Black</td>
<td>000</td>
</tr>
<tr>
<td>13</td>
<td>Periwinkle blue</td>
<td>78F</td>
<td>D</td>
<td>Blue</td>
<td>00F</td>
</tr>
<tr>
<td>14</td>
<td>Light gray</td>
<td>C0C</td>
<td>E</td>
<td>Yellow</td>
<td>FF0</td>
</tr>
<tr>
<td>15</td>
<td>White</td>
<td>FFF</td>
<td>F</td>
<td>White</td>
<td>FFF</td>
</tr>
</tbody>
</table>
$CA04  \textbf{InitCursor} \\
Reinitializes the cursor. The cursor is set to the arrow cursor and made visible.

This routine also checks the master SCB and sets the cursor accordingly. Use this routine if you want to change modes in the middle of a program. The steps you take to do this are

1. Hide the cursor if it is not already hidden.
2. Set the master SCB to the mode you want.
3. Set all the SCBs to the master SCB.
4. Set the color table the way you want it.
5. Repaint the screen for the new mode.
6. Call \texttt{InitCursor}.

\textbf{Parameters}  The stack is not affected by this call. There are no input or output parameters.

\textbf{Errors}  None

\textbf{C}  
\begin{verbatim}
extern pascal void InitCursor();
\end{verbatim}
$1904  \textbf{InitPort}

Infinites specified memory locations as a standard port.

InitPort, unlike the OpenPort routine, assumes that the region handles are valid and does not allocate new handles. Otherwise, InitPort performs the same functions as OpenPort.

\textbf{Parameters}

\textbf{Stack before call}

\begin{center}
\begin{tabular}{c|c|c|c|}
\hline
\textit{previous contents} & \multicolumn{3}{c}{Long—POINTER to port} \\
\hline
\texttt{portPtr} & \multicolumn{3}{|c}{\leftarrow SP} \\
\hline
\end{tabular}
\end{center}

\textbf{Stack after call}

\begin{center}
\begin{tabular}{c|c|c|c|}
\hline
\textit{previous contents} & \multicolumn{3}{c}{\leftarrow SP} \\
\hline
\end{tabular}
\end{center}

\textbf{Errors}

Memory Manager errors    Returned unchanged

\textbf{C}

```
extern pascal void InitPort(portPtr)
GrafPortPtr portPtr;
```
Insets a specified rectangle by specified displacements. The value specified as \( dH \) is added to the left and subtracted from the right; the value specified as \( dV \) is added to the top and subtracted from the bottom.

**Parameters**

**Stack before call**

```
previous contents
rectPtr    Long—POINTER to rectangle
          Word—INTEGER; horizontal displacement
          Word—INTEGER; vertical displacement
          ← SP
```

**Stack after call**

```
previous contents
          ← SP
```

**Errors**

None

**C**

```c
extern pascal void InsetRect(rectPtr, dH, dV)
Rect *rectPtr;
Integer    dH;
Integer    dV;
```
InsetRgn

Shrinks or expands a specified region. All points on the region boundary are moved inward a distance of \( dH \) horizontally and \( dV \) vertically. If \( dH \) or \( dV \) is negative, the points are moved outward in that direction. InsetRgn leaves the region centered on the same position but moves the outline. InsetRgn of a rectangular region works just like InsetRect.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgnHandle</td>
</tr>
<tr>
<td>( dH )</td>
</tr>
<tr>
<td>( dV )</td>
</tr>
</tbody>
</table>

- Long—HANDLE to region being inset
- Word—INTEGER; horizontal displacement
- Word—INTEGER; vertical displacement

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

\( \leftarrow \) SP

Errors

Memory Manager errors Returned unchanged

C

```c
extern pascal void InsetRgn(rgnHandle, dH, dV)
RgnHandle   rgnHandle;
Integer     dH;
Integer     dV;
```
$6504$ **InvertArc**

Inverts the pixels in the interior of a specified arc.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rectPtr</td>
</tr>
<tr>
<td>startAngle</td>
</tr>
<tr>
<td>arcAngle</td>
</tr>
</tbody>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

None

**C**

```c
extern pascal void InvertArc(rectPtr, startAngle, arcAngle)
```

Rect *rectPtr;

Integer startAngle;

Integer arcAngle;

16-164 QuickDraw II routines
$5B04 \textbf{InvertOval} \\
Inverts the pixels in the interior of a specified oval.

\textbf{Parameters}

\textbf{Stack before call}

```
| \textit{previous contents} |
| \hline
| \textit{rectPtr} |
\hline
\text{Long—POINTER to RECT specifying enclosing rectangle} \leftarrow \text{SP}
```

\textbf{Stack after call}

```
| \textit{previous contents} |
\hline
\text{SP} \leftarrow
```

\textbf{Errors} \hspace{1cm} \textbf{None}

\textbf{C}

```
\textit{extern pascal void InvertOval(rectPtr)}

Rect *rectPtr;
```
**InvertPoly**

Inverts the pixels in the interior of a specified polygon. The polygon is inverted by opening a region, drawing lines, closing the region, and inverting the region.

---

**Parameters**

**Stack before call**

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>polyHandle</td>
</tr>
<tr>
<td>Long-HANDLE to polygon</td>
</tr>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>
```

**Stack after call**

```
| previous contents |
| ← SP              |
```

**Errors**

- Memory Manager errors
- Returned unchanged

**C**

```
extern pascal void InvertPoly(polyHandle)
Handle   polyHandle;
```
InvertRect

Inverts the pixels in the interior of a specified rectangle.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rectPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to RECT specifying rectangle

← SP

Stack after call

| previous contents |

← SP

Errors

None

C

extern pascal void InvertRect(rectPtr)

Rect *rectPtr;
$7C04 \text{ invertRgn}$

Inverts the pixels in the interior of a specified region.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>\textit{previous contents}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{rgnHandle}</td>
</tr>
</tbody>
</table>

\begin{itemize}
  \item \texttt{Long}—HANDLE to region
  \item \texttt{SP}
\end{itemize}

**Stack after call**

<table>
<thead>
<tr>
<th>\textit{previous contents}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{SP}</td>
</tr>
</tbody>
</table>

**Errors**

None

**C**

\begin{verbatim}
extern pascal void invertRgn(rgnHandle);
\end{verbatim}
$6004 \textbf{InvertRRect}

Inverts the pixels in the interior of a specified round rectangle.

The corners of the round rectangle are sections of an oval defined by \textit{ovalHeight} and \textit{ovalWidth}. For more information, see Figure 16-12 in the section "Rectangles" in this chapter.

\textbf{Parameters}

\textbf{Stack before call}

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- rectPtr --</td>
</tr>
<tr>
<td>ovalWidth</td>
</tr>
<tr>
<td>ovalHeight</td>
</tr>
</tbody>
</table>

\textbf{Long}—POINTER to RECT specifying enclosing rectangle

\textbf{Word}—INTEGER; width, in pixels, of oval defining rounded corners

\textbf{Word}—INTEGER; height, in pixels, of oval defining rounded corners

\leftarrow SP

\textbf{Stack after call}

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\leftarrow SP</td>
</tr>
</tbody>
</table>

\textbf{Errors} \hspace{1cm} \textbf{None}

\textbf{C}

\texttt{extern pascal void InvertRRect(rectPtr, ovalWidth, ovalHeight)}

\texttt{Rect *rectPtr;}

\texttt{Word ovalWidth;}

\texttt{Word ovalHeight;}

QuickDraw II routines 16-169
KillPoly
Disposes of a specified polygon.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>polyHandle</td>
</tr>
<tr>
<td>Long — HANDLE to polygon to be killed</td>
</tr>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Errors

Memory Manager errors Returned unchanged

C

extern pascal void KillPoly(polyHandle)
Handle polyHandle;

16-170 QuickDraw II routines
**Line**

Draws a line from the current pen location to a new point specified by the horizontal and vertical displacements.

---

**Important**

If a region is open, this command contributes to the region definition; this can cause Memory Manager errors to occur.

---

**Parameters**

**Stack before call**

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>dH</td>
</tr>
<tr>
<td>dV</td>
</tr>
</tbody>
</table>
```

- **Word**—INTEGER; horizontal displacement in points
- **Word**—INTEGER; vertical displacement in points

**Stack after call**

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>
```

**Errors**

Memory Manager errors → Returned unchanged

**C**

```c
extern pascal void Line(dH, dV)
```

```c
Integer dH;
Integer dV;
```
LineTo

Draws a line from the current pen location to a specified point. The point must be expressed in local coordinates.

Important
If a region is open, this command contributes to the region definition; this can cause Memory Manager errors to occur.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
</tr>
<tr>
<td>v</td>
</tr>
</tbody>
</table>

Word—INTEGER; horizontal point to which line will be drawn
Word—INTEGER; vertical point to which line will be drawn
← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Errors

Memory Manager errors Returned unchanged

C

extern pascal void LineTo(h,v)
Integer h;
Integer v;
You can also use the following alternate form of the call:
external pascal void LineTo(point)
Point point;
LocalToGlobal

Converts a point from local coordinates to global coordinates. Local coordinates are based on the current boundary rectangle of the GrafPort. Global coordinates have 0,0 as the upper left corner of the pixel image.

**Parameters**

**Stack before call**

| previous contents |
|-------------------|------------------|
| pointPtr           | Long—POINTER to POINT to be converted |

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

**Errors**

None

**C**

extern pascal void LocalToGlobal(pointPtr)

Point *pointPtr;
$C504$  MapPoly

Maps a specified polygon from a source rectangle to a destination rectangle.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>polyHandle</td>
</tr>
<tr>
<td>srcRectPtr</td>
</tr>
<tr>
<td>destRectPtr</td>
</tr>
</tbody>
</table>

- **Long**—HANDLE to polygon to be mapped
- **Long**—POINTER to RECT defining source rectangle
- **Long**—POINTER to RECT defining destination rectangle

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

- **SP**

**Errors**

None

**C**

```
extern pascal void MapPoly(polyHandle, srcRectPtr, destRectPtr)
Handle    polyHandle;
Rect *    srcRectPtr;
Rect *    destRectPtr;
```
$8A04 \textbf{MapPt}

Maps a specified point from a source rectangle to a destination rectangle.

\textbf{Parameters}

\textbf{Stack before call}

\begin{verbatim}
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>pointPtr</td>
</tr>
<tr>
<td>srcRectPtr</td>
</tr>
<tr>
<td>destRectPtr</td>
</tr>
</tbody>
</table>
\end{verbatim}

\textbf{Stack after call}

\begin{verbatim}
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
</tr>
</tbody>
</table>
\end{verbatim}

\textbf{Errors}

None

\textbf{C}

\begin{verbatim}
extern pascal void MapPt(pointPtr, srcRectPtr, destRectPtr)
Point *pointPtr;
Rect *srcRectPtr;
Rect *destRectPtr;
\end{verbatim}
MapRect
Maps a specified rectangle from a source rectangle to a destination rectangle.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rectPtr</td>
</tr>
<tr>
<td>srcRectPtr</td>
</tr>
<tr>
<td>destRectPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to RECT defining rectangle to be mapped
Long—POINTER to RECT defining source rectangle
Long—POINTER to RECT defining destination rectangle

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<- SP

Errors
None

C

extern pascal void MapRect(rectPtr,srcRectPtr,destRectPtr)
Rect *rectPtr;
Rect *srcRectPtr;
Rect *destRectPtr;
$8C04 \textbf{MapRgn}

Maps a specified region from a source rectangle to a destination rectangle.

\textbf{Parameters}

\textbf{Stack before call}

\begin{verbatim}
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>
|-- mapRgnHandle -- | \textbf{Long}—HANDLE to region to be mapped
|-- srcRectPtr     | \textbf{Long}—POINTER to RECT defining source rectangle
|-- destRectPtr    | \textbf{Long}—POINTER to RECT defining destination rectangle
|--                   | ← SP
|-------------------|
\end{verbatim}

\textbf{Stack after call}

\begin{verbatim}
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
</tbody>
</table>
\end{verbatim}

\textbf{Errors}

Memory Manager errors Returned unchanged

\textbf{C}

\begin{verbatim}
extern pascal void MapRgn(mapRgnHandle, srcRectPtr, destRectPtr)
RgnHandle mapRgnHandle;
Rect *srcRectPtr;
Rect *destRectPtr;
\end{verbatim}
Move

Moves the current pen location by specified horizontal and vertical displacements.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{dH}</td>
</tr>
<tr>
<td>\text{dV}</td>
</tr>
</tbody>
</table>

\text{Word—INTEGER; horizontal displacement}

\text{Word—INTEGER; vertical displacement}

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\leftarrow SP</td>
</tr>
</tbody>
</table>

Errors

None

C

\begin{verbatim}
extern pascal void Move(dH, dV)
Integer dH;
Integer dV;
\end{verbatim}
MovePortTo

Changes the location of the current GrafPort’s port rectangle. This routine does not affect the pixel image but changes the active area of the GrafPort. The call is normally used by the Window Manager.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h )</td>
</tr>
<tr>
<td>( v )</td>
</tr>
</tbody>
</table>

- **Word**—INTEGER; horizontal coordinate of upper left corner
- **Word**—INTEGER; vertical coordinate of upper left corner

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \leftarrow SP )</td>
</tr>
</tbody>
</table>

Errors

None

C

```c
extern pascal void MovePortTo(h,v)
```

```c
Integer  h;
Integer  v;
```
$3A04  MoveTo

Moves the current pen location to a specified point. The point is specified in local coordinates.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
</tr>
<tr>
<td>v</td>
</tr>
</tbody>
</table>

| Word—INTEGER; horizontal point in local coordinates |
| Word—INTEGER; vertical point in local coordinates |

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

| ← SP |

Errors

None

C

extern pascal void MoveTo(h,v)

Integer h;

Integer v;

You can also use the following alternate form of the call:

extern pascal void MoveTo(point)

Point point;
**$6704 NewRgn**

Allocates space for a new region and initializes it to an empty region. The empty region for this purpose is a rectangular region with a bounding box of (0,0,0,0).

---

**Important**

NewRgn is the only routine that creates a new region; all other routines work with existing regions.

---

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
</tbody>
</table>

Long—Space for result

← SP

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgnHandle</td>
</tr>
</tbody>
</table>

Long—HANDLE to new region

← SP

**Errors**

Memory Manager errors   Returned unchanged

**C**

extern pascal RgnHandle NewRgn()
**$5204**  
**NotEmptyRect**

Indicates whether a specified rectangle is not empty. An empty rectangle has the top greater than or equal to the bottom or the left greater than or equal to the right.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Word — Space for result</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td></td>
</tr>
<tr>
<td>rectPtr</td>
<td>Long — POINTER to RECT defining rectangle</td>
</tr>
</tbody>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Word — BOOLEAN; TRUE if rectangle not empty, FALSE if empty</th>
</tr>
</thead>
<tbody>
<tr>
<td>notEmptyFlag</td>
<td></td>
</tr>
</tbody>
</table>

**Errors**

None

**C**

```c
extern pascal Boolean NotEmptyRect(rectPtr)
Rect *rectPtr;
```

**$9204**  
**ObscureCursor**

Hides the cursor until the mouse moves. This routine can get the cursor out of the way of typing.

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

**Errors**

None

**C**

```c
extern pascal void ObscureCursor()
```
$C404$ OffsetPoly

Offsets a specified polygon by specified horizontal and vertical displacements.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- polyHandle --</td>
</tr>
<tr>
<td>dH</td>
</tr>
<tr>
<td>dV</td>
</tr>
</tbody>
</table>

| Long—HANDLE to polygon |
| Word—INTEGER; horizontal displacement in pixels |
| Word—INTEGER; vertical displacement in pixels |

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Errors

Memory Manager errors  Returned unchanged

C

extern pascal void OffsetPoly(polyHandle, dH, dV)
Handle polyHandle;
Integer dH;
Integer dV;
$4804 \textbf{OffsetRect}

Offsets a specified rectangle by specified displacements. The value of \( dH \) is added to the left and right; the value of \( dV \) is added to the top and bottom.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rectPtr</td>
</tr>
<tr>
<td>( dH )</td>
</tr>
<tr>
<td>( dV )</td>
</tr>
</tbody>
</table>

- **Long**—POINTER to rectangle
- **Word**—INTEGER; horizontal displacement in pixels
- **Word**—INTEGER; vertical displacement in pixels

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

- \( \leftarrow \text{SP} \)

**Errors**

None

**C**

```c
extern pascal void OffsetRect(rectPtr, dH, dV)

Rect *rectPtr;
Integer dH;
Integer dV;
```

16-184 QuickDraw II routines
OffsetRgn

Moves a region on the coordinate plane a distance of $dH$ horizontally and $dV$ vertically. The region retains its size and shape.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgnHandle</td>
</tr>
<tr>
<td>$dH$</td>
</tr>
<tr>
<td>$dV$</td>
</tr>
</tbody>
</table>

- **Long**—HANDLE to region being offset
- **Word**—INTEGER; horizontal displacement in pixels
- **Word**—INTEGER; vertical displacement in pixels

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leftarrow$ SP</td>
</tr>
</tbody>
</table>

**Errors**

Memory Manager errors Returned unchanged

**C**

```
extern pascal void OffsetRgn(rgnHandle, dH, dV)

RgnHandle rgnHandle;
Integer dh;
Integer dv;
```
**OpenPoly**

Returns a handle to a polygon data structure that will be updated by future LineTo calls.

The polygon is completed by making a ClosePoly call.

**Parameters**

**Stack before call**

```
previous contents
    longspace  Long—Space for result
← SP
```

**Stack after call**

```
previous contents
    polyHandle  Long—HANDLE to polygon
← SP
```

**Errors**

$0440$0 polyAlreadyOpen Polygon already open and being saved in current GrafPort

Memory Manager errors Returned unchanged

C

```c
extern pascal Handle OpenPoly()
```
$1804 OpenPort

Initializes specified memory locations as a standard GrafPort, allocates a new visible region and a new clipping region, and makes the GrafPort the current port.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>portPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to space for GrafPort record

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

Errors

Memory Manager errors Returned unchanged

C

extern pascal void OpenPort(portPtr)

GrafPortPtr portPtr;

$6D04 OpenRgn

Allocates temporary space and starts saving lines and framed shapes for later processing as a region definition. The routine takes no inputs; instead, it allocates memory to hold information about the region being created. When the CloseRgn routine is called, the region is created and this memory is freed.

While the region is open, all calls to Line, LineTo, FrameRect, FrameOval, FrameRRect, FrameRgn, and FramePoly contribute to the region definition.

Parameters

The stack is not affected by this call. There are no input or output parameters.

Errors

$0430 rgnAlreadyOpen Region already being saved in current GrafPort

Memory Manager errors Returned unchanged

C

extern pascal void OpenRgn()

QuickDraw II routines 16-187
PaintArc

Paints the interior of a specified arc using the current pen mode and pen pattern.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rectPtr</td>
</tr>
<tr>
<td>startAngle</td>
</tr>
<tr>
<td>arcAngle</td>
</tr>
</tbody>
</table>

- Long—POINTER to RECT defining enclosing rectangle
- Word—INTEGER; starting angle in degrees
- Word—INTEGER; arc angle in degrees

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Errors

None

C

```c
extern pascal void PaintArc(rectPtr, startAngle, arcAngle)
Rect *rectPtr;
Integer startAngle;
Integer arcAngle;
```
$5904 \textbf{PaintOval} \\
Paints the interior of a specified oval using the current pen mode and pen pattern.

\textbf{Parameters}

\textbf{Stack before call}

\begin{tabular}{|c|}
\hline
\textit{previous contents} \\
\textit{rectPtr} \textbf{Long—POINTER to RECT defining enclosing rectangle} \\
\hline
\end{tabular}

\textbf{Stack after call}

\begin{tabular}{|c|}
\hline
\textit{previous contents} \\
\hline
\end{tabular}

\textbf{Errors}  \\
None

\textbf{C}

\begin{verbatim}
extern pascal void PaintOval(rectPtr)
Rect *rectPtr;
\end{verbatim}
$7F04 \textbf{PaintPixels}

Transfers a region of pixels. The pixels are transferred without referencing the current GrafPort. The source and destination are defined in the input, as is the clipping region.

\textbf{Parameters}

Stack before call

\begin{itemize}
  \item \textit{previous contents}
  \item \textit{paintParamPtr} \quad \textbf{Long} — POINTER to parameter block (see Figure 16-35)
\end{itemize}

\leftarrow SP

Stack after call

\begin{itemize}
  \item \textit{previous contents} \quad \leftarrow SP
\end{itemize}

\textbf{Errors}

$0420 \text{ notEqualChunkiness} \quad \text{Source and destination pixel images not the same type (one is for 320 mode display and one for 640 mode display)}$

\textbf{C}

\begin{verbatim}
extern pascal void PaintPixels(paintParamPtr)
PaintParamPtr paintParamPtr;
\end{verbatim}

16-190 \textit{QuickDraw II routines}
### Parameter block

The parameter block pointed to by `paintParamPtr` is shown in Figure 16-35.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><code>ptrToSourceLocInfo</code></td>
<td>Long—POINTER to source location information</td>
</tr>
<tr>
<td>1</td>
<td><code>ptrToDestLocInfo</code></td>
<td>Long—POINTER to destination location information</td>
</tr>
<tr>
<td>2</td>
<td><code>ptrToSourceRect</code></td>
<td>Long—POINTER to source rectangle</td>
</tr>
<tr>
<td>3</td>
<td><code>ptrToDestPoint</code></td>
<td>Long—POINTER to destination point</td>
</tr>
<tr>
<td>4</td>
<td><code>mode</code></td>
<td>Word—mode</td>
</tr>
<tr>
<td>5</td>
<td><code>maskHandle</code></td>
<td>Long—HANDLE to ClipRgn</td>
</tr>
</tbody>
</table>

Figure 16-35
PaintPixels parameter block
$BD04  PaintPoly

Paints the interior of a specified polygon using the current pen mode and pen pattern.

Important
Because this call allocates and deallocates some temporary memory space, Memory Manager errors can occur.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>polyHandle</td>
</tr>
</tbody>
</table>

Long—HANDLE to polygon

Stack after call

| previous contents |

Errors
Memory Manager errors Returned unchanged

C

extern pascal void PaintPoly(polyHandle)

Handle polyHandle;
$5404  \textbf{PaintRect}  \\
Paints the interior of a specified rectangle using the current pen mode and pen pattern.

\section*{Parameters}

\subsection*{Stack before call}

\begin{center}
\begin{tabular}{c|c}
\textit{previous contents} & \\
\hline
\texttt{rectPtr} & \texttt{Long}—\texttt{POINTER to RECT} defining enclosing rectangle \\
\hline
\end{tabular}
\end{center}

\subsection*{Stack after call}

\begin{center}
\begin{tabular}{c|c}
\textit{previous contents} & \\
\hline
 & \leftarrow \texttt{SP} \\
\end{tabular}
\end{center}

\section*{Errors}

None

\subsection*{C}

\begin{verbatim}
extern pascal void PaintRect(rectPtr)

Rect *rectPtr;
\end{verbatim}
$7A04 \textbf{PaintRgn} \quad \text{Paints the interior of a specified region using the current pen mode and pen pattern.}

\textbf{Parameters}

\textbf{Stack before call}

\begin{center}

\begin{tabular}{c|c}
\textit{previous contents} & Long--HANDLE to region \\
--- & SP
\end{tabular}
\end{center}

\textbf{Stack after call}

\begin{center}

\begin{tabular}{c|c}
\textit{previous contents} & SP
\end{tabular}
\end{center}

\textbf{Errors} \quad \text{None}

\textbf{C}

\begin{verbatim}
extern pascal void PaintRgn(rgnHandle)
RgnHandle rgnHandle;
\end{verbatim}
$5E04 \textbf{PaintRRect}

Paints the interior of a specified round rectangle using the current pen mode and pen pattern.

The corners of the round rectangle are sections of an oval defined by \textit{ovalHeight} and \textit{ovalWidth}. For more information, see Figure 16-12 in the section "Rectangles" in this chapter.

\textbf{Parameters}

\textbf{Stack before call}

\begin{tabular}{|c|}
\hline
\textit{previous contents} \\
\hline
--- \textit{recPtr} & \textbf{Long}---\textbf{POINTER to RECT defining enclosing rectangle} \\
--- \textit{ovalWidth} & \textbf{Word}---\textbf{INTEGER; width, in pixels, of oval defining rounded corners} \\
--- \textit{ovalHeight} & \textbf{Word}---\textbf{INTEGER; height, in pixels, of oval defining rounded corners} \\
\hline
\end{tabular}

\textbf{Stack after call}

\begin{tabular}{|c|}
\hline
\textit{previous contents} & \textbf{SP} \\
\hline
\end{tabular}

\textbf{Errors}

None

\textbf{C}

\begin{verbatim}
extern pascal void PaintRRect(rectPtr, ovalWidth, ovalHeight)

Rect *rectPtr;
Word ovalWidth;
Word ovalHeight;
\end{verbatim}
**$3604** PenNormal

Sets the pen state to the standard state, as shown in Table 16-8. Pen location and visibility are not changed.

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

**Errors**

None

**C**

```c
extern pascal void PenNormal()
```

---

**Standard pen state table**

**Table 16-8**

<table>
<thead>
<tr>
<th>Pen state</th>
<th>Standard state</th>
</tr>
</thead>
<tbody>
<tr>
<td>PenSize</td>
<td>1,1</td>
</tr>
<tr>
<td>PenMode</td>
<td>Copy</td>
</tr>
<tr>
<td>PenPat</td>
<td>Black</td>
</tr>
<tr>
<td>PenMask</td>
<td>1's</td>
</tr>
</tbody>
</table>

---

16-196 QuickDraw II routines
**$D604 PPToPort**

Transfers pixels from a source pixel map to the current port and clips the pixels to the current visible region and clipping region. The routine differs from PaintPixels in that the current GrafPort is used as the destination.

PPToPort can help you do the clipping correctly when you are painting a pixel image to a window.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- srcLocPtr</td>
</tr>
<tr>
<td>-- srcRectPtr</td>
</tr>
<tr>
<td>destX</td>
</tr>
<tr>
<td>destY</td>
</tr>
<tr>
<td>transferMode</td>
</tr>
</tbody>
</table>

**Long**—POINTER to source LocInfo record

**Long**—POINTER to RECT defining source rectangle

**Word**—INTEGER; X coordinate of upper left corner of destination

**Word**—INTEGER; Y coordinate of upper left corner of destination

**Word**—Same as pen mode (see Table 16-9 in “SetPenMode”)

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

$0420 notEqualChunkiness Source and destination pixel images not the same type (one is for 320 mode display and one for 640 mode display)

**C**

```c
extern pascal void PPToPort(srcLocPtr, srcRectPtr, destX, destY, transferMode)
LocInfoPtr srcLocPtr;
Rect *srcRectPtr;
Integer destX;
Integer destY;
Word transferMode;
```
You can also use the following alternate form of the call:

```c
extern pascal void PPToPort(srcLocPtr, srcRectPtr, dest, transferMode)
LocInfoPtr srcLocPtr;
Rect *srcRectPtr;
Point dest;
Word transferMode;
```
Pt2Rect

Copies a specified point to the upper left corner of a specified rectangle and another point to the lower right corner of the rectangle.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>point1Ptr</td>
</tr>
<tr>
<td>point2Ptr</td>
</tr>
<tr>
<td>rectPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to first source POINT
Long—POINTER to second source POINT
Long—POINTER to destination rectangle
← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

← SP

Errors

None

C

extern pascal void Pt2Rect(point1Ptr,point2Ptr,rectPtr)
Point *point1Ptr;
Point *point2Ptr;
Rect *rectPtr;

QuickDraw II routines 16-199
$4F04  PtInRect

Dectes whether the pixel below and to the right of a specified point is in a specified rectangle. The routine returns TRUE if the pixel is within the rectangle and FALSE if it is not. For example, PtInRect((10,10),(10,10,20,20)) is TRUE, but PtInRect((20,20),(10,10,20,20)) is FALSE.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>wordspace</td>
</tr>
<tr>
<td></td>
<td>pointPtr</td>
</tr>
<tr>
<td></td>
<td>rectPtr</td>
</tr>
</tbody>
</table>

- **Word**—Space for result
- **Long**—POINTER to POINT
- **Long**—POINTER to RECT defining rectangle

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pointFlag</td>
</tr>
</tbody>
</table>

- **Word**—BOOLEAN; TRUE if pixel in rectangle, FALSE if not

Errors

None

C

extern pascal Boolean PtInRect(pointPtr,rectPtr)
Point *pointPtr;
Rect *rectPtr;

16-200  QuickDraw II routines
PtInRgn

Checks to see whether the pixel below and to the right of a specified point is within a specified region. The routine returns TRUE if the pixel is within the region and FALSE if it is not.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
<td>Word—Space for result</td>
</tr>
<tr>
<td>pointPtr</td>
<td>Long—POINTER to POINT</td>
</tr>
<tr>
<td>rgnHandle</td>
<td>Long—HANDLE to region</td>
</tr>
<tr>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pixelFlag</td>
<td>Word—BOOLEAN; TRUE if pixel is within region, FALSE if not</td>
</tr>
<tr>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

Errors

Memory Manager errors Returned unchanged

C

extern pascal Boolean PtInRgn(pointPtr, rgnHandle)

Point *pointPtr;
RgnHandle rgnHandle;
$8604 \textbf{Random}

Returns a pseudorandom number in the range $-32768$ to $32767$. The sequence of numbers generated by repeated calls to this routine depends on the \textit{randomSeed} value set by a SetRandSeed call. In particular, a call to SetRandSeed with a given \textit{randomSeed} value, followed by a sequence of calls to Random (with no SetRandSeed calls in between), will always produce the same sequence of pseudorandom numbers. This can be useful in debugging.

\textbf{Parameters}

\textbf{Stack before call}

\begin{itemize}
  \item \textbf{previous contents}
  \item \textbf{workspace}
  \item \textbf{Word}—Space for result
\end{itemize}

\textbf{Stack after call}

\begin{itemize}
  \item \textbf{previous contents}
  \item \textbf{randomInt}
  \item \textbf{Word}—INTEGER; pseudorandom number
\end{itemize}

\textbf{Errors}

None

\textbf{C}

\begin{verbatim}
extern pascal Integer Random()
\end{verbatim}
RectInRgn

Checks whether a specified rectangle intersects a specified region.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
<td>Word</td>
</tr>
<tr>
<td>-- rectPtr</td>
<td>Long</td>
</tr>
<tr>
<td>-- rgnHandle</td>
<td>Long</td>
</tr>
<tr>
<td>-- SP</td>
<td>Word</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>encloseFlag</td>
<td>Word</td>
</tr>
<tr>
<td>-- SP</td>
<td>Word</td>
</tr>
</tbody>
</table>

Errors

Memory Manager errors Returned unchanged

C

extern pascal Boolean RectInRgn(rectPtr,rgnHandle)

Rect *rectPtr;
RgnHandle rgnHandle;
RectRgn

Destroys previous region information by setting a specified region to a specified rectangle. If the input does not describe a valid rectangle, the region is set to an empty region. If the original region was not rectangular, the region is resized.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgnHandle</td>
</tr>
<tr>
<td>rectPtr</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Errors

Memory Manager errors Returned unchanged

C

extern pascal void RectRgn(rgnHandle, rectPtr)
RgnHandle rgnHandle;
Rect *rectPtr;
**$CE04 RestoreBufDims**

Restores QuickDraw II's internal buffers to the sizes described in the eight-byte record created by the SaveBufDims routine. You can use this routine when you want your application to change temporarily (but be able to restore) the size of the QuickDraw II buffers.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>sizelnfoPtr</td>
</tr>
</tbody>
</table>

← SP

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

← SP

**Errors**

Memory Manager errors  Returned unchanged

**C**

```c
extern pascal void RestoreBufDims(sizeInfoPtr)
BufDimRecPtr sizeInfoPtr;
```
$CD04 \textbf{SaveBufDims}

Saves QuickDraw II's buffer-sizing information in an eight-byte record. You can use this routine when you want your application to change temporarily (but be able to restore) the size of the QuickDraw II buffers. The buffer-sizing record is shown in Figure 16-36.

\textbf{Parameters}

\textbf{Stack before call}

\begin{table}
\begin{tabular}{|l|}
\hline
\textit{previous contents} \hspace{5em} \textbf{Long—POINTER} to space for record (see Figure 16-36) \hline
\hline
\hspace{1em} \textit{sizeInfoPtr} \hspace{2em} \leftarrow \textit{SP} \hline
\end{tabular}
\end{table}

\textbf{Stack after call}

\begin{table}
\begin{tabular}{|l|}
\hline
\textit{previous contents} \hline
\hline
\hspace{1em} \leftarrow \textit{SP} \hline
\end{tabular}
\end{table}

\textbf{Errors}

None

\textbf{C}

\begin{verbatim}
extern pascal void SaveBufDims(sizeInfoPtr)
BufDimRecPtr sizeInfoPtr;
\end{verbatim}

\textbf{Buffer-sizing record}

The eight-byte record created by SaveBufDims is shown in Figure 16-36.

\begin{table}
\begin{tabular}{|c|c|}
\hline
\textbf{Offset} & \textbf{Field} \hline
0 & \textbf{maxWidth} \hspace{1em} \textit{Word—Application-defined maximum pixel image width} \hline
1 & \textbf{textBufHeight} \hspace{1em} \textit{Word—Current text buffer height, in pixels} \hline
2 & \textbf{textBufferWords} \hspace{1em} \textit{Word—Current text buffer width, in words} \hline
3 & \textbf{fontWidth} \hspace{1em} \textit{Word—Equal to maxFBR Extent} \hline
\end{tabular}
\end{table}

\textbf{Figure 16-36}

BufDimRec

16-206 QuickDraw II routines
$8904 \textbf{ScalePt}

Scales a specified point from a source rectangle to a destination rectangle.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>--- ---</td>
</tr>
<tr>
<td>pointPtr</td>
</tr>
<tr>
<td>srcRectPtr</td>
</tr>
<tr>
<td>destRectPtr</td>
</tr>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

None

C

```c
extern pascal void ScalePt(pointPtr, srcRectPtr, destRectPtr)
Point *pointPtr;
Rect *srcRectPtr;
Rect *destRectPtr;
```
ScrollRect

Shifts the pixels inside the intersection of a specified rectangle, visible region, clipping region, port rectangle, and bounds rectangle. No other pixels are affected. The pixels are shifted a distance of \(dH\) horizontally and \(dV\) vertically (those shifted out of the scroll area are lost). Positive directions are to the right and down. The background pattern fills the space created by the scroll. In addition, the region for \(updateRgnHandle\) is changed to the area filled with the background pattern.

\*Note: The update region must be an existing region; ScrollRect does not create it. If you do not want the update region, you may pass NIL.

Parameters

Stack before call

| previous contents |
| --- | --- |
| \(rectPtr\) | Long—POINTER to RECT defining rectangle |
| \(dH\) | Word—INTEGER; horizontal distance to scroll in pixels |
| \(dV\) | Word—INTEGER; vertical distance to scroll in pixels |
| \(updateRgnHandle\) | Long—HANDLE to region |

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\leftarrow SP)</td>
</tr>
</tbody>
</table>

Errors

Memory Manager errors Returned unchanged

C

```c
extern pascal void ScrollRect(rectPtr, dH, dV, updateRgnHandle)
Rect *rectPtr;
Integer dH;
Integer dV;
RgnHandle updateRgnHandle;
```
**SectRect**

Calculates the intersection of two rectangles and places the intersection in a destination rectangle. The destination rectangle can be one of the source rectangles. If the result is not empty, the output is TRUE; if the result is empty, the output is FALSE.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
<tr>
<td>rect1Ptr</td>
</tr>
<tr>
<td>rect2Ptr</td>
</tr>
<tr>
<td>intersectRectPtr</td>
</tr>
</tbody>
</table>

- **Word**—Space for result
- **Long**—POINTER to RECT defining first source rectangle
- **Long**—POINTER to RECT defining second source rectangle
- **Long**—POINTER to RECT defining destination rectangle

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>notEmptyFlag</td>
</tr>
</tbody>
</table>

- **Word**—BOOLEAN; TRUE if rectangle not empty, FALSE if empty

**Errors**

None

**C**

```c
extern pascal Boolean SectRect(rect1Ptr, rect2Ptr, intersectRectPtr)

Rect *rect1Ptr;
Rect *rect2Ptr;
Rect *intersectRectPtr;
```
$7104 \textbf{SectRgn}

Calculates the intersection of two regions and places the intersection in a destination region. The destination region, which may be one of the source regions, must already exist; SectRgn does not allocate it.

If the regions do not intersect, or if one of the regions is empty, the destination is set to the empty region.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>\textbf{Long}—HANDLE to one source region</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{rgn1Handle}</td>
<td>\textbf{rgn2Handle}</td>
</tr>
<tr>
<td>\textbf{Long}—HANDLE to another source region</td>
<td></td>
</tr>
<tr>
<td>\textbf{destRgnHandle}</td>
<td>\textbf{Long}—HANDLE to destination region</td>
</tr>
</tbody>
</table>

\textbf{Stack after call}

| previous contents |
| \textbf{SP} |

**Errors**

| Memory Manager errors | Returned unchanged |

\textbf{C}

\begin{verbatim}
extern pascal void SectRgn(rgn1Handle, rgn2Handle, destRgnHandle)
RgnHandle rgn1Handle;
RgnHandle rgn2Handle;
RgnHandle destRgnHandle;
\end{verbatim}
$1404 \textbf{SetAllSCBs}

Sets all SCBs (scan line control bytes) to a specified value.

\textbf{Parameters}

\textbf{Stack before call}

\begin{center}
\begin{tabular}{c|c|c}
& \textit{previous contents} & Word—New value for SCBs \\
\hline
\textit{newSCB} & & $\leftarrow \text{SP}$
\end{tabular}
\end{center}

\textbf{Stack after call}

\begin{center}
\begin{tabular}{c|c|c}
& \textit{previous contents} & $\leftarrow \text{SP}$
\end{tabular}
\end{center}

\textbf{Errors}

None

\textbf{C}

\begin{verbatim}
extern pascal void SetAllSCBs(newSCB)

Word newSCB;
\end{verbatim}
$8004  SetArcRot

Sets the arcRot field in the GrafPort to a specified value.

Parameters

Stack before call

| previous contents | arcRotValue | Word − INTEGER; new value of arcRot field ← SP |

Stack after call

| previous contents | ← SP |

Errors  None

C

extern pascal void SetArcRot(arcRotValue)

Integer arcRotValue;
$A204 SetBackColor

Sets the bgColor (background color) field in the GrafPort to a specified value. Background color has either a two- or a four-bit value, depending on the port SCB. If the port SCB indicates 320 mode, the lower four bits of backColor are used. If the port SCB indicates 640 mode, the lower two bits of backColor are used.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>backColor</td>
</tr>
</tbody>
</table>

Word–INTEGER; background color

← SP

Stack after call

| previous contents |

← SP

Errors

None

C

extern pascal void SetBackColor(backColor)

Word   backColor;
SetBackPat

Sets the background pattern to a specified pattern.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>patternPtr</td>
</tr>
</tbody>
</table>

- Long—POINTER to pattern

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

← SP

Errors

None

C

extern pascal void SetBackPat(patternPtr)

Pattern patternPtr;
$CB04 SetBufDims

Sets the size of the QuickDraw II clipping and text buffers. This routine overrides the `maxWidth` value supplied to QDStartUp and the text buffer defaults set at that time.

*Note:* You only need to make this call if your application is going to use, or allow the user to choose, fonts that have unusually large values of `chExtra` and `spExtra`.

SetBufDims pads the text buffer to permit values of `chExtra ≤ jbrExtent` (of the currently active font), `spExtra ≤ jbrExtent`, and style modifications that add up to 36 pixels to the bounds width (width of foreground and background) of any character.

When QDStartUp is called, it makes an internal call to SetBufDims with the following values:

- `maxWidth` As supplied by the application
- `maxFontHeight` 2 · (height of system font)
- `maxFBRExtent` 2 · (jbrExtent of system font)

### Parameters

#### Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxWidth</td>
</tr>
<tr>
<td>maxFontHeight</td>
</tr>
<tr>
<td>maxFBRExtent</td>
</tr>
</tbody>
</table>

- `Word`—INTEGER; width, in bytes, of widest pixel image to be used
- `Word`—INTEGER; height, in pixels, of tallest font application will use
- `Word`—INTEGER; greatest jbrExtent, in pixels, of any font to be used

#### Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

### Errors

- Memory Manager errors Returned unchanged

### C

```c
extern pascal void SetBufDims(maxWidth, maxFontHeight, maxFBRExtent)

Word  maxWidth;
Word  maxFontHeight;
Word  maxFBRExtent;
```

(continued)
More about parameters

The `maxWidth` parameter is the width, in bytes, of the widest pixel image the application will draw into. The `maxFontHeight` parameter is the height of the tallest font the application will use (that is, `fRectHeight` from the font record, computable as `ascent` plus `descent` from the `GetFontInfo` call).

The `maxFBRExtent` parameter is the greatest `fbrExtent` value of any font the application will use. A field in the font record, `fbrExtent` is returned by a `GetFontLore` or `GetFontGlobals` call. It is defined as the greatest (horizontal) distance, in pixels, from the character origin to the farthest foreground or background pixel of any character in the font. For more information, see the section "Fonts and Text in QuickDraw II" in this chapter.
$D404  SetCharExtra

Sets the *chExtra* field in the GrafPort to the specified value. The *chExtra* field is used to add width to every character in the font that has width. It does not affect 0 width characters. This field is present because some fonts that look fine in one graphics mode need a little extra space between characters in another mode.

If you set a very large value of *chExtra*, you may have to change the size of the QuickDraw II buffer. See the sections "SetBufDims" and "ForceBufDims" in this chapter.

---

**Important**

SetCharExtra uses FIXED values. You can use the Integer Math Tool Set routine FixRatio to convert values to FIXED values.

---

**Parameters**

**Stack before call**

\[
\begin{array}{c}
\text{previous contents} \\
\text{charExtra} \quad \text{Long—FIXED; value for chExtra field} \\
\end{array}
\]

**Stack after call**

\[
\begin{array}{c}
\text{previous contents} \\
\text{SP} \\
\end{array}
\]

**Errors**

None

**C**

```
extern pascal void SetCharExtra(charExtra)

Fixed charExtra;
```
SetClip

Copies a specified region into the clipping region. The handle to the clipping region is not changed.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgnHandle</td>
</tr>
</tbody>
</table>

Long—HANDLE to region

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

← SP

Errors

Memory Manager errors Returned unchanged

C

extern pascal void SetClip(rgnHandle)

RgnHandle rgnHandle;
SetClipHandle

Sets the clipRgn handle field in the GrafPort to a specified value.

Parameters

Stack before call

\[ \text{previous contents} \]

\[ \text{rgnHandle} \]

\[ \text{Long—HANDLE to clipping region} \]

\[ \leftarrow \text{SP} \]

Stack after call

\[ \text{previous contents} \]

\[ \leftarrow \text{SP} \]

Errors

None

C

\[ \text{extern pascal void SetClipHandle(rgnHandle)} \]

\[ \text{RgnHandle rgnHandle;} \]
SetColorEntry

Sets the value of a color in a specified color table.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>tableNumber</td>
</tr>
<tr>
<td>entryNumber</td>
</tr>
<tr>
<td>newColor</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Errors

$0450$  badTableNum  Invalid table number; 0 to 15 are valid

$0451$  badColorNum  Invalid color number; 0 to 15 are valid

C

extern pascal void SetColorEntry(tableNumber, entryNumber, newColor)

Word  tableNumber;
Word  entryNumber;
ColorValue  newColor;
Sets a specified color table to specified values. The 16 color tables are stored starting at $9E00. Each table takes $20 bytes. Each word in the table represents one of 4,096 colors. The high-order nibble of the high-order byte is ignored.

Parameters

Stack before call

| previous contents | Word—INTEGER; number of table whose color values will be set |
| tableNumber       | Long—POINTER to color table |
| srcTablePtr       | ← SP |

Stack after call

| previous contents | ← SP |

Errors

$0450 badTableNum Invalid table number; 0 to 15 are valid

C

extern pascal void SetColorTable(tableNumber, srcTablePtr)
Word     tableNumber;
ColorTable srcTablePtr;
SetCursor

Sets the cursor to an image passed in a specified cursor record. If the cursor is hidden, it remains hidden and appears in the new form when it becomes visible again. If the cursor is visible, it appears in the new form immediately. See the section "Cursors" in this chapter.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursorPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to cursor record

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>
| ← SP

Errors

Memory Manager errors Returned unchanged

C

extern pascal void SetCursor(cursorPtr)

Pointer cursorPtr;
SetEmptyRgn

Destroys previous region information by setting a specified region to an empty region. The empty region for this purpose is a rectangular region with a bounding box of (0,0,0,0). If the original region was not rectangular, the region is resized.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgnHandle</td>
</tr>
</tbody>
</table>

Long—HANDLE to region being modified

Stack after call

| previous contents |

Errors

Memory Manager errors Returned unchanged

C

extern pascal void SetEmptyRgn(rgnHandle)

RgnHandle rgnHandle;
Sets the current font to a specified font. The call also zeros out the GrafPort's `fontID` field. After the call, you can set the font ID to anything desired by using the QuickDraw II routine `SetFontID` (see the section "SetFontID" in this chapter).

**Important**
Under most circumstances, your application should work with the Font Manager to set the font it needs. Use this call only if your application is handling all font manipulation by itself.

### Parameters

#### Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>newFontHandle</td>
</tr>
</tbody>
</table>

- `newFontHandle` - HANDLE to font

#### Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

- `- SP`

### Errors

None

### C

```c
extern pascal void SetFont(newFontHandle)
FontHndl newFontHandle;
```
$9804 \textbf{SetFontFlags}

Sets the font flags word to a specified value. The font flags word is used to indicate special operations performed on the text. At the time of publication, two flags are defined. See Figure 16-37 for the available values for those flags.

---

Important
The \texttt{chExtra} and \texttt{spExtra} values, and changes to character width, are applied after the character widths are fixed by the \texttt{fontFlags} setting.

---

Parameters

Stack before call

\[
\begin{array}{c|c|c}
\hline
\text{previous contents} & \text{Word—INTEGER; font flags (see Figure 16-37)} \\
\hline
\text{fontFlags} & \leftarrow \text{SP} \\
\hline
\end{array}
\]

Stack after call

\[
\begin{array}{c|c|c}
\hline
\text{previous contents} & \leftarrow \text{SP} \\
\hline
\end{array}
\]

Errors \hspace{1cm} \text{None}

C

\begin{verbatim}
extern pascal void SetFontFlags(fontFlags)
Word fontFlags;
\end{verbatim}

(continued)
Font flags

The values available for `fontFlags` are shown in Figure 16-37.

![Figure 16-37](image)

- **Reserved**: set to 0
- **Proportional font**: 00
- **Nonproportional spacing**: 01
- **Numeric spacing**: 10
- **Invalid value**: 11

If bits 1-0 are set to 00, the font is considered to be a proportionally spaced font.

If bits 1-0 are set to 01, the font is considered to be a fixed-width font (rather than a proportional font), all characters will be equally spaced, and the width of each character will be that of the `widMax` field in the `fontinfo` record.

If bits 1-0 are set to 10, the font is considered to be a fixed-width font, all characters will be equally spaced, and the width of each character will be equal to the character width of the font's 0 (zero) character. This feature makes it easier to line up columns of numbers. Because the width used in numeric spacing is usually less than `widMax`, some characters—for instance, W's and M's—end up overlapping other characters. Consequently, numeric spacing is useful with the characters most commonly used with numbers, such as the space character or the period, but is not appropriate for general text.

If bit 2 is set to 0, the foreground and background colors are treated as pixel values; that is, either as a two- or four-bit number depending on the GrafPort's SCB. The other bits in the word are ignored. Each foreground pixel is given the value of the foreground color value, and each background pixel is given the value of the background color value. For example, in 640 mode with a foreground color word of 0110011001100110 and bit 2 set to 0, each pixel will have a value of 10.

If bit 2 is set to 1, the foreground and background colors are treated as a word's worth of pixel values. This feature is useful when you are trying to draw text in 640 mode using dithered colors. Each foreground pixel in a destination word is given the value of the corresponding pixel in the foreground color word. Each background pixel in a destination word is given the value of the corresponding pixel in the background color word. For example, in 640 mode with a foreground color word of 0110011001100110 and bit 2 set to 1, odd-numbered pixels will have a value of 10 and even-numbered pixels will have a value of 01.
$D004  SetFontID

Sets the fontID field in the GrafPort. This routine does not change the current font.
SetFontID is designed for use by the Font Manager for the benefit of the picture routines. The picture routines use the font ID to try to find the font the application really wanted to draw with, rather than the one that was available when the picture was recorded.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Long—Font ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>fontID</td>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

| previous contents                      | ← SP         |

Errors

None

C

extern pascal void SetFontID(fontID)

FontID    fontID;
SetForeColor

Sets the \texttt{fgColor} (foreground color) field in the GrafPort to a specified value. Foreground color has either a two- or four-bit value, depending on the port SCB. If the port SCB indicates 320 mode, the lower four bits of \texttt{foreColor} are used. If the port SCB indicates 640 mode, the lower two bits of \texttt{foreColor} are used.

**Parameters**

**Stack before call**

| previous contents | Word—INTEGER; foreground color |
| foreColor         | ← SP |

**Stack after call**

| previous contents | ← SP |

**Errors**

None

**C**

```c
extern pascal void SetForeColor(foreColor)

Word foreColor;
```
SetGrafProcs

Sets the grafProcs field of the current GrafPort to a specified value.

Parameters

Stack before call

| previous contents | grafProcsPtr | Long—POINTER to grafProcs record
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SP</td>
<td></td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>SP</th>
</tr>
</thead>
</table>

Errors

None

C

extern pascal void SetGrafProcs(grafProcsPtr)
QDProcsPtr grafProcsPtr;
$B604  SetIntUse

Indicates to the cursor drawing code whether the code should use scan line interrupts. QuickDraw II normally uses scan line interrupts to draw the cursor without flicker. If an application wants to use scan line interrupts for some process of its own, it must tell QuickDraw II not to use them.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th></th>
<th>Word—BOOLEAN; TRUE for cursor to use scan line interrupts, FALSE for cursor to not use scan line interrupts</th>
</tr>
</thead>
<tbody>
<tr>
<td>useInt</td>
<td>← SP</td>
<td></td>
</tr>
</tbody>
</table>

Stack after call

| previous contents | ← SP |

Errors

None

C

extern pascal void SetIntUse(useInt)

Word useInt;
$1604 \textbf{SetMasterSCB}

Sets the master SCB to a specified value. The master SCB is the global mode byte used throughout QuickDraw II. It is used by routines such as InitPort to decide what standard values should be put into the GrafPort.

**Parameters**

**Stack before call**

| \textit{previous contents} | \textit{masterSCB} | Word—Value for master SCB (low-order byte only; high-order byte = 0) | SP |

**Stack after call**

| \textit{previous contents} | SP |

**Errors**

None

**C**

```c
extern pascal void SetMasterSCB(masterSCB)

Word masterSCB;
```
$2304 \textbf{SetOrigin}

Adjusts the contents of the port rectangle and the bounds rectangle so the upper left corner of the port rectangle is set to the specified point. The visible region is also affected, but the clipping region is not. The pen position does not change.

\textbf{Parameters}

\textbf{Stack before call}

\begin{tabular}{|c|}
\hline
\textit{previous contents} \\
\hline
\textit{h} \hspace{1cm} \textbf{Word}—INTEGER; horizontal coordinate of upper left corner \\
\hline
\textit{v} \hspace{1cm} \textbf{Word}—INTEGER; vertical coordinate of upper left corner \\
\hline
\end{tabular}

\textbf{Stack after call}

\begin{tabular}{|c|}
\hline
\textit{previous contents} \\
\hline
\textbf{SP} \hspace{1cm} \textbf{SP} \\
\hline
\end{tabular}

\textbf{Errors}

Memory Manager errors Returned unchanged

\textbf{C}

\texttt{extern pascal void SetOrigin(h,v)}

\begin{verbatim}
Integer h;
Integer v;
You can also use the following alternate form of the call:
\end{verbatim}

\texttt{extern pascal void SetOrigin(point)}

\begin{verbatim}
Point point;
\end{verbatim}

16-232 \textit{QuickDraw II routines}
SetPenMask

Sets the pen mask to a specified mask.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>maskPtr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long—POINTER to pen mask</td>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

| previous contents | ← SP |

Errors

None

C

extern pascal void SetPenMask(maskPtr)
Mask maskPtr;
$2E04 \textbf{SetPenMode} \\
Sets the current pen mode to a specified pen mode, as shown in Table 16-9.

\textbf{Parameters}

\textbf{Stack before call}

\begin{verbatim}
previous contents
\hline
penMode
\end{verbatim}

Word—INTEGER; new pen mode (see Table 16-9) \textarrow{SP}

\textbf{Stack after call}

\begin{verbatim}
previous contents
\end{verbatim}

\textarrow{SP}

\textbf{Errors}

None

\textbf{C}

\begin{verbatim}
extern pascal void SetPenMode(penMode)

Word penMode;
\end{verbatim}

\textbf{Pen modes}

Table 16-9 shows the available pen modes. Each 1 and 0 is the value of a bit in a pixel.

\textbullet \textit{Note:} Special text modes are also available. See Table 16-10 in the section "SetTextMode" in this chapter for those modes.
<table>
<thead>
<tr>
<th>Integer</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000</td>
<td>modeCopy</td>
<td>Copy source (or not source) to destination. modeCopy is the typical drawing mode. For text, the fully colored text pixels (both foreground and background) are copied into the destination.</td>
</tr>
<tr>
<td>$0001</td>
<td>modeOR</td>
<td>Overlay (OR) source (or not source) and destination. Use modeOR to nondestructively overlay new images on top of existing images; use notOR to overlay inverted images. For text, the fully colored text pixels (both foreground and background) are ORed with the destination.</td>
</tr>
<tr>
<td>$0002</td>
<td>modeXOR</td>
<td>Exclusive or (XOR) pen with destination. Use these modes for cursor drawing and rubber-banding. If an image is drawn in XOR mode, the appearance of the destination at the image location can be restored by drawing the image again in XOR mode. For text, the fully colored text pixels (both foreground and background) are XORed with the destination.</td>
</tr>
<tr>
<td>$0003</td>
<td>modeBIC</td>
<td>Bit Clear (BIC) pen with destination ((NOT pen) AND destination). Use this mode to explicitly erase (turn off) pixels, often prior to overlaying another image. You can use notBIC to display the intersection of two images. For text, the fully colored text pixels (both foreground and background) are BICed with the destination.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>modeCopy</th>
<th>Pen</th>
<th>notCopy</th>
<th>Pen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>modeOR</th>
<th>Pen</th>
<th>notOR</th>
<th>Pen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>modeXOR</th>
<th>Pen</th>
<th>notXOR</th>
<th>Pen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>modeBIC</th>
<th>Pen</th>
<th>notBIC</th>
<th>Pen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

QuickDraw II routines 16-235
Sets the current pen pattern to a specified pen pattern.

**Parameters**

**Stack before call**

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>patternPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to pattern
```

**Stack after call**

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

← SP
```

**Errors**

None

**C**

```c
extern pascal void SetPenPat(patternPtr)
Pattern patternPtr;
```
$2C04  \textbf{SetPenSize}

Sets the current pen size to a specified pen size.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>penWidth</td>
</tr>
<tr>
<td>penHeight</td>
</tr>
</tbody>
</table>

| ← SP |

**Stack after call**

| previous contents |
|← SP |

**Errors**

None

**C**

\[
\text{extern pascal void SetPenSize(penWidth, penHeight)}
\]

\[
\text{Word penWidth;}
\]

\[
\text{Word penHeight;}
\]
**$2A04 SetPenState**

Sets the pen state in the GrafPort to specified values.

**Parameters**

**Stack before call**

| previous contents | \[\text{Long} \rightarrow \text{pointer to pen state record (see Figure 16-38)} \]  
| penStatePtr \[\leftarrow \text{SP} \] |

**Stack after call**

| previous contents | \[\leftarrow \text{SP} \] |

**Errors**

None

**C**

```c
extern pascal void SetPenState(penStatePtr)
penStatePtr penStatePtr;
```

**Pen state record**

The record pointed to by `penStatePtr` is formatted as shown in Figure 16-38.

**Figure 16-38**

Pen state record

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>[\text{psPrSize} ]</td>
<td>Long—POINT specifying pen size</td>
</tr>
<tr>
<td>1</td>
<td>[\text{psPrMode} ]</td>
<td>Word—Pen mode</td>
</tr>
<tr>
<td>2</td>
<td>[\text{psPrPat} ]</td>
<td>32 bytes—Pen pattern</td>
</tr>
<tr>
<td>5</td>
<td>[\text{psPrMask} ]</td>
<td>8 bytes—Pen mask</td>
</tr>
<tr>
<td>25</td>
<td>[\text{psPrPat} ]</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>[\text{psPrMask} ]</td>
<td></td>
</tr>
</tbody>
</table>

16-238 QuickDraw II routines
SetPicSave

Sets the picSave field in the GrafPort to a specified value.

Warning
This is an internal routine that should not be used by application programs.

Parameters

Stack before call

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>picSaveValue</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

Stack after call

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

Errors
None

C

```c
extern pascal void SetPicSave(picSaveValue)
Longint      picSaveValue;
```
$4204 \quad \textbf{SetPolySave} \\
Sets the \textit{polySave} field in the GrafPort to a specified value.

\begin{center}
\begin{tabular}{|c|}
\hline
\textbf{Warning} \\
This is an internal routine that should not be used by application programs. \\
\hline
\end{tabular}
\end{center}

\textbf{Parameters}

\begin{center}
\begin{tabular}{|c|}
\hline
\textbf{Stack before call} \\
\begin{tabular}{|c|}
\hline
\textit{previous contents} \\
\textit{polySaveValue} \quad \textbf{Long}—New value for \textit{polySave} field \\
\hline
\end{tabular} \\
\textbf{SP} \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{|c|}
\hline
\textbf{Stack after call} \\
\begin{tabular}{|c|}
\hline
\textit{previous contents} \\
\hline
\end{tabular} \\
\textbf{SP} \\
\hline
\end{tabular}
\end{center}

\textbf{Errors} \quad \text{None}

\textbf{C} \quad \text{extern pascal void SetPolySave(polySaveValue)}
\begin{verbatim}
Longint polySaveValue;
\end{verbatim}

16-240 \quad \text{QuickDraw II routines}
SetPort

Makes a specified port the current GrafPort.

Parameters

Stack before call

\[
\begin{array}{|c|}
\hline
\text{previous contents} \\
\hline
\text{Long—POINTER to port} \\
\hline
\end{array}
\]

\[← SP\]

Stack after call

\[
\begin{array}{|c|}
\hline
\text{previous contents} \\
\hline
\hline
\end{array}
\]

\[← SP\]

Errors

None

C

\[
\text{extern pascal void SetPort(portPtr)} \\
\text{GrafPortPtr portPtr;}
\]
$1D04 \textbf{SetPortLoc}\\
Sets the current port's \textit{locInfo} record to specified location information.

\textbf{Parameters}

\textbf{Stack before call}
\begin{center}
\begin{tabular}{c|c|c}
\hline
\textit{previous contents} & \textit{locInfoPtr} & \texttt{Long-POINTER to location information} \\
\hline
\end{tabular}
\end{center}
\begin{center}
\texttt{SP}
\end{center}

\textbf{Stack after call}
\begin{center}
\begin{tabular}{c|c|c}
\hline
\textit{previous contents} & \texttt{SP} \\
\hline
\end{tabular}
\end{center}

\textbf{Errors} \quad \text{None}

\textbf{C} \\
\texttt{extern pascal void SetPortLoc(locInfoPtr)}

\texttt{LocInfoPtr locInfoPtr;}

\textbf{16-242 QuickDraw II routines}
SetPortRect

Sets the current GrafPort's port rectangle to the specified rectangle.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rectPtr</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Errors

None

C

extern pascal void SetPortRect(rectPtr)
Rect *rectPtr;
**SetPortSize**

Changes the size of the current GrafPort’s port rectangle. The routine does not affect the pixel image; it just changes the active area of the GrafPort. Normally, the call is made only by the Window Manager.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>portWidth</th>
<th>Word—INTEGER; width of active area in pixels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>portHeight</td>
<td>Word—INTEGER; height of active area in pixels</td>
</tr>
<tr>
<td></td>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>← SP</th>
</tr>
</thead>
</table>

**Errors**

None

**C**

```c
extern pascal void SetPortSize(portWidth, portHeight)

Word    portWidth;

Word    portHeight;
```
$8204 \textbf{SetPt}\\
Sets a point to specified horizontal and vertical values.

\textbf{Parameters}

\textbf{Stack before call}\\
\begin{tabular}{l|l}
\textit{previous contents} & \\
\hline
srcPtPtr & \textbf{Long}--\text{POINTER to POINT} \\
\hline
h & \textbf{Word}--\text{INTEGER}; horizontal value of point \\
v & \textbf{Word}--\text{INTEGER}; vertical value of point \\
\hline
\end{tabular}

\textbf{Stack after call}\\
\begin{tabular}{l|l}
\textit{previous contents} & \\
\hline
\hline
\end{tabular}

\textbf{Errors}\\
None

\textbf{C}\\
\begin{verbatim}
extern pascal void SetPt(srcPtPtr, h, v)\\
Point *srcPtPtr;
Integer h;
Integer v;
\end{verbatim}
$8704  SetRandSeed

Sets the seed value for the random number generator. The algorithm uses a 32-bit seed to produce a 16-bit random number. See the section "Random" in this chapter.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>randomSeed</td>
</tr>
<tr>
<td>Long--LONGINT; seed value</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Errors

None

C

```
extern pascal void SetRandSeed(randomSeed)

Longint   randomSeed;
```
SetRect

Sets a specified rectangle to specified values.

Parameters

Stack before call

previously
contents

rectPtr
left
top
right
bottom

Long—POINTER to space for RECT defining rectangle to be set
Word—INTEGER; left X coordinate for rectangle
Word—INTEGER; top Y coordinate for rectangle
Word—INTEGER; right X coordinate for rectangle
Word—INTEGER; bottom Y coordinate for rectangle
←SP

Stack after call

previously
contents
←SP

Errors

None

C

extern pascal void SetRect(rectPtr, left, top, right, bottom)

Rect *rectPtr;
Integer    left;
Integer    top;
Integer    right;
Integer    bottom;
$6B04 \textbf{SetRectRgn} \newline
Destroys previous region information by setting a specified region to a specified rectangle. If the inputs do not describe a valid rectangle, the region is set to the empty region. If the original region was not rectangular, the region is resized.

\textbf{Parameters}

\textbf{Stack before call}

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgnHandle</td>
</tr>
<tr>
<td>left</td>
</tr>
<tr>
<td>top</td>
</tr>
<tr>
<td>right</td>
</tr>
<tr>
<td>bottom</td>
</tr>
</tbody>
</table>

- **Long**—HANDLE to region being set
- **Word**—INTEGER; left X coordinate for rectangle
- **Word**—INTEGER; top Y coordinate for rectangle
- **Word**—INTEGER; right X coordinate for rectangle
- **Word**—INTEGER; bottom Y coordinate for rectangle

\textbf{Stack after call}

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

\textbf{Errors}

- Memory Manager errors
- Returned unchanged

\textbf{C}

```c
extern pascal void SetRectRgn(rgnHandle, left, top, right, bottom)
RgnHandle    rgnHandle;
Integer      left;
Integer      top;
Integer      right;
Integer      bottom;
```
SetRgnSave

Sets the rgnSave field in the GrafPort to a specified value.

Warning
This is an internal routine that should not be used by application programs.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgnSaveValue</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

Errors

None

C

extern pascal void SetRgnSave(rgnSaveValue)

Handle   rgnSaveValue;
SetSCB

Sets the SCB (scan line control byte) to a specified value.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>scanLine</td>
</tr>
<tr>
<td>newSCB</td>
</tr>
</tbody>
</table>

<- SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

<- SP

Errors

$0452 badScanLine Invalid scan line number; 0 to 199 are valid

C

extern pascal void SetSCB(scanLine, newSCB)

Word scanLine;

Word newSCB;
SetSolidBackPat

Sets the background pattern to a solid pattern using a specified color. Only an appropriate number of bits in colorNum are used. If the port SCB indicates 320 mode, four bits are used; if it indicates 640 mode, two bits are used.

Parameters

Stack before call

| previous contents | colorNum | Word—INTEGER; new color value | ← SP |

Stack after call

| previous contents | ← SP |

Errors

None

C

extern pascal void SetSolidBackPat(colorNum)

Word colorNum;
SetSolidPenPat

Sets the pen pattern to a solid pattern using the specified color. Only an appropriate number of bits in colorNum are used. If the port SCB indicates 320 mode, four bits are used; if it indicates 640 mode, two bits are used.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>colorNum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Word—INTEGER; new color value</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Errors

None

C

extern pascal void SetSolidPenPat(colorNum)
Word colorNum;

16-252 QuickDraw II routines
SetSpaceExtra

Sets the \textit{spExtra} field in the GrafPort to a specified value. The \textit{spExtra} field is used by programs that are trying to justify text to a left and right boundary. When the \textit{spExtra} field is nonzero, its value is added to the width of each space printed in a string.

\textbf{Important}

SetSpaceExtra uses FIXED values. You can use the Integer Math Tool Set routine FixRatio to convert values to FIXED values.

\textbf{Parameters}

\textbf{Stack before call}

\begin{center}
\begin{tabular}{c|c}
\textit{previous contents} & \textit{Long-FIXED; new value for \textit{spExtra} field} \\
\hline
\textit{spaceExtra} & \textit{SP} \\
\end{tabular}
\end{center}

\textbf{Stack after call}

\begin{center}
\begin{tabular}{c|c}
\textit{previous contents} & \textit{SP} \\
\end{tabular}
\end{center}

\textbf{Errors}

None

\textbf{C}

\begin{verbatim}
extern pascal void SetSpaceExtra(spaceExtra)
Fixed spaceExtra;
\end{verbatim}

\textbf{A justifying example}

You want to display the words \textit{a quick brown fox} and left- and right-justify them in a rectangle that measures 200 pixels across. You measure the string and find it to be 193 pixels long. The string has 3 spaces between words, so you divide 3 into the 7 pixels remaining (200 \(-\) 193 \(=\) 7). Thus, you set \textit{spExtra} to \(2 \ 1/3\) \((7 + 3 = 2 \ 1/3)\).
$8D04  **SetStdProcs**  
Sets up a specified record of pointers for customizing QuickDraw II operations. At the time of publication, more details were unavailable.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Long—POINTER to stdProcs record</th>
</tr>
</thead>
<tbody>
<tr>
<td>stdProcRecPtr</td>
<td>← SP</td>
</tr>
</tbody>
</table>

**Stack after call**

| previous contents | ← SP |

**Errors**

None

**C**

```c
extern pascal void SetStdProcs(stdProcRecPtr)
QDProcsPtr stdProcRecPtr;
```
$4804 \textbf{SetSysField}

Sets the \textit{sysField} field in the GrafPort to a specified value.

\underline{Warning}
This is an internal routine that should not be used by application programs.

\textbf{Parameters}

\textbf{Stack before call}

\begin{center}

\begin{tabular}{l|c|c}
\hline
\textit{previous contents} & \textit{sysFieldValue} & \textit{Long} - New value for \textit{sysField} field \\
\hline
\end{tabular}
\end{center}

\rightarrow \text{SP}

\textbf{Stack after call}

\begin{center}

\begin{tabular}{l|c|c}
\hline
\textit{previous contents} & \rightarrow \text{SP} \\
\hline
\end{tabular}
\end{center}

\textbf{Errors}

None

\textbf{C}

extern pascal void SetSysField(sysFieldValue)

Longint sysFieldValue;
$B204 \textbf{SetSysFont}

Sets a specified font as the system font. The default system font is used unless this call is made. A handle to the system font is placed in the \texttt{fontHandle} field of each GrafPort when it is opened or initialized.

**Parameters**

**Stack before call**

\begin{align*}
\text{previous contents} & \quad \textbf{fontHandle} \quad \text{Long}-\text{HANDLE to font that will be system font} \\
& \leftarrow \text{SP}
\end{align*}

**Stack after call**

\begin{align*}
\text{previous contents} & \leftarrow \text{SP}
\end{align*}

**Errors**

None

**C**

\begin{verbatim}
extern pascal void SetSysFont(fontHandle)
FontHndl   fontHandle;
\end{verbatim}
$9A04 SetTextFace

Sets the text face to a specified value. Up to 16 operations on the text are possible. Each bit in textFace represents a different face, as shown in Figure 16-39.

Parameters

Stack before call


previous contents


textFace


Word—INTEGER; text face (see Figure 16-39)

← SP

Stack after call


previous contents


← SP

Errors

None

C

extern pascal void SetTextFace(textFace)

TextStyle textFace;

(continued)
Text face flag

The bit values for the `textFace` parameter are shown in Figure 16-39.

---

**Important**

Shadow, outline, and italic styles are available only if QuickDraw II Auxiliary has been loaded and started up. Also, fonts that have a descent value of less than 2 will not be underlined.

---

Figure 16-39
Text face flag
Sets the text mode to a specified value. There are eight text-only modes (four modes and their opposites), as shown in Table 16-10. The fastest modes are the modes that only transfer the foreground to the destination. The fastest of the foreground modes are mode ForeOR and mode ForeXOR; mode ForeBIC is almost as fast, and mode ForeCOPY is the slowest.

In addition to the text-only modes, the pen modes apply to text. See the section "SetPenMode" in this chapter.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Word—INTEGER; text mode (see Table 16-10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>textMode</td>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

| previous contents | ← SP                                      |

Errors

None

C

extern pascal void SetTextMode(textMode)

Word textMode;

(continued)
**Text modes**

The modes shown in Table 16-10 are used only for text. They apply when drawing from a one-bit-per-pixel world to a two- or four-bit-per-pixel world. You need this routine only when drawing from the font to a destination pixel map.

### Table 16-10

**Text-only modes**

<table>
<thead>
<tr>
<th>Integer</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0004</td>
<td>modeForeCopy</td>
<td>Copies only the foreground pixels into the destination; background pixels are not altered</td>
</tr>
<tr>
<td>$8004</td>
<td>notForeCOPY</td>
<td>Same as modeForeCopy, except that foreground pixels are turned to background pixels and background pixels are turned to foreground pixels before the operation is performed</td>
</tr>
<tr>
<td>$0005</td>
<td>modeForeOR</td>
<td>ORs only the foreground pixels into the destination; background pixels are not altered</td>
</tr>
<tr>
<td>$8005</td>
<td>notForeOR</td>
<td>Same as modeForeOR, except that foreground pixels are turned to background pixels and background pixels are turned to foreground pixels before the operation is performed</td>
</tr>
<tr>
<td>$0006</td>
<td>modeForeXOR</td>
<td>XORs only the foreground pixels into the destination; background pixels are not altered</td>
</tr>
<tr>
<td>$8006</td>
<td>notForeXOR</td>
<td>Same as modeForeXOR, except that foreground pixels are turned to background pixels and background pixels are turned to foreground pixels before the operation is performed</td>
</tr>
<tr>
<td>$0007</td>
<td>modeForeBIC</td>
<td>BICs only the foreground pixels into the destination; background pixels are not altered</td>
</tr>
<tr>
<td>$8007</td>
<td>notForeBIC</td>
<td>Same as modeForeBIC, except that foreground pixels are turned to background pixels and background pixels are turned to foreground pixels before the operation is performed</td>
</tr>
</tbody>
</table>

---

16-260  QuickDraw II routines
**$D204 SetTextSize**

Sets the `txSize` field of the GrafPort to a specified value.

**Parameters**

Stack before call

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>textSize</td>
</tr>
</tbody>
</table>
```

Stack after call

```
| previous contents |
```

**Errors**

None

**C**

```
extern pascal void SetTextSize(textSize)

Word    textSize;
```
Sets the userField field in the GrafPort to a specified value. Your application can attach data to a GrafPort by using this field as a pointer to some other data area.

Parameters

Stack before call

```
previous contents
--- userFieldValue --- Long—New value for userField field
```

Stack after call

```
previous contents
```

Errors

None

C

```
extern pascal void SetUserField(userFieldValue)
Longint userFieldValue;
```
SetVisHandle

Sets the visRgn field in the GrafPort to a specified value.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- rgnHandle --</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Long—HANDLE to visible region

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

← SP

Errors

None

C

extern pascal void SetVisHandle(rgnHandle)

RgnHandle    rgnHandle;
$B404  SetVisRgn

Copies a specified region into the visible region (but does not change the visRgn field of the GrafPort).

Parameters

Stack before call

\[
\begin{array}{c|c}
\text{previous contents} & \text{Long—HANDLE to region} \\
\hline
\text{rgnHandle} & \leftarrow \text{SP}
\end{array}
\]

Stack after call

\[
\begin{array}{c|c}
\text{previous contents} & \leftarrow \text{SP}
\end{array}
\]

Errors

None

C

extern pascal void SetVisRgn(rgnHandle)

RgnHandle rgnHandle;

$9104  ShowCursor

Shows the cursor by incrementing the cursor level (if the level is already 0, it is not incremented). A cursor level of 0 indicates the cursor is visible; a cursor level of less than 0 indicates the cursor is not visible.

Parameters

The stack is not affected by this call. There are no input or output parameters.

Errors

None

C

extern pascal void ShowCursor()
ShowPen

Increments the pen level. A positive pen level indicates that drawing will occur; a negative pen level indicates that drawing will not occur.

Parameters

The stack is not affected by this call. There are no input or output parameters.

Errors

None

C

extern pascal void ShowPen()

SolidPattern

Sets a specified pattern to a solid pattern using a specified color. Only an appropriate number of bits in colorNum are used. If the port SCB indicates 320 mode, four bits are used; if it indicates 640 mode, two bits are used.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>colorNum</td>
</tr>
<tr>
<td>patternPtr</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

Errors

None

C

extern pascal void SolidPattern(colorNum,patternPtr)

Word    colorNum;
Pattern  patternPtr;

QuickDraw II routines 16-265
$AD04 StringBounds

Puts the string bounds rectangle of a specified Pascal-type string into a specified buffer.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>stringPtr</td>
</tr>
<tr>
<td>resultPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to Pascal-type string
Long—POINTER to space for RECT defining rectangle

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

<- SP

Errors
None

C

```c
extern pascal void StringBounds(stringPtr,resultPtr)

Pointer   stringPtr;
Rect *resultPtr;
```
$A904  **StringWidth**

Returns the sum of all the character widths, in pixels (pen displacements), of a specified Pascal-type string. This would be the pen displacement if the string were to be drawn.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
<td>Word—Space for result</td>
</tr>
<tr>
<td>-- stringPtr</td>
<td>Long—POINTER to Pascal-type string</td>
</tr>
<tr>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>stringWidth</td>
<td>Word—INTEGER; width of string in pixels</td>
</tr>
<tr>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

None

**C**

extern pascal Integer StringWidth(stringPtr)

Pointer stringPtr;

QuickDraw II routines 16-267
SubPt

Subtracts the source point from the destination point and leaves the result in the destination point. For example, a source point of (1,2) and a destination point of (10,20) result in a destination point of (9,18).

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcPtPtr</td>
</tr>
<tr>
<td>destPtPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to POINT

Long—POINTER to POINT used as source and destination

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

← SP

Errors

None

C

extern pascal void SubPt(srcPtPtr, destPtPtr)

Point *srcPtPtr;
Point *destPtPtr;

QuickDraw II routines
TextBounds

Puts the character bounds rectangle of specified text into a specified buffer.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>textPtr</td>
</tr>
<tr>
<td>textLength</td>
</tr>
<tr>
<td>resultPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to text

Word—INTEGER; length of text in bytes

Long—POINTER to space for RECT defining rectangle

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

errors

None

C

extern pascal void TextBounds(textPtr, textLength, resultPtr)

Pointer textPtr;

Word textLength;

Rect *resultPtr;
$AB04 \quad TextWidth$

Returns the character width, in pixels (pen displacement), of specified text.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
</tr>
<tr>
<td>textPtr</td>
</tr>
<tr>
<td>textLength</td>
</tr>
</tbody>
</table>

- **Word**—Space for result
- **Long**—POINTER to text
- **Word**—INTEGER; length of text in bytes

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>textWidth</td>
</tr>
</tbody>
</table>

- **Word**—INTEGER; width of text in pixels

**Errors**

None

**C**

```c
extern pascal Integer TextWidth(textPtr, textLength)

Pointer textPtr;
Word textLength;
```
$4E04 \quad \textbf{UnionRect}

Calculates the smallest rectangle that contains both source rectangles and places the result in a destination rectangle.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rect1Ptr</td>
</tr>
<tr>
<td>rect2Ptr</td>
</tr>
<tr>
<td>unionRectPtr</td>
</tr>
</tbody>
</table>

*Long*-POINTER to:
- first source rectangle
- second source rectangle
- destination rectangle

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
</tr>
</tbody>
</table>

**Errors**

None

**C**

```c
extern pascal void UnionRect(rect1Ptr, rect2Ptr, unionRectPtr)

Rect *rect1Ptr;
Rect *rect2Ptr;
Rect *unionRectPtr;
```
$7204 \textbf{UnionRgn}

Calculates the smallest region that contains every point that is in either source region and places the result in a destination region. The destination region (which may be one of the source regions) must already exist; UnionRgn does not allocate it.

If both regions are empty, the destination is set to an empty region.

\textbf{Parameters}

\textbf{Stack before call}

\begin{verbatim}
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgn1Handle</td>
</tr>
<tr>
<td>rgn2Handle</td>
</tr>
<tr>
<td>unionRgnHandle</td>
</tr>
</tbody>
</table>
\end{verbatim}

\textbf{Stack after call}

\begin{verbatim}
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
</tr>
</tbody>
</table>
\end{verbatim}

\textbf{Errors}

Memory Manager errors Returned unchanged

\begin{verbatim}
C
extern pascal void UnionRgn(rgn1Handle, rgn2Handle, unionRgnHandle);
RgnHandle      rgn1Handle;
RgnHandle      rgn2Handle;
RgnHandle      unionRgnHandle;
\end{verbatim}

16-272 \textbf{QuickDraw II routines}
Calculates the difference between the union and the intersection of two regions and places the result in a destination region. The destination region (which may be one of the source regions) must already exist; this routine does not allocate it.

If the regions are not coincident, the destination is set to an empty region.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rgn1Handle</code></td>
</tr>
<tr>
<td>Long—HANDLE to one source region</td>
</tr>
<tr>
<td><code>rgn2Handle</code></td>
</tr>
<tr>
<td>Long—HANDLE to another source region</td>
</tr>
<tr>
<td><code>xorRgnHandle</code></td>
</tr>
<tr>
<td>Long—HANDLE to destination region</td>
</tr>
</tbody>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

Memory Manager errors Returned unchanged

**C**

```c
extern pascal void XorRgn(rgn1Handle, rgn2Handle, xorRgnHandle)
RgnHandle     rgn1Handle;
RgnHandle     rgn2Handle;
RgnHandle     xorRgnHandle;
```
QuickDraw II summary

This section briefly summarizes the constants, data structures, and tool set error codes contained in QuickDraw II.

Important

These definitions are provided in the appropriate Interface file.

<table>
<thead>
<tr>
<th>Table 16-11 QuickDraw II constants</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
</tr>
<tr>
<td>------------------------</td>
</tr>
<tr>
<td>Color data</td>
</tr>
<tr>
<td>table320</td>
</tr>
<tr>
<td>table640</td>
</tr>
<tr>
<td>GrafPort sizes</td>
</tr>
<tr>
<td>maskSize</td>
</tr>
<tr>
<td>locSize</td>
</tr>
<tr>
<td>patsize</td>
</tr>
<tr>
<td>pnStateSize</td>
</tr>
<tr>
<td>portSize</td>
</tr>
<tr>
<td>Color masks</td>
</tr>
<tr>
<td>blueMask</td>
</tr>
<tr>
<td>greenMask</td>
</tr>
<tr>
<td>redMask</td>
</tr>
<tr>
<td>Font flags</td>
</tr>
<tr>
<td>widMaxSize</td>
</tr>
<tr>
<td>zeroSize</td>
</tr>
<tr>
<td>Master colors</td>
</tr>
<tr>
<td>black</td>
</tr>
<tr>
<td>blue</td>
</tr>
<tr>
<td>darkGreen320</td>
</tr>
<tr>
<td>green320</td>
</tr>
<tr>
<td>green640</td>
</tr>
<tr>
<td>lightBlue320</td>
</tr>
<tr>
<td>purple320</td>
</tr>
<tr>
<td>darkGray320</td>
</tr>
<tr>
<td>periwinkleBlue320</td>
</tr>
<tr>
<td>brown320</td>
</tr>
<tr>
<td>lightGray320</td>
</tr>
<tr>
<td>red320</td>
</tr>
</tbody>
</table>
### QuickDraw II constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master colors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lilac320</td>
<td>$0DAF</td>
<td>Works in 320 mode</td>
</tr>
<tr>
<td>red640</td>
<td>$0F00</td>
<td>Works in 640 mode</td>
</tr>
<tr>
<td>orange320</td>
<td>$0F70</td>
<td>Works in 320 mode</td>
</tr>
<tr>
<td>flesh320</td>
<td>$0FA9</td>
<td>Works in 320 mode</td>
</tr>
<tr>
<td>yellow</td>
<td>$0FF0</td>
<td>Works in 320 and 640 modes</td>
</tr>
<tr>
<td>white</td>
<td>$0FFF</td>
<td>Works in 320 and 640 modes</td>
</tr>
<tr>
<td>Pen modes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>modeCopy</td>
<td>$0000</td>
<td>Copy source to destination</td>
</tr>
<tr>
<td>modeOR</td>
<td>$0001</td>
<td>Overlay source and destination</td>
</tr>
<tr>
<td>modeXOR</td>
<td>$0002</td>
<td>XOR pen with destination</td>
</tr>
<tr>
<td>modeBIC</td>
<td>$0003</td>
<td>Bit Clear pen with destination</td>
</tr>
<tr>
<td>notCopy</td>
<td>$8000</td>
<td>Copy (not source) to destination</td>
</tr>
<tr>
<td>notOR</td>
<td>$8001</td>
<td>Overlay (not source) and destination</td>
</tr>
<tr>
<td>notXOR</td>
<td>$8002</td>
<td>XOR (not pen) with destination</td>
</tr>
<tr>
<td>notBIC</td>
<td>$8003</td>
<td>Bit Clear (not pen) with destination</td>
</tr>
<tr>
<td>Pen and text modes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>modeForeCopy</td>
<td>$0004</td>
<td>Copy foreground pixels into destination</td>
</tr>
<tr>
<td>modeForeOR</td>
<td>$0005</td>
<td>OR foreground pixels into destination</td>
</tr>
<tr>
<td>modeForeXOR</td>
<td>$0006</td>
<td>XOR foreground pixels into destination</td>
</tr>
<tr>
<td>modeForeBIC</td>
<td>$0007</td>
<td>BIC foreground pixels into destination</td>
</tr>
<tr>
<td>notForeCOPY</td>
<td>$8004</td>
<td>Turn background to foreground, then copy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>foreground pixels into destination</td>
</tr>
<tr>
<td>notForeOR</td>
<td>$8005</td>
<td>Turn background to foreground, then OR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>foreground pixels into destination</td>
</tr>
<tr>
<td>notForeXOR</td>
<td>$8006</td>
<td>Turn background to foreground, then XOR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>foreground pixels into destination</td>
</tr>
<tr>
<td>notForeBIC</td>
<td>$8007</td>
<td>Turn background to foreground, then BIC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>foreground pixels into destination</td>
</tr>
<tr>
<td>Mode for QDStartUp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mode320</td>
<td>$0</td>
<td>320 mode</td>
</tr>
<tr>
<td>mode640</td>
<td>$80</td>
<td>640 mode</td>
</tr>
<tr>
<td>SCB byte masks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>colorTable</td>
<td>$0F</td>
<td>Color table number</td>
</tr>
<tr>
<td>scbReserved</td>
<td>$10</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>scbFill</td>
<td>$20</td>
<td>Fill mode on</td>
</tr>
<tr>
<td>scbInterrupt</td>
<td>$40</td>
<td>Interrupt generated when scan line refreshed</td>
</tr>
<tr>
<td>scbColorMode</td>
<td>$80</td>
<td>640 mode on</td>
</tr>
</tbody>
</table>

(continued)
### Table 16-11 (continued)
**QuickDraw II constants**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Text styles</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boldMask</td>
<td>$0001</td>
<td>Mask for bold bit</td>
</tr>
<tr>
<td>italicMask</td>
<td>$0002</td>
<td>Mask for italic bit</td>
</tr>
<tr>
<td>underlineMask</td>
<td>$0004</td>
<td>Mask for underline bit</td>
</tr>
<tr>
<td>outlineMask</td>
<td>$0008</td>
<td>Mask for outline bit</td>
</tr>
<tr>
<td>shadowMask</td>
<td>$0010</td>
<td>Mask for shadow bit</td>
</tr>
</tbody>
</table>

### Table 16-12
**QuickDraw II data structures**

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BufDimRec (buffer sizing record)</td>
<td></td>
<td>Word</td>
<td>Application-defined maximum pixel image width</td>
</tr>
<tr>
<td>maxWidth</td>
<td>$0</td>
<td>Word</td>
<td>Current text buffer height in pixels</td>
</tr>
<tr>
<td>textBufferSize</td>
<td>$2</td>
<td>Word</td>
<td>Current width of text buffer in words</td>
</tr>
<tr>
<td>textBufferWords</td>
<td>$4</td>
<td>Word</td>
<td>Equal to maxFBRExtent used in call</td>
</tr>
<tr>
<td>fontWidth</td>
<td>$6</td>
<td>Word</td>
<td>Offset in number of words to Macintosh font part</td>
</tr>
<tr>
<td>Font (font record)</td>
<td></td>
<td>Word</td>
<td>Font family number</td>
</tr>
<tr>
<td>offseToMF</td>
<td>$0</td>
<td>Word</td>
<td>Offset in number of words to Macintosh font part</td>
</tr>
<tr>
<td>family</td>
<td>$02</td>
<td>Word</td>
<td>Style font was designed with</td>
</tr>
<tr>
<td>style</td>
<td>$04</td>
<td>TextStyle</td>
<td>Point size of font</td>
</tr>
<tr>
<td>version</td>
<td>$08</td>
<td>Word</td>
<td>Version number of font definition</td>
</tr>
<tr>
<td>maxFBRExtent</td>
<td>$0A</td>
<td>Word</td>
<td>Maximum horizontal distance, in pixels, to far edge of any foreground or background pixel of any character of font</td>
</tr>
<tr>
<td>FontGlobalsRecord</td>
<td></td>
<td>Word</td>
<td>Family number</td>
</tr>
<tr>
<td>fgFontID</td>
<td>$00</td>
<td>Word</td>
<td>Style font was designed with</td>
</tr>
<tr>
<td>fgStyle</td>
<td>$02</td>
<td>TextStyle</td>
<td>Point size of font</td>
</tr>
<tr>
<td>fgSize</td>
<td>$04</td>
<td>Word</td>
<td>Version number of font definition</td>
</tr>
<tr>
<td>fgVersion</td>
<td>$06</td>
<td>Word</td>
<td>Maximum character width of any character in font</td>
</tr>
<tr>
<td>fgWidMax</td>
<td>$08</td>
<td>Word</td>
<td>Maximum horizontal distance, in pixels, to far edge of any foreground or background pixel of any character of font</td>
</tr>
<tr>
<td>fgFBRExtent</td>
<td>$0A</td>
<td>Word</td>
<td>Number of pixels above base line in font rectangle</td>
</tr>
<tr>
<td>FontInfoRecord</td>
<td></td>
<td>Integer</td>
<td>Number of pixels below base line in font rectangle</td>
</tr>
<tr>
<td>ascent</td>
<td>$00</td>
<td>Integer</td>
<td>Maximum character width of any character in font</td>
</tr>
<tr>
<td>descent</td>
<td>$02</td>
<td>Integer</td>
<td>Recommended number of blank pixel rows between descent of one text line and ascent of the next</td>
</tr>
<tr>
<td>widMax</td>
<td>$04</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>leading</td>
<td>$06</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
### QuickDraw II data structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>GrafPort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>portInfo</td>
<td>$00</td>
<td>LocInfo</td>
<td>Location information</td>
</tr>
<tr>
<td>portRect</td>
<td>$10</td>
<td>Rect</td>
<td>Port rectangle</td>
</tr>
<tr>
<td>clipRgn</td>
<td>$18</td>
<td>RgnHandle</td>
<td>Handle to clipping region</td>
</tr>
<tr>
<td>visRgn</td>
<td>$1C</td>
<td>RgnHandle</td>
<td>Handle to visible region</td>
</tr>
<tr>
<td>bkPat</td>
<td>$20</td>
<td>Pattern</td>
<td>Background pattern</td>
</tr>
<tr>
<td>pnLoc</td>
<td>$40</td>
<td>Point</td>
<td>Pen location</td>
</tr>
<tr>
<td>pnSize</td>
<td>$44</td>
<td>Point</td>
<td>Pen size</td>
</tr>
<tr>
<td>pnMode</td>
<td>$48</td>
<td>Word</td>
<td>Pen mode</td>
</tr>
<tr>
<td>pnPat</td>
<td>$4A</td>
<td>Pattern</td>
<td>Pen pattern</td>
</tr>
<tr>
<td>pnMask</td>
<td>$6A</td>
<td>Mask</td>
<td>Pen mask</td>
</tr>
<tr>
<td>pnVis</td>
<td>$72</td>
<td>Word</td>
<td>Pen visibility</td>
</tr>
<tr>
<td>fontHandle</td>
<td>$74</td>
<td>FontHndl</td>
<td>Handle to font</td>
</tr>
<tr>
<td>fontID</td>
<td>$78</td>
<td>FontID</td>
<td>Font ID</td>
</tr>
<tr>
<td>fontFlags</td>
<td>$7C</td>
<td>Word</td>
<td>Font flags</td>
</tr>
<tr>
<td>txSize</td>
<td>$7E</td>
<td>Integer</td>
<td>Text size</td>
</tr>
<tr>
<td>txFace</td>
<td>$80</td>
<td>TextStyle</td>
<td>Text face</td>
</tr>
<tr>
<td>txMode</td>
<td>$82</td>
<td>Word</td>
<td>Text mode</td>
</tr>
<tr>
<td>spExtra</td>
<td>$84</td>
<td>Fixed</td>
<td>Value of space extra</td>
</tr>
<tr>
<td>chExtra</td>
<td>$88</td>
<td>Fixed</td>
<td>Value of char extra</td>
</tr>
<tr>
<td>fgColor</td>
<td>$8C</td>
<td>Word</td>
<td>Foreground color</td>
</tr>
<tr>
<td>bgColor</td>
<td>$8E</td>
<td>Word</td>
<td>Background color</td>
</tr>
<tr>
<td>picSave</td>
<td>$90</td>
<td>Handle</td>
<td>picSave</td>
</tr>
<tr>
<td>rgnSave</td>
<td>$94</td>
<td>Handle</td>
<td>rgnSave</td>
</tr>
<tr>
<td>polySave</td>
<td>$98</td>
<td>Handle</td>
<td>polySave</td>
</tr>
<tr>
<td>grafProcs</td>
<td>$9C</td>
<td>QdProcsPtr</td>
<td>Pointer to GrafProcs record</td>
</tr>
<tr>
<td>arcRot</td>
<td>$A0</td>
<td>Integer</td>
<td>arcRot</td>
</tr>
<tr>
<td>userField</td>
<td>$A2</td>
<td>Longint</td>
<td>userField</td>
</tr>
<tr>
<td>sysField</td>
<td>$A6</td>
<td>Longint</td>
<td>sysField</td>
</tr>
<tr>
<td>LocInfo</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>portSCB</td>
<td>$00</td>
<td>Word</td>
<td>SCB byte</td>
</tr>
<tr>
<td>ptrToPixImage</td>
<td>$02</td>
<td>Pointer</td>
<td>Pointer to pixel image</td>
</tr>
<tr>
<td>width</td>
<td>$06</td>
<td>Word</td>
<td>Width</td>
</tr>
<tr>
<td>boundsRect</td>
<td>$08</td>
<td>Rect</td>
<td>Boundary rectangle</td>
</tr>
<tr>
<td>PaintParam (PaintPixels parameter block)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ptrToSourceLocInfo</td>
<td>$00</td>
<td>LocInfoPtr</td>
<td>Pointer to source location information</td>
</tr>
<tr>
<td>ptrToDestLocInfo</td>
<td>$04</td>
<td>LocInfoPtr</td>
<td>Pointer to destination location information</td>
</tr>
<tr>
<td>ptrToSourceRect</td>
<td>$08</td>
<td>RectPtr</td>
<td>Pointer to source rectangle</td>
</tr>
<tr>
<td>ptrToDestPoint</td>
<td>$0C</td>
<td>PointPtr</td>
<td>Pointer to destination point</td>
</tr>
<tr>
<td>mode</td>
<td>$10</td>
<td>Word</td>
<td>Mode</td>
</tr>
<tr>
<td>maskHandle</td>
<td>$12</td>
<td>Handle</td>
<td>Handle to clipping region</td>
</tr>
</tbody>
</table>

(continued)

QuickDraw II summary 16-277
Table 16-12 (continued)
QuickDraw II data structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>PenState record</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>psPnSize</td>
<td>$00</td>
<td>Point</td>
<td>Pen size</td>
</tr>
<tr>
<td>psPnMode</td>
<td>$04</td>
<td>Word</td>
<td>Pen mode</td>
</tr>
<tr>
<td>psPnPal</td>
<td>$06</td>
<td>Pattern</td>
<td>Pen pattern</td>
</tr>
<tr>
<td>psPnMask</td>
<td>$26</td>
<td>Mask</td>
<td>Pen mask</td>
</tr>
<tr>
<td>ROMFontRecord</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rfFamNum</td>
<td>$00</td>
<td>Word</td>
<td>Font family number of ROM font</td>
</tr>
<tr>
<td>rfFamStyle</td>
<td>$02</td>
<td>Word</td>
<td>Style of ROM font</td>
</tr>
<tr>
<td>rfSize</td>
<td>$04</td>
<td>Word</td>
<td>Point size of ROM font</td>
</tr>
<tr>
<td>rfFontHandle</td>
<td>$06</td>
<td>FontHndl</td>
<td>Handle to font</td>
</tr>
<tr>
<td>rfNamePtr</td>
<td>$0A</td>
<td>Pointer</td>
<td>Pointer to font name</td>
</tr>
<tr>
<td>rfFBRExtent</td>
<td>$0E</td>
<td>Word</td>
<td>fnrExtent for ROM font</td>
</tr>
</tbody>
</table>

Note: The actual assembly-language equates have a lowercase letter o in front of all of the names given in this table.

Table 16-13
QuickDraw II error codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0401</td>
<td>alreadyInitialized</td>
<td>Attempt made to start up QuickDraw II without first shutting it down</td>
</tr>
<tr>
<td>$0402</td>
<td>cannotReset</td>
<td>Never used</td>
</tr>
<tr>
<td>$0403</td>
<td>notInitialized</td>
<td>Quickdraw II not initialized</td>
</tr>
<tr>
<td>$0410</td>
<td>screenReserved</td>
<td>Memory Manager reported that screen memory (bank $E1 from $2000 to $9FFF) is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>already owned by someone else</td>
</tr>
<tr>
<td>$0411</td>
<td>badRect</td>
<td>Invalid rectangle specified</td>
</tr>
<tr>
<td>$0420</td>
<td>notEqualChunkiness</td>
<td>Source and destination pixels not the same type</td>
</tr>
<tr>
<td>$0430</td>
<td>rgnAlreadyOpen</td>
<td>Region already being saved in current GrafPort</td>
</tr>
<tr>
<td>$0431</td>
<td>rgnNotOpen</td>
<td>No region open in current GrafPort</td>
</tr>
<tr>
<td>$0432</td>
<td>rgnScanOverflow</td>
<td>Region scan overflow</td>
</tr>
<tr>
<td>$0433</td>
<td>rgnFull</td>
<td>Region full</td>
</tr>
<tr>
<td>$0440</td>
<td>polyAlreadyOpen</td>
<td>Polygon already open</td>
</tr>
<tr>
<td>$0441</td>
<td>polyNotOpen</td>
<td>Polygon not open</td>
</tr>
<tr>
<td>$0442</td>
<td>polyTooBig</td>
<td>Polygon too big</td>
</tr>
<tr>
<td>$0450</td>
<td>badTableNum</td>
<td>Invalid table number; 0 to 15 are valid</td>
</tr>
<tr>
<td>$0451</td>
<td>badColorNum</td>
<td>Invalid color number; 0 to 15 are valid</td>
</tr>
<tr>
<td>$0452</td>
<td>badScanLine</td>
<td>Invalid scan line number; 0 to 199 are valid</td>
</tr>
<tr>
<td>$04FF</td>
<td>notImplemented</td>
<td>Call not implemented</td>
</tr>
</tbody>
</table>
QuickDraw II Auxiliary adds features that did not appear in QuickDraw II because of implementation time or memory space. In particular, QuickDraw II adds the capability to deal with pictures and allows applications to outline, shadow, and italicize text.

The picture routines listed in this chapter have QuickDraw II tool set numbers, and thus could be considered a part of QuickDraw II. However, those routines are not available unless QuickDraw II Auxiliary has been loaded and started up, so they are included here.

A preview of the QuickDraw II Auxiliary routines

To introduce you to the capabilities of QuickDraw II, all QuickDraw II routines are grouped by function and briefly described in Table 16-1. These routines are described in detail later in this chapter, where they are separated into housekeeping routines (discussed in routine number order) and the rest of the QuickDraw II routines (discussed in alphabetical order).
Table 17-1
QuickDraw II Auxiliary routines and their functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Housekeeping routines</strong></td>
<td></td>
</tr>
<tr>
<td>QDAuxBootInit</td>
<td>Initializes QuickDraw II Auxiliary; called only by the Tool Locator—must not be called by an application</td>
</tr>
<tr>
<td>QDAuxStartup</td>
<td>Starts up QuickDraw II Auxiliary for use by an application</td>
</tr>
<tr>
<td>QDAuxShutDown</td>
<td>Shuts down QuickDraw II Auxiliary when an application quits</td>
</tr>
<tr>
<td>QDAuxVersion</td>
<td>Returns the version number of QuickDraw II Auxiliary</td>
</tr>
<tr>
<td>QDAuxReset</td>
<td>Resets QuickDraw II Auxiliary; called only when the system is reset—must not be called by an application</td>
</tr>
<tr>
<td>QDAuxStatus</td>
<td>Indicates whether QuickDraw II Auxiliary is active</td>
</tr>
<tr>
<td><strong>Auxiliary routines</strong></td>
<td></td>
</tr>
<tr>
<td>OpenPicture</td>
<td>Allocates memory for the recording of drawing commands into a picture definition and returns a handle to the picture</td>
</tr>
<tr>
<td>PicComment</td>
<td>Inserts a specified comment into the currently open picture</td>
</tr>
<tr>
<td>ClosePicture</td>
<td>Completes the picture definition process begun by an OpenPicture call</td>
</tr>
<tr>
<td>DrawPicture</td>
<td>Takes the drawing commands recorded in the picture definition, maps them from the picture frame into a specified destination rectangle, and draws them</td>
</tr>
<tr>
<td>KillPicture</td>
<td>Releases all memory occupied by a specified picture</td>
</tr>
<tr>
<td>CopyPixels</td>
<td>Copies a pixel image from one place to another, stretching or compressing it as necessary to make the source pixels fit the destination rectangle</td>
</tr>
<tr>
<td>DrawIcon</td>
<td>Draws a specified icon in a specified mode at a specified location and clips to the current visible and clipping regions</td>
</tr>
<tr>
<td>WaitCursor</td>
<td>Changes the cursor to a predefined cursor that looks like a watch</td>
</tr>
</tbody>
</table>

About pictures

A picture, as defined by QuickDraw II, is a record of drawing commands. The OpenPicture call establishes the picture frame that is used in conjunction with the destination rectangle to map objects from one space to another when the picture is drawn.

The mapping occurs as follows: If the picture frame is (0,0,100,100) and the destination rectangle is (50,50,60,60), a line recorded from (10,10) to (90,90) would appear from (51,51) to (59,59) when the picture is drawn.

Any text drawn is also scaled by using the picture frame and the destination rectangle. Using the previous example, a text size of 60 would appear as 6 when the picture is drawn. If the horizontal and vertical coordinates are not scaled by an equal amount, the vertical change is used to select the correct size for the font. The actual scaling of the font is handled by the Font Manager.
One of the most common uses of pictures is for printing. The Print Manager uses pictures to record what you want to put on a page, then plays back the picture over and over again into a band buffer. The band buffer is then printed. Because of this technique, your application only has to record the drawing commands once; the print driver can use the resulting picture and deal with the appropriate band buffer.

Another common use of pictures is to pass data back and forth from one application to another. The picture data type is one of two standard types defined by the Scrap Manager.

### Style modification support

At the time of publication, QuickDraw II provides ROM support for bold and underlined text, while QuickDraw II Auxiliary supports outlined, shadowed, and italicized text. Therefore, if your application is using outlined, shadowed, or italicized text, or if you are allowing the user to choose such style modifications, you must load and start up QuickDraw II Auxiliary.

### QuickDraw II Auxiliary icon record

An icon is a small graphic object that is usually symbolic of an operation or of a larger entity, such as a document. The **QuickDraw II Auxiliary icon record** indicates whether the icon is in color or black and white, the size of the icon, the height and width of the icon, the icon image, and the mask controlling the appearance of the icon, as shown in Figure 17-1.

*Note*: At the time of publication, this record had not been included in the APW interface file.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>iconType</td>
<td>Word—Bit 15 set to 1 = color icon</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set to 0 = black and white icon</td>
</tr>
<tr>
<td>$1</td>
<td>'iconSize'</td>
<td>Word—INTEGER: number of bytes in icon image</td>
</tr>
<tr>
<td>$2</td>
<td>'iconHeight'</td>
<td>Word—INTEGER: height of icon in pixels</td>
</tr>
<tr>
<td>$3</td>
<td>'iconWidth'</td>
<td>Word—INTEGER: width of icon in pixels</td>
</tr>
<tr>
<td>$4</td>
<td>'iconImage'</td>
<td>x Bytes—Icon image: <code>iconSize</code> bytes long, each row of pixels is $1 + (icon width -1)/2$ bytes wide</td>
</tr>
<tr>
<td>$5</td>
<td>'iconMask'</td>
<td>x Bytes—Icon mask: <code>iconSize</code> bytes long, each row of pixels is $1 + (icon width -1)/2$ bytes wide</td>
</tr>
</tbody>
</table>

*Figure 17-1*

QuickDraw II Auxiliary Icon record
There is also a `displayMode` word that controls how the `iconMask` is applied. When the `displayMode` word is 0, the icon is copied to the destination through the specified mask. The `displayMode` word can also have the values shown in Figure 17-2.

![Figure 17-2](image)

The `displayMode` word

All three of the operations in bits 2–0 can occur at once, and the testing is in the following order:

1. Check `openIconBit` (bit 1). Is it open (set to 1)?
2. Check `offLineBit` (bit 2). Is it off-line (set to 1)?
3. Check `selectedIconBit` (bit 0). Is it selected (set to 1)?

Color is only applied to the black and white icons if bits 15–8 are not all 0. Colored pixels in an icon are inverted by black pixels becoming white and any other color of pixel becoming black.

Your application draws the icon by using a `DrawIcon` call. See the section "DrawIcon" in this chapter.
Using QuickDraw II Auxiliary

This section discusses how the QuickDraw II Auxiliary routines fit into the general flow of an application and gives you an idea of which routines you'll need to use under normal circumstances. Each routine is described in detail later in this chapter.

QuickDraw II Auxiliary depends on the presence of the tool sets shown in Table 17-2 and requires that at least the indicated version of the tool set be present.

### Table 17-2

<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
<th>Minimum version needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01 #01</td>
<td>Tool Locator</td>
<td>1.0</td>
</tr>
<tr>
<td>$02 #02</td>
<td>Memory Manager</td>
<td>1.0</td>
</tr>
<tr>
<td>$03 #03</td>
<td>Miscellaneous Tool Set</td>
<td>1.0</td>
</tr>
<tr>
<td>$04 #04</td>
<td>QuickDraw II</td>
<td>1.2</td>
</tr>
</tbody>
</table>

In addition, if your application is using the DrawPicture routine, the Font Manager (tool set number $1B) must be loaded and started up.

The first QuickDraw II Auxiliary call your application must make is QDAuxStartUp. Conversely, when you quit your application, you must make the QDAuxShutDown call.

The OpenPicture begins the picture definition process. While OpenPicture is in effect, QuickDraw II drawing commands and any comments from PicComment calls are recorded and placed in the picture definition. When your application is through recording the picture, use the ClosePicture routine to stop the picture definition process and the DrawPicture routine to draw the picture in the destination rectangle. Finally, when you are completely through with a picture, you can use the KillPicture routine to release the memory the picture occupies.

Your application can draw an icon by using a Drawlcon call, produce a cursor that looks like a watch by using a WaitCursor call, or copy a pixel image from one place to another by using a CopyPixels call.
$0112  **QDAuxBootInit**
Initializes QuickDraw II Auxiliary; called only by the Tool Locator.

**Warning**
An application must never make this call.

**Parameters**
The stack is not affected by this call. There are no input or output parameters.

**Errors**
None

**C**
Call must not be made by an application.

$0212  **QDAuxStartUp**
Starts up QuickDraw II Auxiliary for use by an application.

**Important**
Your application must make this call before it makes any other QuickDraw II Auxiliary calls.

**Parameters**
The stack is not affected by this call. There are no input or output parameters.

**Errors**
None

**C**
extern pascal void QDAuxStartUp()
$0312 \textbf{QDAuxShutDown}

Shuts down QuickDraw II Auxiliary when an application quits.

\textbf{Important}

If your application has started up QuickDraw II Auxiliary, the application must make this call before it quits.

\begin{itemize}
\item \textbf{Parameters} The stack is not affected by this call. There are no input or output parameters.
\item \textbf{Errors} None
\item \textbf{C}
\begin{verbatim}
extern pascal void QDAuxShutDown()
\end{verbatim}
\end{itemize}

$0412 \textbf{QDAuxVersion}

Returns the version number of QuickDraw II Auxiliary.

\begin{itemize}
\item \textbf{Parameters}
\item \textbf{Stack before call}
\begin{verbatim}
previous contents
wordspace
\end{verbatim}
\begin{verbatim}
\end{verbatim}
\textbf{Word—Space for result}
\begin{verbatim}
← SP
\end{verbatim}
\item \textbf{Stack after call}
\begin{verbatim}
previous contents
versionInfo
\end{verbatim}
\begin{verbatim}
\end{verbatim}
\textbf{Word—Version number of QuickDraw II Auxiliary}
\begin{verbatim}
← SP
\end{verbatim}
\item \textbf{Errors} None
\item \textbf{C}
\begin{verbatim}
extern pascal Word QDAuxVersion()
\end{verbatim}
\end{itemize}

QuickDraw II Auxiliary housekeeping routines
$0512  QDAuxReset

Resets QuickDraw II Auxiliary; called only when the system is reset.

Warning
An application must never make this call.

Parameters
The stack is not affected by this call. There are no input or output parameters.

Errors
None

C
Call must not be made by an application.

$0612  QDAuxStatus

Indicates whether QuickDraw II Auxiliary is active.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
</tr>
</tbody>
</table>

Word—Space for result

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>activeFlag</td>
</tr>
</tbody>
</table>

Word—BOOLEAN; TRUE if QuickDraw II Auxiliary active, FALSE if inactive

Errors
None

C
extern pascal Boolean QDAuxStatus()

17-8  QuickDraw II Auxiliary housekeeping routines
ClosePicture

Completes the picture definition process begun by an OpenPicture call.

---

**Important**

Calls to OpenPicture and ClosePicture must be balanced; that is, one ClosePicture call must be made for every OpenPicture call.

ClosePicture calls the QuickDraw II routine ShowPen, thus balancing the HidePen call made by the OpenPicture call.

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

**Errors**

None

**C**

```c
extern pascal void ClosePicture()
```
CopyPixels

Copies a pixel image from one place to another, stretching or compressing it as necessary to make the source pixels fit the destination rectangle.

If the destination locinfo record is the same as the locinfo record of the current GrafPort, the pixels are also clipped to the GrafPort’s visible and clipping regions.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcLocPtr</td>
</tr>
<tr>
<td>destLocPtr</td>
</tr>
<tr>
<td>srcRect</td>
</tr>
<tr>
<td>destRect</td>
</tr>
<tr>
<td>xferMode</td>
</tr>
<tr>
<td>maskRgn</td>
</tr>
</tbody>
</table>

Long—POINTER to locinfo record of source rectangle
Long—POINTER to locinfo record of destination rectangle
Long—POINTER to RECT defining source rectangle
Long—POINTER to RECT defining destination rectangle
Word—Pen mode
Long—HANDLE to mask region

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

Errors

None

C

```c
extern pascal void CopyPixels(srcLocPtr, destLocPtr, srcRect, destRect,
                                xferMode, maskRgn)
LocInfoPtr srcLocPtr;
LocInfoPtr destLocPtr;
Rect *srcRect;
Rect *destRect;
Word xferMode;
RgnHandle maskRgn;
```

QuickDraw II Auxiliary routines
$0B12** DrawIcon**

Draws a specified icon in a specified mode at a specified location and clips to the current visible and clipping regions. The routine does not contribute to a picture definition, nor does it print the icon.

The QuickDraw II Auxiliary icon record and display mode word are described in the section "QuickDraw II Auxiliary Icon Record" in this chapter.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>iconPtr</td>
</tr>
<tr>
<td>displayMode</td>
</tr>
<tr>
<td>xPos</td>
</tr>
<tr>
<td>yPos</td>
</tr>
</tbody>
</table>

- **Long**—POINTER to icon record (see Figure 17-1)
- **Word**—Bit flag defining icon's appearance (see Figure 17-2)
- **Word**—X coordinate of upper left corner of icon
- **Word**—Y coordinate of upper left corner of icon

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

None

**C**

extern pascal void DrawIcon(iconPtr, displayMode, xPos, yPos)

Pointer iconPtr;
Word displayMode;
Word xPos;
Word yPos;
DrawPicture

Takes the drawing commands recorded in the picture definition, maps them from the picture frame into a specified rectangle, and draws them.

**Warning**

If you call DrawPicture with the initial, arbitrarily large clipping region, and the destination rectangle is either offset or larger than the picture frame, the clipping region might be set to empty and no drawing will be done.

DrawPicture passes any picture comments to a low-level procedure accessed through the grafProcs field of the GrafPort. For more information, see the section "PicComment" in this chapter.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- picHandle --</td>
</tr>
<tr>
<td>-- destRect --</td>
</tr>
</tbody>
</table>

- Long—HANDLE to picture
- Long—POINTER to RECT defining destination rectangle

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

None

**C**

```c
extern pascal void DrawPicture(picHandle, destRect)
Handle picHandle;
Rect *destRect;
```
$BB04 \textbf{KillPicture} \quad \text{Releases all memory occupied by a specified picture. Use this call only when your application is completely through with a picture.} \quad \textbf{Parameters} \quad \textbf{Stack before call} \quad \text{previous contents} \quad \text{Long---HANDLE to picture} \quad \leftarrow SP \\
\text{previous contents} \quad \leftarrow SP \\
\text{Errors} \quad \text{None} \quad \text{C} \quad \text{extern pascal void KillPicture(pichandle)} \quad \text{Handle pichandle;
OpenPicture

Allocates memory for the recording of drawing commands into a picture definition, and returns a handle to the picture.

---

**Warning**

A GrafPort’s clipping region is initialized to an arbitrarily large region. To ensure that the clipping region is still valid when drawing occurs and the region is mapped from the `picFrame` to the `destRect`, you should always change the clipping region to a smaller region before calling OpenPicture.

OpenPicture also calls the QuickDraw II routine HidePen, so no drawing will occur on the screen while the picture is open.

*Note:* No drawing will occur unless you call ShowPen just after OpenPicture, or you call ShowPen without previously balancing it by a call to HidePen.

When a picture is open, the GrafPort’s `picSave` field contains a handle to information related to the picture definition. If you want to temporarily disable the collection of routine calls and picture comments, save the current value of `picSave` and set the field to NIL; then restore the value when you want to resume the picture definition.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>longspace</code></td>
</tr>
<tr>
<td><code>picFrame</code></td>
</tr>
</tbody>
</table>

- `Long`—Space for result
- `Long`—POINTER to RECT defining picture frame

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>picHandle</code></td>
</tr>
</tbody>
</table>

- `Long`—HANDLE to picture

**Errors**

None

**C**

```
extern pascal Handle OpenPicture(picFrame)
```

17-14  QuickDraw II Auxiliary routines
$B804 \textbf{PicComment}

Inserts a specified comment into the currently open picture. An application that processes the comments must include a procedure to do the processing and store a pointer to that procedure in the \textit{grafProcs} field of the GrafPort.

\textbf{Note:} The standard low-level procedure for processing picture comments simply ignores all comments.

\section*{Parameters}

\textbf{Stack before call}

\begin{tabular}{|c|c|}
\hline
\textit{previous contents} &  \\
\hline
\textit{commentKind} & \textbf{Word}—Type of comment  \\
\hline
\textit{dataSize} & \textbf{Word}—Size of additional data; 0 if none  \\
\hline
\textit{dataHandle} & \textbf{Long}—HANDLE to additional data; NIL if none  \\
\hline
\end{tabular}

\begin{tabular}{|c|}
\hline
\textbf{← SP} &  \\
\hline
\end{tabular}

\textbf{Stack after call}

\begin{tabular}{|c|}
\hline
\textbf{← SP} &  \\
\hline
\end{tabular}

\section*{Errors}

\textbf{None}

\textbf{C}

\begin{verbatim}
extern pascal void PicComment(commentKind, dataSize, dataHandle)

Integer commentKind;
Integer dataSize;
Handle dataHandle;
\end{verbatim}
$0A12 WaitCursor

Changes the cursor to a predefined cursor that looks like a watch.

- Note: You can restore the standard arrow cursor by making the QuickDraw II call InitCursor.

A desk accessory or tool set can make this without checking whether QuickDraw II Auxiliary is active and without checking for errors. If QuickDraw II Auxiliary has not been loaded and started up, the dispatcher will return an error, but nothing else will happen.

Parameters

The stack is not affected by this call. There are no input or output parameters.

Errors

None

C

extern pascal void WaitCursor()

QuickDraw II Auxiliary summary

QuickDraw II Auxiliary does not contain any predefined constants, data structures, or error codes.
The **Standard Apple Numeric Environment (SANE)** is extended-precision IEEE arithmetic, with elementary functions. It scrupulously conforms to the IEEE standard 754 for binary floating-point arithmetic and to the proposed IEEE standard 854, which is a radix-independent and word-length-independent standard for floating-point arithmetic.

SANE provides sufficient numeric support for most applications. It includes
- IEEE types single (32-bit), double (64-bit), and extended (80-bit)
- A 64-bit type for large integer computations, such as those used in accounting
- Fundamental floating-point operations (+, -, *, /, rem)
- Comparisons
- Binary-decimal and float-integer conversions
- Scanning and formatting for ASCII numeric strings
- Logs, trigs, and exponentials
- Compound and annuity functions for financial computations
- A random-number generator
- Functions for management of the floating-point environment
- Other functions required or recommended by the IEEE standard

The SANE Tool Set fully supports the Standard Apple Numeric Environment, matching the functions of the Macintosh SANE packages, as well as those of the 6502 assembly language SANE software from which it is derived.
The SANE Tool Set comprises the usual tool set housekeeping routines and the routines SANEFP816, SANEElems816, and SANEDecStr816 that serve as entry points for the major pieces of SANE code. Each call to SANEFP816, SANEElems816, or SANEDecStr816 passes an opword parameter specifying the operation to be performed. For example, the opword $0206 passed to SANEFP816 indicates "divide by a value of type single."

This chapter describes only the basic functions of the SANE routines and the differences between the 65C816 version and the 6502 version. SANE functions are completely documented in the *Apple Numerics Manual*, which you will need if you use SANE routines in your application.

### A preview of the SANE Tool Set routines

To introduce you to the capabilities of the SANE Tool Set, all SANE Tool Set routines are grouped by function and briefly described in Table 18-1. These routines are described in detail later in this chapter, where they are separated into housekeeping routines (discussed in routine number order) and the rest of the SANE Tool Set routines (discussed in alphabetical order).

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Housekeeping routines</strong></td>
<td></td>
</tr>
<tr>
<td>SANEBootInit</td>
<td>Initializes the SANE Tool Set; called only by the Tool Locator—must not be called by an application</td>
</tr>
<tr>
<td>SANEStartup</td>
<td>Starts up the SANE Tool Set for use by an application</td>
</tr>
<tr>
<td>SANEShutDown</td>
<td>Shuts down the SANE Tool Set when an application quits</td>
</tr>
<tr>
<td>SANEVersion</td>
<td>Returns the version number of the SANE Tool Set</td>
</tr>
<tr>
<td>SANEReset</td>
<td>Resets the SANE Tool Set; called only when the system is reset—must not be called by an application</td>
</tr>
<tr>
<td>SANESTatus</td>
<td>Indicates whether the SANE tool set is active</td>
</tr>
<tr>
<td><strong>SANE routines</strong></td>
<td></td>
</tr>
<tr>
<td>SANEDecStr816</td>
<td>Contains numeric scanners and formatter</td>
</tr>
<tr>
<td>SANEElems816</td>
<td>Contains elementary functions, financial functions, and a random-number generator</td>
</tr>
<tr>
<td>SANEFP816</td>
<td>Contains basic arithmetic operations, comparisons, conversions, environmental control, and IEEE auxiliary operations</td>
</tr>
</tbody>
</table>

18-2 Chapter 18: SANE Tool Set
Using the SANE Tool Set

This section discusses how the SANE Tool Set routines fit into the general flow of an application and gives you an idea of which routines you'll need to use under normal circumstances. Each routine is described in detail later in this chapter.

The SANE Tool Set depends upon the presence of the tool sets shown in Table 18-2 and requires that at least the indicated version of the tool set be present.

Table 18-2
SANE Tool Set—other tool sets required

<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
<th>Minimum version needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01</td>
<td>#01 Tool Locator</td>
<td>1.0</td>
</tr>
<tr>
<td>$02</td>
<td>#02 Memory Manager</td>
<td>1.0</td>
</tr>
</tbody>
</table>

The first SANE Tool Set call that your application must make is SANEStartUp. Conversely, when you quit your application, you must call SANEShutDown.

You can program with the SANE tool set using any assembler that generates code for the Apple II GS. The equate file SANE.equs and the macro file M16.SANE help you use SANE with the Apple II GS Programmer's Workshop (APW) assembler.

\* Note: The APW C Compiler fully supports SANE and includes C interfaces to all SANE tool set routines.

The following code for a binary operation illustrates a typical invocation of a SANE function:

```
PUSHLONG <4-byte source operand address>
PUSHLONG <4-byte destination operand address>
PUSHWORD <OpWord>
(SANE macro)
```

Some SANE operations require different numbers of arguments, some pass 16-bit integer arguments by value, and some return results in the X, Y, and status registers.

The following example illustrates the use of the numeric scanner and formatter. The procedure accepts as an argument an ASCII string representing a number of degrees and returns the trigonometric sine of the argument as a numeric ASCII string. Both input and output are Pascal strings; that is, byte 0 gives the length, and byte 1 contains the first character in the string. The caller of the procedure pushes the address of the input string and performs a JSR to SINE. The procedure overwrites the input string with the result (whose length may be as large as 80) and clears the stack. SINE uses SANEFP816, SANEElems816, and SANEDecStr816, which are the three principal functions in the SANE Tool Set.
; Somewhere early in the program, initialize SANE
; Call the Memory Manager to reserve 256 bytes of bank $0 for use
; as SANE direct page. #SANEdirectpg is the address of this memory in this example
PUSHWORD #SANEdirectpg
_SANESTartUp
; Near the end of the program, shut down SANE.
_SANEShutDown
; Call the Memory Manager to release the memory reserved for the
; SANE direct page (often by releasing ALL reserved memory)

Procedure SINE (vars : DecStr)

SINE ENTRY

PLA
STA return ; Save return address
PLA
STA sAdr ; Address of s -> sAdr
PLA
STA sAdr+2
LDA #1
STA index ; 1 -> index
PUSHLONG sAdr
PUSHLONG #index
PUSHLONG #theDec
PUSHLONG #vp
FPSTR2DEC ; s -> theDec
PUSHLONG #theDec
PUSHLONG #x
FD2X ; theDec -> x
PUSHLONG #const
PUSHLONG #x
FMULX ; Convert to radians: x*const -> x
PUSHLONG #x
FSINX ; Sin(x) -> x
PUSHLONG #theForm
PUSHLONG #x
PUSHLONG #theDec
FX2DEC ; x -> theDec
PUSHLONG #theForm
PUSHLONG #theDec
PUSHLONG sAdr
FD2STR ; theDec -> s
LDA return
PHA
RTS

index ds 2
vp ds 2
const dc h'AE C8 E9 94 12 35 FA 8E F9 3F' ; Constant = π/180
x ds 10
theForm dc i'1,10' ; First style, then digits
theDec ds 33 ; Sign, exponent, length, ASCII = (2+2+1+28)
return ds 2
sAdr ds 4

Using the SANE Tool Set 18-5
Performance characteristics and limitations

Your application must preserve bytes 24 to 29 (decimal) of the SANE direct page between calls to SANE. Those 6 bytes hold the floating-point environment and halt vector. The remainder of the SANE direct page is scratch space used only during SANE execution and does not need to be preserved across calls to SANE. The space is thus available to the application.

---

**Warning**

Future implementations of SANE on the IIGS may not store the floating-point environment and halt vector on the direct page. If you access these variables directly, you may forfeit upward compatibility. Always access these variables only through SANE calls.

---

Except for the SANEVersion routine, SANE Tool Set routines remove all arguments from the stack and return no results on the stack. Temporary stack growth during calls does not exceed 40 bytes (50 for elementary function calls).

The SANE Tool Set conforms to the general tool set rules for management of the CPU registers, modes, and busy flag. However, SANE never returns tool set error codes; all floating-point errors are handled internally by setting exception flags (see the *Apple Numerics Manual*).

Typical timings, based on a few sample values, are as follows:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add, subtract</td>
<td>0.5 - 1.2</td>
</tr>
<tr>
<td>Multiply</td>
<td>1.0 - 5.0</td>
</tr>
<tr>
<td>Divide</td>
<td>1.9 - 5.2</td>
</tr>
<tr>
<td>Scanner</td>
<td>0.7 - 1.4</td>
</tr>
<tr>
<td>Formatter</td>
<td>0.8 - 1.0</td>
</tr>
<tr>
<td>Extended-to-decimal</td>
<td>2.4 - 5.8</td>
</tr>
<tr>
<td>Decimal-to-extended</td>
<td>0.8 - 3.5</td>
</tr>
<tr>
<td>Trigonometric, exponential, logarithmic</td>
<td>50 - 100</td>
</tr>
</tbody>
</table>
Differences between 65C816 and 6502 SANE

The 65C816 version of SANE differs from the 6502 version in several ways. First, all address parameters in the 65C816 are 4 bytes instead of 2 bytes. Another difference is that, if you are not using macro calls, you access the routines via the tool set dispatcher rather than via a JSR statement. Thus, invocations end with

```assembly
LDX #SANEnum + FuncNum*256
JSL $E10000
```

instead of

```assembly
JSR <xx6502>
```

If you are using the macro calls, you don't see this difference, because the macro automatically expands into the correct version of the call.

The low-order bytes of the X and Y registers return information as documented in Part II of the *Apple Numerics Manual*. The high-order byte of X duplicates the contents of the low-order byte of Y. The high-order byte of Y is undefined.

Figure 18-1 shows the relationship between the return information in 6502 registers and that in 65816 registers.

---

**Figure 18-1**
SANE return information
The Remainder call uses the N flag from the Processor Status register and both bits 7 and 15 of the X register to return the sign of the quotient. The low 7 bits of X contain the absolute value of the quotient.

The final difference between the 65816 and 6502 versions lies in the halt mechanism. When a halt occurs in 65816 SANE, the input parameters and SANE opcode are located in the SANE direct page, as shown in Figure 18-2. For one- and two-argument calls, C holds the address of DST. For two-argument calls, B holds the address of SRC. For binary-to-decimal conversion, A holds the address of the decimal record, B holds the address of the binary value, and C holds the address of the decform record.

Important

Halts occur only on calls to SANEFP816. SANEElms816 stimulates the halt mechanism only through the procexit call. SANEDecStr816 makes no calls to SANEFP816 and therefore never stimulates the halt mechanism.

The halt vector is the address of a halt-handling routine stored as a 4-byte address. The SetHaltVector routine expects the 4-byte halt vector to be passed by value on the stack. GetHaltVector returns a 3-byte halt vector in the X and Y registers. X contains the two low bytes (first and second) of the halt vector, and Y contains the second and third bytes; the second byte of the halt vector occurs in both the X and Y registers.
Figure 18-2 illustrates how X and Y return the halt vector.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2 return addresses</td>
</tr>
<tr>
<td>2</td>
<td>Caller's direct page</td>
</tr>
<tr>
<td>4</td>
<td>Caller's data bank</td>
</tr>
<tr>
<td>6</td>
<td>Opword</td>
</tr>
<tr>
<td>8</td>
<td>&quot;C&quot; address</td>
</tr>
<tr>
<td>10</td>
<td>&quot;B&quot; address</td>
</tr>
<tr>
<td>12</td>
<td>&quot;A&quot; address</td>
</tr>
<tr>
<td>14</td>
<td>Halt vector</td>
</tr>
<tr>
<td>16</td>
<td>Environment</td>
</tr>
<tr>
<td>18</td>
<td>Pending exceptions</td>
</tr>
<tr>
<td>20</td>
<td>Pending X-lo</td>
</tr>
<tr>
<td>22</td>
<td>Pending X-hi, Y-lo</td>
</tr>
</tbody>
</table>

Figure 18-2
SANE direct page on halt
When a halt occurs, the 65816 SANE executes a JSL to HaltVector. The halt handler can continue execution as if no halt had occurred by executing RTI.

**Important**

SANE is not reentrant! Your application can’t call SANE within the halt handler. During the halt, the direct page can’t be altered, and making a SANE call would alter the direct page.

After return to the SANEFP816 code, SANE uses the A register, not pending exceptions, to set the final floating-point exceptions. Execute LDA PendingExceptions at the end of the halt handler to ensure that the exceptions from the current call are handled correctly.

**Warning**

Pending exceptions is a sum of the five exception constants, represented as an integer from 0 to 31. Unpredictable results occur if the A register contains a value outside this range when your application exits the halt handler.
**$010A  SANEBootInit**

Initializes the SANE Tool Set; called only by the Tool Locator.

---

**Warning**

An application must never make this call.

---

**Parameters**
The stack is not affected by this call. There are no input or output parameters.

**Errors**

None

**C**

Call must not be made by an application.
$020A  **SANEStartUp**

Starts up the SANE Tool Set for use by an application. This routine clears the SANE environment word, installing the default settings of round-to-nearest, round-to-extended-precision, all exceptions clear, and all halts disabled (these terms are defined in the *Apple Numerics Manual*).

**Important**

If you are using assembly language, your application must make this call before making any other SANE calls. If you are using APW C, the call is made automatically.

SANEStartUp also sets the halt vector to 0. As with 6502 SANE, halts remain inoperative until the application both enables halts and also gives the halt vector a nonzero address.

The `dPageAddr` parameter is the address of one page (256 bytes) in bank $0 that your application makes available to the SANE Tool Set for its use.

**Parameters**

**Stack before call**

```
<table>
<thead>
<tr>
<th>previous contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dPageAddr</code></td>
<td></td>
</tr>
<tr>
<td>Word—Bank $0 starting address of one page of direct-page space</td>
<td>&lt;- SP</td>
</tr>
</tbody>
</table>
```

**Stack after call**

```
<table>
<thead>
<tr>
<th>previous contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;- SP</td>
</tr>
</tbody>
</table>
```

**Errors**

None

**C**

The call is made automatically in APW C.
$030A  **SANEShutDown**

Shuts down the SANE Tool Set.

**Important**

If your application has started up the SANE Tool Set, the application must make this call before it quits.

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

**Errors**

None

**C**

`extern pascal void SANEShutDown()`

---

$040A  **SANEVersion**

Returns the version number of the SANE Tool Set.

**Parameters**

**Stack before call**

```
| previous contents |
| wordspace        |
```

**Word**—Space for result

`← SP`

**Stack after call**

```
| previous contents |
| versionInfo       |
```

**Word**—Version number of SANE Tool Set

`← SP`

**Errors**

None

**C**

`extern pascal Word SANEVersion()`
$050A  SANEReset

Resets the SANE Tool Set; called only when the system is reset.

**Warning**
An application must never make this call.

**Parameters**
The stack is not affected by this call. There are no input or output parameters.

**Errors**
None

C
Call must not be made by an application.

$060A  SANEStatus

Indicates whether the SANE Tool Set is active.

**Parameters**

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>wordspace</th>
<th>Word—Space for result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>activeFlag</th>
<th>Word—BOOLEAN; TRUE if SANE Tool Set active, FALSE if inactive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**
None

C
 extern pascal Boolean SANEStatus()
$0A0A  SANEDecStr816
Contains numeric scanners and formatter.

Parameters
See the Apple Numerics Manual for details.

Errors
None

C
extern pascal Void SANEDecStr816()

$0B0A  SANEElems816
Contains elementary functions, financial functions, and a random-number generator.

Parameters
See the Apple Numerics Manual for details.

Errors
None

C
extern pascal Void SANEElems816()

$090A  SANEFP816
Contains basic arithmetic operations, comparisons, conversions, environmental control, and IEEE auxiliary operations.

Parameters
See the Apple Numerics Manual for details.

Errors
None

C
extern pascal Void SANEFP816()

SANE Tool Set summary
The constants and data structures for the SANE Tool Set are defined in the Apple Numerics Manual. The SANE Tool Set does not contain any tool set error codes.
The **Scheduler** delays activation of a desk accessory or of other tasks until the resources that the desk accessory or task needs become available. Much of the system code is not **reentrant**; that is, the code cannot be called while it is already executing. For example, if a desk accessory was activated while the system was within nonreentrant code, the system would most likely fail. To prevent such a situation from occurring, the Apple IIGS provides a busy flag that the Scheduler can check to see if a needed resource is busy or available.

If you are writing an application or a desk accessory, you won't need to use the Scheduler. The only time you need to use the Scheduler is when you are writing one of the following:

- Your own tool set
- Interrupt handlers that access ProDOS 16 or the tool set routines

For example, an application that performed background printing might need to access the Scheduler.

---

**A preview of the Scheduler routines**

To introduce you to the capabilities of the Scheduler, all Scheduler routines are grouped by function and briefly described in Table 19-1. These routines are described in detail later in this chapter, where they are separated into housekeeping routines (discussed in routine number order) and the rest of the Scheduler routines (discussed in alphabetical order).
Table 19-1
Scheduler routines and their functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Housekeeping routines</strong></td>
<td></td>
</tr>
<tr>
<td>SchBootInit</td>
<td>Initializes the Scheduler; called only by the Tool Locator—must not be called by an application</td>
</tr>
<tr>
<td>SchStartUp</td>
<td>Starts up the Scheduler for use by an application</td>
</tr>
<tr>
<td>SchShutDown</td>
<td>Shuts down the Scheduler when an application quits</td>
</tr>
<tr>
<td>SchVersion</td>
<td>Returns the version number of the Scheduler</td>
</tr>
<tr>
<td>SchReset</td>
<td>Resets the Scheduler; called only when the system is reset—must not be called by an application</td>
</tr>
<tr>
<td>SchStatus</td>
<td>Indicates whether the Scheduler is active</td>
</tr>
<tr>
<td><strong>Scheduler queue routines</strong></td>
<td></td>
</tr>
<tr>
<td>SchAddTask</td>
<td>Adds a task to the Scheduler queue</td>
</tr>
<tr>
<td>SchFlush</td>
<td>Flushes all tasks in the Scheduler queue—must not be called by an application</td>
</tr>
</tbody>
</table>

Using the Scheduler

This section discusses how the Scheduler routines fit into the general flow of an application and gives you an idea of which routines you’ll need to use under normal circumstances. Each routine is described in detail later in this chapter.

The Scheduler depends upon the presence of the tool sets shown in Table 19-2 and requires that at least the indicated version of the tool set be present.

Table 19-2
Scheduler—other tool sets required

<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
<th>Minimum version needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>#01 Tool Locator</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Your application should make a SchStartUp call before making any other Scheduler calls.

*Note: At the time of publication, the SchStartUp call was not an absolute requirement, because the Tool Locator automatically started up the Scheduler at boot time. However, you should make the call anyway to guarantee that your application remains compatible with all future versions of the system.*

Your application should also call SchShutDown when the application quits.
The Scheduler revolves around the **busy flag** located at $E1/00FF. If you wish to change the state of the busy flag, you should use the routines INCBUSYFLG and DECBUSYFLG. Those routines are not tool set routines, but are accessed directly from vectors in bank $E1 as follows:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCBUSYFLG</td>
<td>$E1/0064</td>
</tr>
<tr>
<td>DECBUSYFLG</td>
<td>$E1/0068</td>
</tr>
</tbody>
</table>

When a nonreentrant module is entered, your tool set or interrupt handler should perform a JSL to INCBUSYFLG in full native mode (e, m, and x flags all set to 0). When exiting from the module, the application should perform a JSL to DECBUSYFLG. DECBUSYFLG decrements the busy flag and executes any tasks placed in the Scheduler queue with the tool set routine SchAddTask.

Your tool set or interrupt handler should use SchAddTask after it checks the state of the busy flag. If the flag is set to other than 0, the necessary system resources are not currently available, and you can add a task to the Scheduler queue by using the SchAddTask routine.

Note that your application should never call the SchFlush routine.
$0107 \textbf{SchBootInit} \\
Initializes the Scheduler; called only by the Tool Locator.

\textbf{Warning} \\
An application must never make this call.

\textbf{Parameters} \\
The stack is not affected by this call. There are no input or output parameters.

\textbf{Errors} \\
None

\textbf{C} \\
Call must not be made by an application.

$0207 \textbf{SchStartUp} \\
Starts up the Scheduler for use by an application.

\textit{Note}: At the time of publication, the SchStartUp call was not an absolute requirement, because the Tool Locator automatically started up the Scheduler at boot time. However, you should make the call anyway, to guarantee that your application remains compatible with all future versions of the system.

\textbf{Parameters} \\
The stack is not affected by this call. There are no input or output parameters.

\textbf{Errors} \\
None

\textbf{C} \\
\texttt{extern pascal void SchStartUp()}
$0307  **SchShutDown**

Shuts down the Scheduler when an application quits.

---

**Important**

If your application has started up the Scheduler, the application must make this call before it quits.

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

**Errors**

None

**C**

```c
extern pascal void SchShutDown()
```

---

$0407  **SchVersion**

Returns the version number of the Scheduler.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
</tbody>
</table>

| Word—Space for result |
| ← SP |

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>versionInfo</td>
</tr>
</tbody>
</table>

| Word—Version number of Scheduler |
| ← SP |

**Errors**

None

**C**

```c
extern pascal Word SchVersion()
```
$0507  **SchReset**

Resets the Scheduler; called only when the system is reset.

---

**Warning**

An application must never make this call.

---

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

**Errors**

None

C  
Cali must not be made by an application.

---

$0607  **SchStatus**

Indicates whether the Scheduler is active.

---

**Parameters**

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>workspace</th>
<th>Word—Space for result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>activeFlag</th>
<th>Word—BOOLEAN; TRUE if Scheduler active, FALSE if inactive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

None

C  
extern pascal Boolean SchStatus()

---

19-6  
Scheduler housekeeping routines
$0907  **SchAddTask**

Adds a task to the Scheduler queue. The queue has space for four items, enough to support the Desk Manager as well as other small interrupt handlers.

The Scheduler uses a JSL to launch the procedure pointed to by taskPtr. If the task can't be added to the queue because the queue is already full, onQueueFlag will be FALSE.

*Note:* The Scheduler is not designed to support multitasking.

When the busy flag is decremented to 0, the tasks in the queue are executed in the posted order.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td></td>
</tr>
<tr>
<td>taskPtr</td>
<td></td>
</tr>
</tbody>
</table>

- **Word**—Space for result
- **Long**—POINTER to task to be added

- **SP**

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>onQueueFlag</td>
<td></td>
</tr>
</tbody>
</table>

- **Word**—BOOLEAN; TRUE if task added to Scheduler's queue,

- **SP**FALSE if queue was full

**Errors**

None

**C**

```
extern pascal Boolean SchAddTask(taskPtr)
VoidProcPtr taskPtr;
```
$0A07  **SchFlush**
Flushes all tasks in the Scheduler's queue.

---

**Important**
An application must never make this call.

---

**Parameters**
The stack is not affected by this call. There are no input or output parameters.

**Errors**
None

**C**
Call must not be made by an application.

---

**Scheduler summary**
The Scheduler does not contain any predefined constants, data structures, or tool set error codes.
Chapter 20

Scrap Manager

The Scrap Manager lets an application handle cutting and pasting. From the user's point of view, all data that is cut or copied resides in the Clipboard. The Cut command deletes data from a document and places it in the Clipboard. The Copy command copies data into the Clipboard without deleting it from the document. A subsequent Paste command will insert the contents of the Clipboard at a specified place, whether that place is in the same document or another or in the same application or another. An application that supports cutting and pasting may also provide a Clipboard window for displaying the current contents of the scrap; it may show the Clipboard window at all times or only when requested via the Show (or Hide) Clipboard command.

Note: The Scrap Manager is intended to transfer limited amounts of data; attempts to transfer very large amounts of data may fail due to lack of memory.

The nature of the data to be transferred varies according to the application. For example, in a word processor, the data is text; in a graphics application, it's a picture. The amount of information retained about the data being transferred also varies. Between two text applications, text can be cut and pasted without any loss of information; however, if the user of a graphics application cuts a picture consisting of text and then pastes it into a word processor document, the text in the picture may not be editable in the word processor, or it may be editable but not look exactly the same as in the graphics application. The Scrap Manager allows for a variety of data types and provides a mechanism whereby applications have some control over how much information is retained when data is transferred.

The desk scrap is usually stored in memory, but can be stored on the disk (in the file CLIPBOARD in the SYSTEM subdirectory of the boot volume) if there's not enough room for it in memory.

Macintosh programmers: The scrap does not have to be in memory when an application starts or stops.
A preview of the Scrap Manager routines

To introduce you to the capabilities of the Scrap Manager, all Scrap Manager routines are grouped by function and briefly described in Table 20-1. These routines are described in detail later in this chapter, where they are separated into housekeeping routines (discussed in routine number order) and the rest of the Scrap Manager routines (discussed in alphabetical order).

Table 20-1  
Scrap Manager routines and their functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Housekeeping routines</strong></td>
<td></td>
</tr>
<tr>
<td>ScrapBootInit</td>
<td>Initializes the Scrap Manager; called only by the Tool Locator—must not be called by an application</td>
</tr>
<tr>
<td>ScrapStartUp</td>
<td>Starts up the Scrap Manager for use by an application</td>
</tr>
<tr>
<td>ScrapShutDown</td>
<td>Shuts down the Scrap Manager when an application quits</td>
</tr>
<tr>
<td>ScrapVersion</td>
<td>Returns the version number of the Scrap Manager</td>
</tr>
<tr>
<td>ScrapReset</td>
<td>Resets the Scrap Manager; called only when the system is reset—must not be called by an application</td>
</tr>
<tr>
<td>ScrapStatus</td>
<td>Indicates whether the Scrap Manager is active</td>
</tr>
<tr>
<td><strong>Scrap routines</strong></td>
<td></td>
</tr>
<tr>
<td>UnloadScrap</td>
<td>Writes the desk scrap from memory to the scrap file and releases the memory it occupied</td>
</tr>
<tr>
<td>LoadScrap</td>
<td>Reads the desk scrap from the scrap file into memory</td>
</tr>
<tr>
<td>ZeroScrap</td>
<td>Clears the contents of the scrap and increments the scrap count</td>
</tr>
<tr>
<td>PutScrap</td>
<td>Appends specified data to the scrap that has the same scrap type as the data</td>
</tr>
<tr>
<td>GetScrap</td>
<td>Copies scrap information of the appropriate type to a specified handle, setting the handle to the correct size</td>
</tr>
<tr>
<td>GetScrapCount</td>
<td>Returns the current scrap count</td>
</tr>
<tr>
<td>GetScrapState</td>
<td>Returns a flag indicating the current state of the scrap</td>
</tr>
<tr>
<td>GetScrapHandle</td>
<td>Returns a copy of the handle for the scrap of a specified type</td>
</tr>
<tr>
<td>GetScrapSize</td>
<td>Returns the size of the specified scrap</td>
</tr>
<tr>
<td>GetScrapPath</td>
<td>Returns a pointer to the pathname used for the Clipboard file</td>
</tr>
<tr>
<td>SetScrapPath</td>
<td>Sets a pointer to the pathname used for the Clipboard file</td>
</tr>
</tbody>
</table>
Memory and the desk scrap

A large desk scrap can prevent an application from being loaded. If your application needs to know about whether there's enough room for the desk scrap in memory, you can set up your application so that a small initial segment of it is loaded. That segment can contain a Scrap Manager call to get the scrap size.

After a decision is made about whether to keep the scrap in memory or on disk, the remaining segments of the application can be loaded as needed. Of course, if there isn't enough room for the scrap at application load time, there probably won't be room for it later when a user tries to paste its contents into a document.

There are other disadvantages to keeping the desk scrap on disk: The disk may be locked, it may not have enough room for the scrap, or it may be removed during use of the application.

Important
If the application can't write the scrap to disk, it should put up an alert box informing the user, who may want to cancel the operation at that point.

Desk scrap data types

From the user's point of view there can be only one item on the Clipboard at a time, but the application may store more than one version of the information in the scrap, each representing the same Clipboard contents in a different form. For example, text cut from a word processor document may be stored in the desk scrap both as text and as a QuickDraw II picture.

Why would you want to do this? You might want your application to keep information in its own internal format, but you may also want it to be able to communicate via the Clipboard with other applications. When a user cuts or copies something to the Clipboard, the application can put it there in two different ways:

1. It can put it there internally so that a subsequent paste operation can be easily handled.

2. It can put it there publicly so that if the user tries to paste it into another application or desk accessory, the other application can handle it.
There are two public scrap types, as shown in Table 20-2.

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Scrap type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>textScr</td>
<td>Text</td>
</tr>
<tr>
<td>1</td>
<td>picScr</td>
<td>Picture</td>
</tr>
</tbody>
</table>

Applications must write at least one of these standard types of data to the desk scrap and must be able to read both types. Most applications will prefer one of these types over the other; for example, a word processor prefers text, whereas a graphics application prefers pictures. An application should write at least its preferred standard type of data to the desk scrap, and it may write both types (to pass the most information possible to the receiving application, which may prefer the other type).

An application reading the desk scrap looks for its preferred data type. If the application's preferred type isn't there—or if it's there but it was written by an application having a different preferred type—the receiving application may or may not be able to convert the data to the type it needs. If it cannot, some information may be lost in the transfer process. For example, a graphics application can easily convert text to a picture, but the reverse isn't true.

---

**Using the Scrap Manager**

This section discusses how the Scrap Manager routines fit into the general flow of an application and gives you an idea of which routines you'll need under normal circumstances. Each routine is described in detail later in this chapter.

The Scrap Manager depends upon the presence of the tool sets shown in Table 20-3 and requires that at least the indicated version of the tool set be present.

<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
<th>Minimum Version needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01</td>
<td>#01 Tool Locator</td>
<td>1.0</td>
</tr>
<tr>
<td>$02</td>
<td>#02 Memory Manager</td>
<td>1.0</td>
</tr>
</tbody>
</table>

The first Scrap Manager call that your application must make is ScrapStartUp. Conversely, when you quit your application, you must call ScrapShutDown.
If your application supports display of the Clipboard, you can call GetScrapCount to find out whether a desk accessory has changed the desk scrap. The **scrap count** indicates how many times the scrap has changed. Save the value of this field when one of your application's windows is deactivated and a system window is activated. Check the value each time through the main event loop to see whether it has changed; if it has, the contents of the desk scrap have changed. If the Clipboard window is visible, it needs to be updated whenever the count changes.

When the user gives a Cut or Copy command, your application needs to write the cut or copied data to the desk scrap. First call ZeroScrap to clear its previous contents; then call PutScrap to put the data into the scrap. If it makes it easier for your application to transfer data, you can call PutScrap more than once with the same scrap type.

When the user gives a Paste command, call GetScrap to access data of a particular type in the desk scrap and to get information about the data.

- **Note**: ZeroScrap, PutScrap, and GetScrap all keep track of whether the scrap is in memory or on the disk, so you don't have to worry about loading it first. After any of these calls, the scrap will be in memory again.

---

### Setting up a private scrap

Instead of using the desk scrap for storing data that's cut and pasted within an application, you may want to set up a **private scrap** for this purpose.

- **Note**: In most applications that use the standard text or picture data types, it's simpler for the application to use the desk scrap. However, if your application defines its own private type of data, or if very large amounts of data might be cut and pasted, using a private scrap may result in faster cutting and pasting within the application.

The format of a private scrap can be anything you want, because no other application will use it. For example, an application can simply maintain a pointer to cut or copied data. The application must, however, be able to convert data between the format of its private scrap and the format of the desk scrap.

- **Note**: The LineEdit scrap is a private scrap for applications that use LineEdit. LineEdit provides routines that access its own scrap and transfer data between the LineEdit scrap and the desk scrap.
If you use a private scrap, you must be sure that the right data is always pasted when the user gives a Paste command (the right data being whatever was most recently cut or copied in any application or desk accessory) and that the Clipboard, if visible, always shows the current data. You should copy the contents of the desk scrap to your private scrap at application startup and whenever a desk accessory is deactivated (call GetScrap to access the desk scrap). When the application is terminated, or when a desk accessory is activated, you should copy the contents of the private scrap to the desk scrap. Call ZeroScrap to clear its previous contents; call PutScrap to write data to the desk scrap.

If transferring data between the two scraps means converting it, and possibly losing information, you can copy the scrap only when you actually need to, at the time something is cut or pasted. The desk scrap needn’t be copied to the private scrap unless one of the following conditions is true:

- A Paste command is given before the first Cut or Copy command after the application starts up.
- A desk accessory that changed the scrap was deactivated.

Until then, you must keep the contents of the desk scrap intact. If the Clipboard window is visible, you must display the desk scrap, instead of the private scrap, in that window.

After one of the preceding conditions has occurred, you can ignore the desk scrap until a desk accessory is activated or the application is terminated; in either of these cases, you must copy the private scrap back to the desk scrap. Thus, whatever was last cut or copied within the application is pasted if a Paste command is given in a desk accessory or in the next application. If no Cut or Copy commands are given within the application, you never have to change the desk scrap.

If your application encounters problems in trying to copy one scrap to another, it should alert the user. If the desk scrap is too large to copy to the private scrap, the user may want to leave the application or simply proceed with an empty Clipboard. If the private scrap is too large to copy to the desk scrap, either because it’s disk based and too large to copy into memory or because it exceeds the maximum size allowed for the desk scrap, the user may want to stay in the application and cut or copy a smaller item.
<table>
<thead>
<tr>
<th>$0116</th>
<th><strong>ScrapBootInit</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Initializes the Scrap Manager; called only by the Tool Locator.</td>
<td></td>
</tr>
</tbody>
</table>

**Warning**
An application must never make this call.

**Parameters**
The stack is not affected by this call. There are no input or output parameters.

**Errors**
None

**C**
Call must not be made by an application.

<table>
<thead>
<tr>
<th>$0216</th>
<th><strong>ScrapStartUp</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Starts up the Scrap Manager for use by an application.</td>
<td></td>
</tr>
</tbody>
</table>

**Important**
Your application must make this call before it makes any other Scrap Manager calls.

**Parameters**
The stack is not affected by this call. There are no input or output parameters.

**Errors**
None

**C**
extern pascal void ScrapStartUp()
$0316  **ScrapShutDown**

Shuts down the Scrap Manager.

---

**Important**

If your application has started up the Scrap Manager, the application must make this call before it quits.

---

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

**Errors**

None

**C**

```c
extern pascal void ScrapShutDown()
```

---

$0416  **ScrapVersion**

Returns the version number of the Scrap Manager.

---

**Parameters**

---

**Stack before call**

- `previous contents`
- `workspace`

<table>
<thead>
<tr>
<th>Word—Space for result</th>
</tr>
</thead>
</table>

| ← SP |

---

**Stack after call**

- `previous contents`
- `versionInfo`

<table>
<thead>
<tr>
<th>Word—Version number of the Scrap Manager</th>
</tr>
</thead>
</table>

| ← SP |

---

**Errors**

None

**C**

```c
extern pascal Word ScrapVersion()
```

---

20-8  **Scrap Manager housekeeping routines**
$0516 \textbf{ScrapReset} \\
Resets the Scrap Manager; called only when the system is reset.

\begin{center}
\textbf{Warning}\\
An application must never make this call.
\end{center}

\begin{description}
\item[Parameters] The stack is not affected by this call. There are no input or output parameters.
\item[Errors] None
\item[C] Call must not be made by an application.
\end{description}

$0616 \textbf{ScrapStatus} \\
Indicates whether the Scrap Manager is active.

\begin{description}
\item[Parameters]
\begin{description}
\item[Stack before call]
\begin{tabular}{l}
\textit{previous contents} \\
\textit{wordspace}
\end{tabular}
\item[Word] Space for result
\item[Stack after call]
\begin{tabular}{l}
\textit{previous contents} \\
\textit{activeFlag}
\end{tabular}
\item[Word] BOOLEAN; TRUE if Scrap Manager active, FALSE if inactive
\end{description}
\item[Errors] None
\item[C] \texttt{extern pascal Boolean ScrapStatus()}
\end{description}
$0D16  GetScrap

Copies scrap information of the appropriate type to a specified handle, setting the handle to the correct size.

*Note:* To copy the desk scrap to the LineEdit scrap, use the LineEdit routine LEMFromScrap.

### Parameters

#### Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>destHandle</th>
<th>scrapType</th>
</tr>
</thead>
</table>

- *destHandle* — **Long**—HANDLE to scrap destination
- *scrapType* — **Word**—Scrap type; picScrap, textScrap, or application defined

#### Stack after call

| previous contents | ← SP |

#### Errors

- **$1610**  badScrapType — No scrap of this type found
- Memory Manager errors — Returned unchanged
- ProDOS errors — Returned unchanged

### C

```c
extern pascal void GetScrap(destHandle, scrapType)
Handle      destHandle;
Word        scrapType;
```
$1216  GetScrapCount

Returns the current scrap count. The count changes every time ZeroScrap is called. You can use this count for testing whether the contents of the desk scrap have changed; if ZeroScrap has been called, presumably PutScrap has also been called. This information may be useful if your application supports display of the Clipboard or has a private scrap.

Parameters

Stack before call

\[
\begin{array}{c|c}
\text{previous contents} & \text{Word—Space for result} \\
\text{wordspace} & \leftarrow \text{SP}
\end{array}
\]

Stack after call

\[
\begin{array}{c|c}
\text{previous contents} & \text{Word—INTEGER; current scrap count} \\
\text{scrapCount} & \leftarrow \text{SP}
\end{array}
\]

Errors

None

C

```c
extern pascal unsigned int GetScrapCount()
```
$0E16  GetScrapHandle

Returns a copy of the handle for the scrap of a specified type.

GetScrapHandle allows you to access the scrap without making a copy of it, which might be important when memory is in short supply.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>~ longspace</td>
</tr>
<tr>
<td>scrapType</td>
</tr>
</tbody>
</table>

- **Long**—Space for result
- **Word**—Scrap type; picScrap, textScrap, or application defined

- ← SP

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>~ scrapHandle</td>
</tr>
</tbody>
</table>

- **Long**—HANDLE to specified type of scrap

- ← SP

**Errors**

- $1610  badScrapType  No scrap of this type found
- Memory Manager errors  Returned unchanged
- ProDOS errors  Returned unchanged

**C**

```pascal
extern pascal Handle GetScrapHandle(scrapType);

Word   scrapType;
```

20-12  Scrap Manager routines
GetScrapPath

Returns a pointer to the pathname used for the Clipboard file.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>longspace</strong></td>
</tr>
</tbody>
</table>

Long—Space for result

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pathPtr</strong></td>
</tr>
</tbody>
</table>

Long—POINTER to pathname

← SP

Errors

None

C

eextern pascal Pointer GetScrapPath()
$0F16 GetScrapSize

Returns the size of a specified scrap.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
<tr>
<td>scrapType</td>
</tr>
</tbody>
</table>

Long—Space for result

Word—Scrap type; picScrap, textScrap, or application defined

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>scrapSize</td>
</tr>
</tbody>
</table>

Long—Size of scrap in bytes

Errors

C

extern pascal LongWord GetScrapSize(scrapType)

Word scrapType;

$1610 badScrapType No scrap of this type found

Memory Manager errors Returned unchanged

ProDOS errors Returned unchanged

20-14 Scrap Manager routines
GetScrapState

Returns a flag indicating the current state of the scrap. The *scrapState* flag is set to a nonzero value if the scrap is in memory; it is set to 0 if the scrap is currently on disk.

- Note: The *scrapState* flag is actually 0 if the scrap should be on disk. The scrap may not be on disk because a user can delete the Clipboard file.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Word—Space for result</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>← SP</td>
</tr>
</tbody>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Word—INTEGER; nonzero if scrap in memory, 0 if on disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>scrapState</td>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

None

**C**

    extern pascal Word GetScrapState()

LoadScrap

Reads the desk scrap from the scrap file into memory. If the desk scrap is already in memory, it does nothing. If the Clipboard file cannot be found, no error is returned; the computer responds as if you had loaded an empty Clipboard file.

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

**Errors**

- Memory Manager errors: Returned unchanged
- ProDOS errors: Returned unchanged

**C**

    extern pascal void LoadScrap()
**$0C16 PutScrap**

Appends specified data to the scrap that has the same type as the data. If the scrap is on disk, the scrap is loaded.

---

**Important**

Don’t forget to call ZeroScrap if you want to clear the scrap’s previous contents. If you don’t call ZeroScrap, the data is appended to the existing scrap.

📍 **Note:** To copy the LineEdit scrap to the desk scrap, use the LineEdit routine LEToScrap.

---

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>numBytes</td>
</tr>
<tr>
<td>scrapType</td>
</tr>
<tr>
<td>srcPtr</td>
</tr>
</tbody>
</table>

- **Long**—LONGINT; number of bytes to write
- **Word**—Scrap type; picScrap, textScrap, or application defined
- **Long**—POINTER to data to be placed in scrap

---

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
</tr>
</tbody>
</table>

---

**Errors**

- Memory Manager errors Returned unchanged
- ProDOS errors Returned unchanged

**C**

```c
extern pascal void PutScrap(numBytes, scrapType, srcPtr)
```

```c
unsigned Longint numBytes;
Word scrapType;
Pointer srcPtr;
```

20-16 **Scrap Manager routines**
**$1116 SetScrapPath**

Sets a pointer to the pathname used for the Clipboard file.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>pathPtr Long—POINTER to pathname of Clipboard</td>
</tr>
</tbody>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
</tr>
</tbody>
</table>

**Errors**

None

**C**

```c
extern pascal void SetScrapPath(pathPtr)
Pointer pathPtr;
```

---

**$0916 UnloadScrap**

Writes the desk scrap from memory to the scrap file and releases the memory the desk scrap occupied. If the desk scrap is already on disk, UnloadScrap does nothing.

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

**Errors**

Memory Manager errors Returned unchanged
ProDOS errors Returned unchanged

**C**

```c
extern pascal void UnloadScrap()
```
$0B16  ZeroScrap

Clears the contents of the scrap, whether the scrap is memory or on disk, and also changes the scrap count. When the user selects Cut or Copy, your application should call ZeroScrap before it calls PutScrap.

Parameters

The stack is not affected by this call. There are no input or output parameters.

Errors

Memory Manager errors  Returned unchanged
ProDOS errors  Returned unchanged

C

extern pascal void ZeroScrap();
Scrap Manager summary

This section briefly summarizes the constants and tool set error codes contained in the Scrap Manager. There are no predefined data structures for the Scrap Manager.

Important

These definitions are provided in the appropriate interface file.

Table 20-4
Scrap Manager constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public scrap type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>textScrap</td>
<td>$0000</td>
<td>Text scrap</td>
</tr>
<tr>
<td>picScrap</td>
<td>$0001</td>
<td>Picture scrap</td>
</tr>
</tbody>
</table>

Table 20-5
Scrap Manager error codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1610</td>
<td>badScrapType</td>
<td>No scrap of this type found</td>
</tr>
</tbody>
</table>
The Sound Tool Set gives you the ability to access the sound hardware without having to know specific hardware input/output addresses.

Sound Tool Set calls (other than the standard housekeeping routines) can be broken down into two groups. The first group is made through the normal tool call mechanism, with parameters being passed to and from the called routines on the stack. The second group is composed of low-level routines that, unlike most tool calls, use an eight-bit accumulator, pass their parameters in registers, and are accessed through a jump table.

There are two other tool sets dealing with sound—the Note Synthesizer and the Note Sequencer. These tool sets are not documented in the Apple JIGS Toolbox Reference. At the time of publication, documentation of these tool sets is planned for another, as yet unnamed, book.

**Note:** This chapter uses the terms *Note Synthesizer* and *Free-Form Synthesizer*. In the world of Apple sound, these terms refer to specific tool sets and not to electronic musical instruments.

---

**A preview of the Sound Tool Set routines**

To introduce you to the capabilities of the Sound Tool Set, all Sound Tool Set routines are grouped by function and briefly described in Table 21-1. These routines are described in detail later in this chapter, where they are separated into housekeeping routines (discussed in routine number order), the rest of the numbered routines (discussed in alphabetical order), and the low-level routines.

---

### Important

The low-level routines do not have tool set routine names and thus do not fit the alphabetical order organization; instead, they are organized in jump table offset order. Also note that the low-level routines do not have routine numbers.
### Table 21-1
Sound Tool Set routines and their functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Housekeeping routines</strong></td>
<td></td>
</tr>
<tr>
<td>SoundBootInit</td>
<td>Initializes the Sound Tool Set; called only by the Tool Locator—must not be called by an application</td>
</tr>
<tr>
<td>SoundStartUp</td>
<td>Starts up the Sound Tool Set for use by an application</td>
</tr>
<tr>
<td>SoundShutDown</td>
<td>Shuts down the Sound Tool Set</td>
</tr>
<tr>
<td>SoundVersion</td>
<td>Returns the version number of the Sound Tool Set</td>
</tr>
<tr>
<td>SoundReset</td>
<td>Resets the Sound Tool Set; called only when the system is reset—must not be made by an application</td>
</tr>
<tr>
<td>SoundToolStatus</td>
<td>Indicates whether the Sound Tool Set is active</td>
</tr>
<tr>
<td><strong>RAM and volume routines</strong></td>
<td></td>
</tr>
<tr>
<td>WriteRamBlock</td>
<td>Writes a specified number of bytes from system RAM into DOC RAM</td>
</tr>
<tr>
<td>ReadRamBlock</td>
<td>Reads a specified number of bytes from DOC RAM into system RAM</td>
</tr>
<tr>
<td>GetSoundVolume</td>
<td>Reads the volume setting for a generator</td>
</tr>
<tr>
<td>SetSoundVolume</td>
<td>Changes the volume setting for the volume registers in the DOC or changes the system volume</td>
</tr>
<tr>
<td>GetTableAddress</td>
<td>Returns the jump table address for the low-level routines</td>
</tr>
<tr>
<td><strong>Free-Form Synthesizer routines</strong></td>
<td></td>
</tr>
<tr>
<td>FFStartSound</td>
<td>Enables the DOC to start generating sound on a particular generator</td>
</tr>
<tr>
<td>FFStopSound</td>
<td>Halts any specified sound generators that are generating sound</td>
</tr>
<tr>
<td>FFSoundStatus</td>
<td>Returns the status of all 15 sound generators</td>
</tr>
<tr>
<td>FFGeneratorStatus</td>
<td>Reads the first two bytes of the generator control block corresponding to a specified generator</td>
</tr>
<tr>
<td>SetSoundMIRQV</td>
<td>Sets up the entry point into the sound-interrupt handler</td>
</tr>
<tr>
<td>SetUserSoundIRQV</td>
<td>Sets up the entry point for an application-defined synthesizer interrupt handler</td>
</tr>
<tr>
<td>FFSoundDoneStatus</td>
<td>Returns the current Free-Form Synthesizer playing status</td>
</tr>
<tr>
<td><strong>Low-level routines</strong></td>
<td></td>
</tr>
<tr>
<td>Read Register</td>
<td>Reads any register in the DOC</td>
</tr>
<tr>
<td>Write Register</td>
<td>Writes a one-byte parameter to any register in the DOC</td>
</tr>
<tr>
<td>Read RAM</td>
<td>Reads any specified DOC RAM location</td>
</tr>
<tr>
<td>Write RAM</td>
<td>Writes a one-byte value to any specified DOC RAM location</td>
</tr>
<tr>
<td>Read Next</td>
<td>Reads the next location pointed to by the Sound GLU address register</td>
</tr>
<tr>
<td>Write Next</td>
<td>Writes one byte of data to the next DOC register or RAM location, depending on the setting of the Sound GLU control register</td>
</tr>
<tr>
<td>Disable Increment</td>
<td>Disables the auto-increment mode set up by a Read Register, Write Register, Read RAM, or Write RAM low-level sound routine, thus allowing your application to read a DOC register or memory location continuously</td>
</tr>
</tbody>
</table>
The Apple IIGS sound hardware supports two sound subsystems. The first subsystem is an extension of the Apple IIe sound capabilities. Using this subsystem, applications toggle a soft switch, which in turn generates clicks in a speaker. In addition, the IIGS allows the application to control the volume of the speaker.

The second subsystem uses a digital oscillator chip (DOC) and 64K of dedicated RAM. The Sound Tool Set contains all of the firmware routines required to access the sound hardware. Figure 21-1 shows the major functional blocks of the sound hardware.

Figure 21-1
Sound hardware block diagram
The **sound GLU** (general logic unit) acts as the interface chip between system hardware and sound hardware. Figure 21-2 shows the sound GLU registers.

![Sound GLU control](image)

Data register

```
7 6 5 4 3 2 1 0
```

Address pointer low

```
7 6 5 4 3 2 1 0
```

Address pointer high

```
15 14 13 12 11 10 9 8
```

**Figure 21-2**

Sound GLU registers

The DOC RAM stores the waveforms used for sound generation. The DOC can create sounds of any pitch and duration. Table 21-2 shows the DOC registers.

**Table 21-2**

DOC register allocation

<table>
<thead>
<tr>
<th>Register number</th>
<th>Function</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$00-1F</td>
<td>Frequency low</td>
<td>FL7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FL6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FL5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FL4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FL3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FL2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FL1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FL0</td>
</tr>
<tr>
<td>$20-3F</td>
<td>Frequency high</td>
<td>FH7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FH6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FH5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FH4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FH3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FH2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FH1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FH0</td>
</tr>
<tr>
<td>$40-5F</td>
<td>Volume</td>
<td>V7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V0</td>
</tr>
<tr>
<td>$60-7F</td>
<td>Data sample</td>
<td>W7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W0</td>
</tr>
<tr>
<td>$80-9F</td>
<td>Waveform table pointer</td>
<td>P7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P0</td>
</tr>
<tr>
<td>$A0-BF</td>
<td>Control</td>
<td>CA3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CA2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CA1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CA0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>H</td>
</tr>
<tr>
<td>$C0-DF</td>
<td>Bank select/table size/resolution</td>
<td>IRQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R0</td>
</tr>
<tr>
<td>$E0</td>
<td>Oscillator interrupt</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$E1</td>
<td>Oscillator enable</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>$E2</td>
<td>Analog/digital converter</td>
<td>S7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S0</td>
</tr>
</tbody>
</table>
For further information on the DOC, see the *Apple II GS Hardware Reference*.

The analog section contains all the circuitry needed to amplify and filter the signal coming from the sound GLU or the DOC. The signal will be sent to the speaker.

The sound connector provides the connection to interface cards that can take the tones generated by the DOC and modify them further. Two examples of possible sound cards are programmable filter stereo interface cards and sound sampling cards.

### Oscillators and generators

An oscillator is the basic sound-generating unit in the DOC. The DOC contains 32 oscillators, each of which can function independently from all the other oscillators.

One of the modes of the DOC is called swap mode. The Free-Form Synthesizer uses this mode to generate sounds. In swap mode, a pair (or **swap pair**) of oscillators form a functional oscillator unit called a **generator**. There are 15 generators defined in the Apple II GS sound system. An oscillator-to-generator translation table converts an oscillator number to the appropriate generator number. That table is accessed through the jump table shown in the section "GetTableAddress" in this chapter.

Each oscillator controls seven DOC registers in the range 00-DF, as shown in Table 21-3.

**Table 21-3**

<table>
<thead>
<tr>
<th>Register function</th>
<th>Oscillator 0 registers</th>
<th>Oscillator 1 registers</th>
<th>Oscillator n registers</th>
<th>...</th>
<th>Oscillator 31 registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency low</td>
<td>$00</td>
<td>$01</td>
<td>$00 + n</td>
<td>...</td>
<td>$1F</td>
</tr>
<tr>
<td>Frequency high</td>
<td>$20</td>
<td>$21</td>
<td>$20 + n</td>
<td>...</td>
<td>$3F</td>
</tr>
<tr>
<td>Volume control</td>
<td>$40</td>
<td>$41</td>
<td>$40 + n</td>
<td>...</td>
<td>$5F</td>
</tr>
<tr>
<td>Data sampling</td>
<td>$60</td>
<td>$61</td>
<td>$60 + n</td>
<td>...</td>
<td>$7F</td>
</tr>
<tr>
<td>Waveform table pointer</td>
<td>$80</td>
<td>$81</td>
<td>$80 + n</td>
<td>...</td>
<td>$9F</td>
</tr>
<tr>
<td>Control register</td>
<td>$A0</td>
<td>$A1</td>
<td>$A0 + n</td>
<td>...</td>
<td>$BF</td>
</tr>
<tr>
<td>Bank select/table size/resolution</td>
<td>$C0</td>
<td>$C1</td>
<td>$C0 + n</td>
<td>...</td>
<td>$DF</td>
</tr>
</tbody>
</table>

Oscillators 30 and 31 are reserved for system use and should not be used by applications. If an interrupt is generated by oscillator 30 or 31, control is passed to the System Failure Manager with an "unclaimed sound interrupt" message. When the SoundBootinit routine is called by the system at boot time, all of the sound interrupt handler pointers point to the System Failure Manager with this message.
The work area for the sound routines (specified in the SoundStartUp call) is broken down into 16 groups of 16 bytes each, with each 16-byte group comprising one generator control block (GCB). The first byte of each contains the synthesizer type being used by that generator. The high nibble is reserved for use by the system and must be zero. The low nibble of the byte (bits 3-0) contains the type. The remaining 15 bytes in the GCB are for use by the application, and their meaning and value may vary depending on the synthesizer type.

Using the Sound Tool Set

This section discusses how the Sound Tool Set routines fit into the general flow of an application and gives you an idea of which routines you'll need to use under normal circumstances. Each routine is described in detail later in this chapter.

The Sound Tool Set depends on the presence of the tool sets shown in Table 21-4, and requires that at least the indicated version of the tool set be present.

<table>
<thead>
<tr>
<th>Tool Set number</th>
<th>Tool Set name</th>
<th>Minimum version needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01</td>
<td>Tool Locator</td>
<td>1.0</td>
</tr>
<tr>
<td>$02</td>
<td>Memory Manager</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Your application must make the SoundStartUp call before it makes any other Sound Tool Set calls. Conversely, when your application quits, it must make the SoundShutDown call.

The Sound Tool Set gives you the ability to control the sound hardware without having to access the hardware registers directly. To provide this capability, the Sound Tool Set must be able to read and write to RAM, read and write to the DOC registers, and raise and lower the volume. This requires a set of low-level sound routines. Unlike the other Sound Tool Set routines, which use the stack to pass parameters in the normal tool call fashion, these routines use registers to pass parameters.

- Note: Because the low-level sound routines have been designed and implemented to increase performance, they may not be available in all programming languages. At the time of publication, for example, the low-level sound routines were not available from Apple IIGS Workshop C.

The low-level routines are entered through a jump table. The table address can be obtained through a call to the GetTableAddress routine. The actual format of the jump table is shown in the section "GetTableAddress" in this chapter.
$0108 SoundBootInit

Initializes the Sound Tool Set; called only by the Tool Locator.

---

**Warning**
An application must never make this call.

This routine performs the following:
- Resets all of the DOC RAM
- Resets the Sound Tool Set's work area
- Resets the oscillators to an uninitialized state

**Parameters**
The stack is not affected by this call. There are no input or output parameters.

**Errors**
None

**C**
Call must not be made by an application.
$0208 \textbf{SoundStartUp}

Starts up the Sound Tool Set for use by an application. The direct page must be page-aligned and locked until the SoundShutDown call is made.

\textbf{Important}
Your application must make this call before it makes any other Sound Tool Set calls.

\textbf{Parameters}

\textbf{Stack before call}

\begin{align*}
\text{previous contents} & \quad \text{Word—Bank} \ 0 \text{ starting address of one page of direct-page space} \\
\text{dPageAddr} & \quad \leftarrow \text{SP}
\end{align*}

\textbf{Stack after call}

\begin{align*}
\text{previous contents} & \quad \leftarrow \text{SP}
\end{align*}

\textbf{Errors}

$0810 \quad \text{noDOCFndErr} \quad \text{No DOC or RAM found}$

$0818 \quad \text{sndAlreadyStrtErr} \quad \text{Sound tools already started}$

\textbf{C}

\begin{verbatim}
extern pascal void SoundStartUp(dPageAddr)
Word dPageAddr;
\end{verbatim}
**SoundShutDown**

Shuts down the Sound Tool Set.

**Important**

If your application has started up the Sound Tool Set, the application must make this call before it quits.

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

**Errors**

None

**C**

```c
extern pascal void SoundShutDown()
```

---

**SoundVersion**

Returns the version number of the Sound Tool Set.

**Parameters**

**Stack before call**

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

**Word**—Space for result

```
<- SP
```

**Stack after call**

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>versionInfo</td>
</tr>
</tbody>
</table>
```

**Word**—Version number of Sound Tool Set

```
<- SP
```

**Errors**

None

**C**

```c
extern pascal Word SoundVersion()
```
$0508  SoundReset

 Resets the Sound Tool Set; called only when the system is reset.

---

Warning

An application must never make this call. This call is used only by the system to control the shutdown of generators. If you want to shut down a generator, use the StopSound routine.

---

Parameters

The stack is not affected by this call. There are no input or output parameters.

Errors

None

C

Call must not be made by an application.

---

$0608  SoundToolStatus

Indicates whether the Sound Tool Set is active.

Parameters

Stack before call

```
previous contents

wordspace

Word—Space for result

← SP
```

Stack after call

```
previous contents

activeFlag

Word—BOOLEAN; TRUE if Sound Tool Set active, FALSE if inactive

← SP
```

Errors

None

C

extern pascal Boolean SoundToolStatus()

21-10  Sound Tool Set housekeeping routines
$1108 \textbf{FFGeneratorStatus}

Reads the first two bytes of the generator control block that corresponds to a specified generator.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{wordspace}</td>
</tr>
<tr>
<td>\textit{genNumber}</td>
</tr>
</tbody>
</table>

- \textbf{Word}—Space for result
- \textbf{Word}—Number of generator whose status will be returned

\[\leftarrow \text{SP}\]

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{genStatus}</td>
</tr>
</tbody>
</table>

- \textbf{Word}—Status of \textit{genNumber} (see Figure 21-3)

\[\leftarrow \text{SP}\]

**Errors**

None

**C**

\begin{verbatim}
extern pascal Word FFGeneratorStatus(genNumber)
Word genNumber;
\end{verbatim}

(continued)
Generator status word

The status returned in the `genStatus` parameter is in the format shown in Figure 21-3.

![Figure 21-3 Generator status word](image)

- **Last Block**
  - Last block of wave = 1
  - Not last block of wave = 0
- **Reserved for future use; must be 0**
- **Generator available** = 00
- **Free-Form Synthesizer** = 01
- **Note Synthesizer** = 02
- **Reserved** = 03-07
- **Application-defined** = 08-0F
- **DOC channel number** (00-0F)
- **Generator number** (00-0E)
Returns the current Free-Form Synthesizer playing status. If the specified generator is currently playing out a waveform, the status returned to the caller will be TRUE. If the generator is not playing, the status will be FALSE ($FFFF).

Parameters

Stack before call

- **previou contents**
  - **workspace**
  - **genNumber**

  - **Word**—Space for result
  - **Word**—Number of generator whose status will be returned

  ← SP

Stack after call

- **previou contents**
  - **genDoneFlag**

  - **Word**—BOOLEAN; status of genNumber: TRUE if done playing, FALSE if playing

Errors

- $0813$ **invalidGenNumErr** Invalid generator number

C

```c
extern pascal Boolean FFSoundDoneStatus(genNumber);
```

Word genNumber;
$1008 \text{ FFSoundStatus}

Returns the status of all 15 sound generators. Any bit position set to 1 in the status word returned from the function call signifies that the corresponding generator is active. The format of the status word returned is the same as that of the stop-sound mask as illustrated in Figure 21-6 in the section "FFStopSound," with bit 0 corresponding to generator 0, bit 1 corresponding to generator 1, and so on.

Parameters

Stack before call

| previous contents | workspace |
| Word—Space for result |

Stack after call

| previous contents | genStatusInfo |
| Word—Status of all generators (bit 0 = generator 0 and so on) |

Errors

None

C

extern pascal Word FFSoundStatus()
FFStartSound

Enables the DOC to start generating sound on a particular generator. If this call is made to a generator that is already active, the previous sound-generation process is terminated and the new sound process is started.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>genNumFFSynth</th>
<th>Word—Channel, generator, and type (see Figure 21-4)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pBlockPtr</td>
<td>Long—POINTER to parameter block for sound (see Figure 21-5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

| previous contents | ← SP                    |

Errors

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0812</td>
<td>noSoundInitErr</td>
</tr>
<tr>
<td>$0813</td>
<td>invalGenNumErr</td>
</tr>
<tr>
<td>$0814</td>
<td>synthModeErr</td>
</tr>
<tr>
<td>$0815</td>
<td>genBusyErr</td>
</tr>
</tbody>
</table>

C

```c
extern pascal void FFStartSound(genNumFFSynth, pBlockPtr)
```

```c
Word genNumFFSynth;
Pointer pBlockPtr;
```

(continued)
Channel-generator-type word

The values for the channel-generator-type word are shown in Figure 21-4.

Figure 21-4
Channel-generator-type word

21-16    Sound Tool Set routines
Parameter block

The values for the parameter block are shown in Figure 21-5. The effective output sample rate can be calculated as follows:

\[ \text{freqOffset} = \frac{(32 \times \text{Output sample rate in hertz})}{1645} \]

For more detailed information on these settings, refer to the *Apple II GS Hardware Reference*.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>waveStart</td>
<td>Long—Starting address of wave</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>waveSize</td>
<td>Word—Waveform size in pages</td>
</tr>
<tr>
<td>3</td>
<td>freqOffset</td>
<td>Word—Output sample rate</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>docBuffer</td>
<td>Word—DOC buffer start address; high-order byte significant, low-order byte = 0</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>docBufferSize</td>
<td>Word—DOC buffer size; high-order byte = 0, low-order byte significant</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>nextWavePtr</td>
<td>Long—POINTER to start of next wave’s parameter block</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>voSetting</td>
<td>Word—DOC volume setting; high-order byte = 0, low-order byte significant</td>
</tr>
</tbody>
</table>

Figure 21-5
Sound parameter block
**FFStopSound**

Halts any specified sound generators that are generating sound. Depending on the setting of a 16-bit mask passed as a parameter to the routine, any of 15 generators will be stopped if running. Each bit position in the stop-sound mask corresponds to a sound generator. Bit 0 corresponds to generator 0, bit 1 corresponds to generator 1, and so on, up to bit 15 (bit 15 must be 0). Figure 21-6 illustrates the stop-sound mask.

**Parameters**

**Stack before call**

| previous contents | genMask | Word—Mask to stop generators (see Figure 21-6) | ← SP |

**Stack after call**

| previous contents | ← SP |

**Errors**

None

**C**

```c
extern pascal void FFStopSound(genMask)

Word genMask;
```

**Stop-sound mask**

The values for the stop-sound mask are shown in Figure 21-6.
Reserved: must be 0

Generator 14 off = 1
Generator 14 on = 0

Generator 13 off = 1
Generator 13 on = 0

Generator 12 off = 1
Generator 12 on = 0

Generator 11 off = 1
Generator 11 on = 0

Generator 10 off = 1
Generator 10 on = 0

Generator 9 off = 1
Generator 9 on = 0

Generator 8 off = 1
Generator 8 on = 0

Generator 7 off = 1
Generator 7 on = 0

Generator 6 off = 1
Generator 6 on = 0

Generator 5 off = 1
Generator 5 on = 0

Generator 4 off = 1
Generator 4 on = 0

Generator 3 off = 1
Generator 3 on = 0

Generator 2 off = 1
Generator 2 on = 0

Generator 1 off = 1
Generator 1 on = 0

Generator 0 off = 1
Generator 0 on = 0

Figure 21-6
Stop-sound mask
Assembly-language example

The following example stops generators 0 and 8:

PEA $1001

_StopSound
GetSoundVolume

Reads the volume setting for a generator. The range of possible values is from $00$ to $\text{FF}$. All eight bits are valid for DOC volume registers.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
<tr>
<td>genNumber</td>
</tr>
</tbody>
</table>

Word—Space for result
Word—Number of generator whose volume will be returned
<- SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>volSetting</td>
</tr>
</tbody>
</table>

Word—Volume setting, from $00$ to $\text{FF}$; high nibble of low-order byte
<- SP

Errors

None

C

extern pascal Word GetSoundVolume(genNumber)

Word genNumber;
$0B08 \textbf{GetTableAddress}

Returns the jump table address for the low-level routines (see Table 21-5).

Besides the offsets to the low-level routines, the jump table contains the following three additional functions:

- The oscillator table translates from generator number to oscillator number and returns the even number of the pair of oscillators.
- The GCB address table points to the first location of the GCB corresponding to a generator.
- The generator table translates from oscillator number to generator number.

\textbf{Parameters}

\textbf{Stack before call}

\begin{verbatim}
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
</tbody>
</table>
\end{verbatim}

Long—Space for result

← SP

\textbf{Stack after call}

\begin{verbatim}
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>jumpTableAddr</td>
</tr>
</tbody>
</table>
\end{verbatim}

Long—Jump table address for low-level routines

← SP

\textbf{Errors} None

\textbf{C}

\begin{verbatim}
extern pascal Pointer GetTableAddress()
\end{verbatim}

21-22 \textbf{Sound Tool Set routines}
Jump table addresses

Table 21-5 illustrates the format of the jump table addresses used by the Sound Tool Set low-level routines.

Table 21-5
Jump table addresses for Sound Tool Set low-level routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Offset</th>
<th>Constant</th>
<th>Address format</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>8 bits</td>
</tr>
<tr>
<td>Read Register</td>
<td>$00</td>
<td>readRegister</td>
<td>Address low</td>
</tr>
<tr>
<td>Write Register</td>
<td>$04</td>
<td>writeRegister</td>
<td>Address low</td>
</tr>
<tr>
<td>Read RAM</td>
<td>$08</td>
<td>readRam</td>
<td>Address low</td>
</tr>
<tr>
<td>Write RAM</td>
<td>$0C</td>
<td>writeRam</td>
<td>Address low</td>
</tr>
<tr>
<td>Read Next</td>
<td>$10</td>
<td>readNext</td>
<td>Address low</td>
</tr>
<tr>
<td>Write Next</td>
<td>$14</td>
<td>writeNext</td>
<td>Address low</td>
</tr>
<tr>
<td>Oscillator table</td>
<td>$18</td>
<td>osctable</td>
<td>Address low</td>
</tr>
<tr>
<td>Generator table</td>
<td>$1C</td>
<td>genTable</td>
<td>Address low</td>
</tr>
<tr>
<td>GCB address table</td>
<td>$20</td>
<td>gcbAddrTable</td>
<td>Address low</td>
</tr>
<tr>
<td>Disable increment</td>
<td>$24</td>
<td>disableInc</td>
<td>Address low</td>
</tr>
</tbody>
</table>
$0A08  **ReadRamBlock**

Reads a specified number of bytes from DOC RAM area into system RAM.

---

**Warning**

Interrupts must be disabled whenever your application accesses the DOC RAM. Your application must disable interrupts before it accesses the RAM and then reenable them afterward.

If the number of bytes and the starting location add up to a value greater than 64K, an error is generated.

---

**Important**

Your application must call the Memory Manager to allocate the buffer for data read from the DOC RAM.

---

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>destPtr</td>
</tr>
<tr>
<td>docStart</td>
</tr>
<tr>
<td>byteCount</td>
</tr>
</tbody>
</table>

**Long**—POINTER to starting RAM address where data will be written

**Word**—Starting DOC address from which data will be read

**Word**—Number of bytes to be copied

← SP

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

← SP

**Errors**

$0810  noDOCFindErr  No DOC or RAM found

$0811  docAddrRngErr  DOC address range error

**C**

```c
extern pascal void ReadRamBlock(destPtr,docStart,byteCount);
```

Pointer destPtr;

Word docStart;

Word byteCount;

---

21-24  **Sound Tool Set routines**
SetSoundMIRQV

Sets up the entry point into the sound interrupt handler. This routine is accessed every time an interrupt is generated by the DOC.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- sMasterIRQ --</td>
</tr>
</tbody>
</table>

Long—New master sound IRQ vector

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

← SP

Errors

None

C

extern pascal void SetSoundMIRQV(sMasterIRQ)

LongWord sMasterIRQ;
SetSoundVolume

Changes the volume setting for the volume registers in the DOC or changes the system volume.

If \textit{genNumber} is specified as $00$–$0E$, the call sets the volume on the corresponding pair of generators in the DOC. If \textit{genNumber} is specified as $0F$ or greater, the call sets the system volume control. The range of values for the volume setting are $00$–$FF$. The generator volume registers use all eight bits of resolution. The system volume control uses only the upper nibble to determine the setting.

\textbf{Parameters}

\textbf{Stack before call}

\begin{itemize}
  \item \textit{previous contents}
  \item \textbf{volume}—Generator or system volume setting; $00$–$FF$
  \item \textbf{genNumber}—Generator whose volume will be set; $00$–$0E$ sets DOC, $0F$–$FF$ sets system volume
\end{itemize}

\textbf{Stack after call}

\begin{itemize}
  \item \textit{previous contents}
  \item $\leftarrow SP$
\end{itemize}

\textbf{Errors}

None

\textbf{C}

\begin{verbatim}
extern pascal void SetSoundVolume(volume, genNumber)
    Word volume;
    Word genNumber;
\end{verbatim}

21-26 Sound Tool Set routines
Sets up the entry point for an application-defined synthesizer interrupt handler. When an interrupt occurs for an application-defined synthesizer, control is passed to the RAM-based synthesizer code through this vector. The old vector installed is passed back to the caller who must preserve the vector.

If control is passed to the user vector, the application-defined interrupt handler must validate that the synthesizer mode matches the synthesizer mode used by this handler. If it does not match, then the handler must pass control farther down the chain through the vector that was preserved. Control is passed through a JSR instruction; therefore, the application must return control through an RTI instruction.

### Parameters

#### Stack before call

- **previous contents**
  - `longspace`
  - `userIRQVector`

- `Long`—Space for result
- `Long`—New user sound IRQ vector

- `← SP`

#### Stack after call

- **previous contents**
  - `oldIRQVector`

- `Long`—Old user sound IRQ vector

- `← SP`

### Errors

- None

### C

```c
extern pascal Pointer SetUserSoundIRQV(userIRQVector)

LongWord userIRQVector;
```
WriteRamBlock

Writes a specified number of bytes from system RAM into DOC RAM.

Warning
Interrupts must be disabled whenever your application accesses the DOC RAM. Your application must disable interrupts before it accesses the RAM and then reenable them afterward.

If the sum of the starting address and the byte count is greater than 64K, an error will be returned.

Warning
Do not include the I/O space in banks $00, $01, $ED, and $E1 in the source-address range of bytes to be written to the DOC RAM. If you do, you will access soft switches that will cause the system to crash.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcPtr</td>
</tr>
<tr>
<td>docStart</td>
</tr>
<tr>
<td>byteCount</td>
</tr>
</tbody>
</table>

Long—POINTER to data to be written from RAM

Word—Starting address of DOC buffer to receive data

Word—Number of bytes to be written

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
</tr>
</tbody>
</table>

Errors

$0810  noDOCFndErr  No DOC or RAM found
$0811  docAddrRngErr  DOC address range error

C

extern pascal void WriteRamBlock(srcPtr, docStart, byteCount)

Pointer    srcPtr;

Word      docStart;

Word      byteCount;

21-28  Sound Tool Set routines
Read register

Reads any register within the DOC. This call is made through the appropriate jump table address provided by a GetTableAddress call. (See the section "GetTableAddress" in this chapter.) The table provided by GetTableAddress includes a table of generator-to-oscillator translations. This table gives the number of the first oscillator in the pair for a generator. The number of the second oscillator equals the number of the first oscillator plus 1.

By getting the oscillator number that corresponds to a particular generator and adding it to the base register number of a register group, an application can find out the settings for an oscillator. A table of the DOC registers has been provided in the section "Sound Hardware" in this chapter.

This routine leaves the sound GLU in auto-increment and register access modes.

Parameters  The stack is not affected by this call. Instead, the following registers are used:

Relevant registers before call

- e = 0; native mode
- m = 1; 8-bit accumulator
- x = 0; 16-bit index registers
- X = DOC register to read

Relevant registers after call

Accumulator (8-bit) = Contents of specified DOC register

Errors  None

C  Call cannot be made from C.
(None) **Write register**

Writes a one-byte parameter to any register in the DOC. The call is made through the appropriate jump table address provided by a GetTableAddress call. To write to an oscillator register that corresponds to a generator, take the following steps:

1. Take the oscillator number from the oscillator table.
2. Add 1 to access the odd oscillator of the pair.
3. Add the base register of the specific register.
4. Make the Write Register call through the Write Register routine's address in the jump table.

This routine leaves the sound GLU in auto-increment and register access modes.

**Parameters**

The stack is not affected by this call. Instead, the following registers are used:

**Relevant registers before call**

- e = 0; native mode
- m = 1; 8-bit accumulator
- x = 0; 16-bit index registers
- Accumulator (8-bit) = data to write
- X = DOC register number

**Relevant registers after call**

None

**Errors**

None

**C**

Call cannot be made from C.

---

21-30 Sound Tool Set low-level routines
Read RAM

Reads any specified DOC RAM location.

**Warning**

Interrupts must be disabled whenever your application accesses the DOC RAM. Your application must disable interrupts before it accesses the RAM and then reenable them afterward.

This call is made through the appropriate jump table address provided by a GetTableAddress call. See the section “GetTableAddress” in this chapter. This routine leaves the sound GLU in auto-increment and RAM access modes.

**Important**

This routine does not do any type of error checking on the address or data.

**Parameters**

The stack is not affected by this call. Instead, the following registers are used:

**Relevant registers before call**

- e = 0; native mode
- m = 1; 8-bit accumulator
- x = 0; 16-bit index registers
- X = DOC RAM address to read

**Relevant registers after call**

None

**Errors**

- None

**C**

Call cannot be made from C.
Write RAM

Writes a one-byte value to any specified DOC RAM location.

Warning
Interrupts must be disabled whenever your application accesses the DOC RAM. Your application must disable interrupts before it accesses the RAM and then reenable them afterward.

This call is made through the appropriate jump table address provided by a GetTableAddress call. See the section “GetTableAddress” in this chapter. This routine leaves the sound GLU in auto-increment and RAM access modes.

Important
This routine does not do any type of error checking on the address or data.

Parameters
The stack is not affected by this call. Instead, the following registers are used:

Relevant registers before call
e = 0; native mode
m = 1; 8-bit accumulator
x = 0; 16-bit index registers
Accumulator (8-bit) = data to write
X = DOC RAM address to write to

Relevant registers after call
None

Errors
None

C
Call cannot be made from C.
Read Next

Reads the next location pointed to by the sound GLU address register.

**Warning**

Interrupts must be disabled whenever your application accesses the DOC RAM. Your application must disable interrupts before it accesses the RAM and then reenable them afterward.

This call is made through the appropriate jump table address provided by a GetTableAddress call.

**Important**

Before making the first Read Next call in a sequence, you must make a Read Register or Read RAM call with the register or address desired minus 2. This leaves the sound GLU control register set to auto-increment mode and the sound GLU address register pointing to the correct DOC register or address.

**Parameters**

The stack is not affected by this call. Instead, the registers listed below are used:

**Relevant registers before call**

None (all will have been set properly by the previous Read Register or Read RAM call)

**Relevant registers after call**

Accumulator (8-bit) = Data byte read

**Errors**

None

**C**

Call cannot be made from C.
Write Next

Writes one byte of data to the next DOC register or RAM location, depending on the setting of the sound GLU control register.

---

**Warning**

Interrupts must be disabled whenever your application accesses the DOC RAM. Your application must disable interrupts before it accesses the RAM and then re-enable them afterward.

---

**Important**

Before making the first Write Next call in a sequence, you must make a Read Register or Read RAM call with the register or address desired minus 2. This leaves the sound GLU control register set to auto-increment mode and the sound GLU address register pointing to the correct DOC register or address.

---

**Parameters**

The stack is not affected by this call. Instead, the following registers are used:

**Relevant registers before call**

- Accumulator (8-bit) = Data byte to write
- All others will have been set properly by the previous Read Register or Read RAM call

**Relevant registers after call**

- None

**Errors**

- None

**C**

- Call cannot be made from C.

---

21-34 Sound Tool Set low-level routines
**Disable Increment**

Disables the auto-increment mode set up by a Read Register, Write Register, Read RAM, or Write RAM low-level sound routine, thus allowing your application to read a DOC register or memory location continuously. Auto-increment mode remains disabled until your application makes another Read Register, Write Register, Read RAM, or Write RAM call.

For example, if you want to read the analog-to-digital converter, your application can make a Read Register call to Register $E2 and then make a Disable Increment call. Because auto-increment mode is disabled, your application can then make a Read Next call to read the A-to-D converter continuously.

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

**Errors**

None

**C**

Call cannot be made from C.
Sound Tool Set summary

This section briefly summarizes the constants, data structures, and tool set errors contained in the Sound Tool Set.

Important

These definitions are provided in the appropriate Interface file.

Table 21-6
Sound Tool Set constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jump table offsets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>readRegister</td>
<td>$00</td>
<td>Read Register routine</td>
</tr>
<tr>
<td>writeRegister</td>
<td>$04</td>
<td>Write Register routine</td>
</tr>
<tr>
<td>readRam</td>
<td>$08</td>
<td>Read RAM routine</td>
</tr>
<tr>
<td>writeRam</td>
<td>$0C</td>
<td>Write RAM routine</td>
</tr>
<tr>
<td>readNext</td>
<td>$10</td>
<td>Read Next routine</td>
</tr>
<tr>
<td>writeNext</td>
<td>$14</td>
<td>Write Next routine</td>
</tr>
<tr>
<td>oscTable</td>
<td>$18</td>
<td>Pointer to oscillator table</td>
</tr>
<tr>
<td>genTable</td>
<td>$1C</td>
<td>Pointer to generator table</td>
</tr>
<tr>
<td>gcbAddrTable</td>
<td>$20</td>
<td>Pointer to GCB address table</td>
</tr>
<tr>
<td>disableInc</td>
<td>$24</td>
<td>Disable Increment routine</td>
</tr>
<tr>
<td>Channel-generator-type word</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ffSynthMode</td>
<td>$0001</td>
<td>Free-Form Synthesizer mode</td>
</tr>
<tr>
<td>noteSynthMode</td>
<td>$0002</td>
<td>Note Synthesizer mode</td>
</tr>
<tr>
<td>Stop-sound mask</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gen0off</td>
<td>$0001</td>
<td>Generator 0 off</td>
</tr>
<tr>
<td>gen1off</td>
<td>$0002</td>
<td>Generator 1 off</td>
</tr>
<tr>
<td>gen2off</td>
<td>$0004</td>
<td>Generator 2 off</td>
</tr>
<tr>
<td>gen3off</td>
<td>$0008</td>
<td>Generator 3 off</td>
</tr>
<tr>
<td>gen4off</td>
<td>$0010</td>
<td>Generator 4 off</td>
</tr>
<tr>
<td>gen5off</td>
<td>$0020</td>
<td>Generator 5 off</td>
</tr>
<tr>
<td>gen6off</td>
<td>$0040</td>
<td>Generator 6 off</td>
</tr>
<tr>
<td>gen7off</td>
<td>$0080</td>
<td>Generator 7 off</td>
</tr>
<tr>
<td>gen8off</td>
<td>$0100</td>
<td>Generator 8 off</td>
</tr>
<tr>
<td>gen9off</td>
<td>$0200</td>
<td>Generator 9 off</td>
</tr>
<tr>
<td>gen10off</td>
<td>$0400</td>
<td>Generator 10 off</td>
</tr>
<tr>
<td>gen11off</td>
<td>$0800</td>
<td>Generator 11 off</td>
</tr>
<tr>
<td>gen12off</td>
<td>$1000</td>
<td>Generator 12 off</td>
</tr>
<tr>
<td>gen13off</td>
<td>$2000</td>
<td>Generator 13 off</td>
</tr>
<tr>
<td>gen14off</td>
<td>$4000</td>
<td>Generator 14 off</td>
</tr>
</tbody>
</table>
### Table 21-6 (continued)
**Sound Tool Set constants**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generator status word</td>
<td></td>
<td></td>
</tr>
<tr>
<td>genAvail</td>
<td>$0000</td>
<td>Generator available</td>
</tr>
<tr>
<td>ffSynth</td>
<td>$0100</td>
<td>Free-Form Synthesizer</td>
</tr>
<tr>
<td>noteSynth</td>
<td>$0200</td>
<td>Note Synthesizer</td>
</tr>
<tr>
<td>lastBlock</td>
<td>$8000</td>
<td>Last block of wave</td>
</tr>
</tbody>
</table>

### Table 21-7
**Sound Tool Set data structures**

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoundParamBlock</td>
<td></td>
<td>Pointer</td>
<td>Starting address of wave</td>
</tr>
<tr>
<td>waveStart</td>
<td>$00</td>
<td>Pointer</td>
<td>Waveform size in pages</td>
</tr>
<tr>
<td>waveSize</td>
<td>$04</td>
<td>Word</td>
<td>Output sample rate</td>
</tr>
<tr>
<td>freqOffset</td>
<td>$06</td>
<td>Word</td>
<td>DOC buffer start address; low-order byte = 0</td>
</tr>
<tr>
<td>docBuffer</td>
<td>$08</td>
<td>Word</td>
<td>DOC buffer start address; high-order byte = 0</td>
</tr>
<tr>
<td>bufferSize</td>
<td>$0A</td>
<td>Word</td>
<td>Pointer to start of next wave's parameter block</td>
</tr>
<tr>
<td>nextWavePtr</td>
<td>$0C</td>
<td>SoundPBPtr</td>
<td>DOC volume setting; high-order byte = 0</td>
</tr>
<tr>
<td>volSetting</td>
<td>$10</td>
<td>Word</td>
<td></td>
</tr>
</tbody>
</table>

*Note: The actual assembly-language equates have a lowercase letter o in front of all names given in this table.*

### Table 21-8
**Sound Tool Set error codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0810</td>
<td>noDOCFndErr</td>
<td>No DOC or RAM found</td>
</tr>
<tr>
<td>$0811</td>
<td>docAddrRngErr</td>
<td>DOC address range error</td>
</tr>
<tr>
<td>$0812</td>
<td>noSAppInitErr</td>
<td>No SoundStartUp call made</td>
</tr>
<tr>
<td>$0813</td>
<td>invalGenNumErr</td>
<td>Invalid generator number</td>
</tr>
<tr>
<td>$0814</td>
<td>synthModeErr</td>
<td>Synthesizer mode error</td>
</tr>
<tr>
<td>$0815</td>
<td>genBusyErr</td>
<td>Generator already in use</td>
</tr>
<tr>
<td>$0817</td>
<td>mstrIRQNotAssgnErr</td>
<td>Master IRQ not assigned</td>
</tr>
<tr>
<td>$0818</td>
<td>sndAlreadyStrtErr</td>
<td>Sound Tool Set already started</td>
</tr>
<tr>
<td>$08FF</td>
<td>unclaimedSndIntErr</td>
<td>Unclaimed sound interrupt error (reported through System Failure Manager)</td>
</tr>
</tbody>
</table>

---

**Sound Tool Set summary** 21-37
The Standard File Operations Tool Set provides the standard user interface for specifying a file to be opened or saved. The tool set provides dialog boxes that allow the user both to open and save a file on a disk in any drive and to change disks in a drive.

A preview of the Standard File Operations Tool Set routines

To introduce you to the capabilities of the Standard File Operations Tool Set, all of its routines are grouped by function and briefly described in Table 22-1. These routines are described in detail later in this chapter, where they are separated into housekeeping routines (discussed in routine number order) and the rest of the Standard File Operations routines (discussed in alphabetical order).
Table 22-1
Standard File Operations Tool Set routines and their functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Housekeeping routines</strong></td>
<td></td>
</tr>
<tr>
<td>SFBootInit</td>
<td>Initializes the Standard File Operations Tool Set; called only by the Tool</td>
</tr>
<tr>
<td>Locator—must not be called by an application</td>
<td></td>
</tr>
<tr>
<td>SFStartUp</td>
<td>Starts up the Standard File Operations Tool Set for use by an application</td>
</tr>
<tr>
<td>SFShutDown</td>
<td>Shuts down the Standard File Operations Tool Set</td>
</tr>
<tr>
<td>SFVersion</td>
<td>Returns the version number of the Standard File Operations Tool Set</td>
</tr>
<tr>
<td>SFRest</td>
<td>Resets the Standard File Operations Tool Set; called only when the system is reset—must not be called by an application</td>
</tr>
<tr>
<td>SFSstatus</td>
<td>Indicates whether the Standard File Operations Tool Set is active</td>
</tr>
<tr>
<td><strong>Other Standard File routines</strong></td>
<td></td>
</tr>
<tr>
<td>SFGetFile</td>
<td>Displays the standard Open File dialog box and returns information about the file selected by the user</td>
</tr>
<tr>
<td>SFPutFile</td>
<td>Displays the standard Save File dialog box and returns information about the name of the file to be saved</td>
</tr>
<tr>
<td>SFGetFile</td>
<td>Displays a custom Open File dialog box and returns information about the file selected by the user</td>
</tr>
<tr>
<td>SFPutFile</td>
<td>Displays a custom Save File dialog box and returns information about the name of the file to be saved</td>
</tr>
<tr>
<td>SFAIlCaps</td>
<td>Allows the application to decide if filenames will be displayed in all uppercase letters or in uppercase and lowercase letters</td>
</tr>
</tbody>
</table>
Standard dialog boxes

The standard Open File dialog box is produced by the SFGetFile routine and is illustrated in Figure 22-1.

Figure 22-1
Standard Open File dialog box

The standard Save File dialog box is produced by the SFPutFile routine and is illustrated in Figure 22-2.

Figure 22-2
Standard Save File dialog box
Standard File dialog templates

The Standard File Operations Tool Set allows you to provide custom dialog boxes for the Open File and Save File dialog boxes. To produce a custom dialog box, you use the SFPPutFile and SFPGetFile routines and provide a pointer to a dialog template in memory. A dialog template is a record passed to the Dialog Manager routine GetNewModalDialog. The template contains information about the dialog to be created, including a bounds rectangle and a list of pointers to item templates.

The following sections provide the templates that give the standard Open File and Save File dialog boxes. All of the templates depend on the following strings:

<table>
<thead>
<tr>
<th>Strings</th>
<th>Start</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveStr</td>
<td>'Save'</td>
</tr>
<tr>
<td>OpenStr</td>
<td>'Open'</td>
</tr>
<tr>
<td>CloseStr</td>
<td>'Close'</td>
</tr>
<tr>
<td>DriveStr</td>
<td>'Next Drive'</td>
</tr>
<tr>
<td>CancelStr</td>
<td>'Cancel'</td>
</tr>
<tr>
<td>FolderStr</td>
<td>'New Folder'</td>
</tr>
<tr>
<td>KbFreeStr</td>
<td>&quot;^0 free of ^1 K.&quot;</td>
</tr>
</tbody>
</table>

Note: Dialog Manager routine ParamText replaces "^0 and ^1 with real values from disk.

Templates for the standard Open File dialog box

For the Open File dialog box, the item part of the template must include the following items in this exact order:

<table>
<thead>
<tr>
<th>Item</th>
<th>Item type</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open button</td>
<td>buttonItem</td>
<td>1</td>
</tr>
<tr>
<td>Close button</td>
<td>buttonItem</td>
<td>2</td>
</tr>
<tr>
<td>Next button</td>
<td>buttonItem</td>
<td>3</td>
</tr>
<tr>
<td>Cancel button</td>
<td>buttonItem</td>
<td>4</td>
</tr>
<tr>
<td>Scroll bar</td>
<td>scrollBarItem</td>
<td>5</td>
</tr>
</tbody>
</table>

Note: The Standard File dialog box only allows standard scroll bar operations; it does not allow custom scroll bar routines.

<table>
<thead>
<tr>
<th>Path</th>
<th>userItem</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files</td>
<td>userItem</td>
<td>7</td>
</tr>
<tr>
<td>Prompt</td>
<td>userItem</td>
<td>8</td>
</tr>
</tbody>
</table>

Chapter 22: Standard File Operations Tool Set
640 mode

GetDialog640

start
using Strings

dc i'0,0,114,400'
dc i'-1'
dc i'0'
dc i'OpenBut640'
dc i'CloseBut640'
dc i'NextBut640'
dc i'CancelBut640'
dc i'Scroll640'
dc i'Path640'
dc i'Files640'
dc i'Prompt640'
dc i'0'

OpenBut640
dc i'1'
dc i'53,265,65,375'
dc i'ButtonItem'
dc i'OpenStr'
dc i'0'
dc i'0'
dc i'0'

CloseBut640
dc i'2'
dc i'71,265,83,375'
dc i'ButtonItem'
dc i'CloseStr'
dc i'0'
dc i'0'
dc i'0'

NextBut640
dc i'3'
dc i'27,265,39,375'
dc i'ButtonItem'
dc i'DriveStr'
dc i'0'
dc i'0'
dc i'0'

CancelBut640
dc i'4'
dc i'97,265,109,375'
dc i'ButtonItem'
dc i'CancelStr'
dc i'0'
dc i'0'
dc i'0'

Scroll640
dc i'5'
dc i'28,214,110,238'
dc i'ScrollBarItem'
dc i'0'
dc i'0'
dc i'3'
dc i'0'

Standard File dialog templates 22-5
320 mode

getDialog320

start

using Strings

dc i'0,0,114,260'
dc i'1'
dc i'0'
dc i'OpenBut320'
dc i'CloseBut320'
dc i'NextBut320'
dc i'CancelBut320'
dc i'Scroll320'
dc i'Path320'
dc i'Files320'
dc i'Prompt320'
dc i'0'

openBut320

dc i'1'
dc i'53,160,65,255'
dc i'ButtonItem'
dc i'OpenStr'
dc i'0'
dc i'0'
dc i'0'

22-6 Chapter 22: Standard File Operations Tool Set
CloseBut 320  dc  1'2'
          dc  1'71,160,83,255'
          dc  1'ButtonItem'
          dc  14'CloseStr'
          dc  1'0'
          dc  1'0'
          dc  14'0'

NextBut 320  dc  1'3'
          dc  1'27,160,39,255'
          dc  1'ButtonItem'
          dc  14'DriveStr'
          dc  1'0'
          dc  1'0'
          dc  14'0'

CancelButton 320  dc  1'4'
          dc  1'97,160,109,255'
          dc  1'ButtonItem'
          dc  14'CancelStr'
          dc  1'0'
          dc  1'0'
          dc  14'0'

Scroll 320  dc  1'5'
          dc  1'27,139,109,151'
          dc  1'ScrollBarItem'
          dc  14'0'
          dc  1'0'
          dc  1'3'
          dc  14'0'

Path 320  dc  1'6'
          dc  1'14,22,26,256'
          dc  1'UserItem'
          dc  14'0'
          dc  1'0'
          dc  1'0'
          dc  14'0'

Files 320  dc  1'7'
          dc  1'27,05,109,140'
          dc  1'UserItem'
          dc  14'0'
          dc  1'0'
          dc  1'0'
          dc  14'0'

Prompt 320  dc  1'8'
          dc  1'00,05,13,255'
          dc  1'UserItem+$8000'
          dc  14'0'
          dc  1'0'
          dc  1'0'
          dc  14'0'

end

Standard File dialog templates  22-7
Templates for the standard Save File dialog box

For the Save File dialog box, the item part of the template must include the following items in this exact order:

<table>
<thead>
<tr>
<th>Item</th>
<th>Item type</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save button</td>
<td>buttonItem</td>
<td>1</td>
</tr>
<tr>
<td>Open button</td>
<td>buttonItem</td>
<td>2</td>
</tr>
<tr>
<td>Close button</td>
<td>buttonItem</td>
<td>3</td>
</tr>
<tr>
<td>Next button</td>
<td>buttonItem</td>
<td>4</td>
</tr>
<tr>
<td>Cancel button</td>
<td>buttonItem</td>
<td>5</td>
</tr>
<tr>
<td>Scroll bar</td>
<td>scrollBarItem</td>
<td>6</td>
</tr>
<tr>
<td>Path</td>
<td>userItem</td>
<td>7</td>
</tr>
<tr>
<td>Files</td>
<td>userItem</td>
<td>8</td>
</tr>
<tr>
<td>Prompt</td>
<td>userItem</td>
<td>9</td>
</tr>
<tr>
<td>Filename</td>
<td>editText</td>
<td>10</td>
</tr>
<tr>
<td>Free space</td>
<td>statText</td>
<td>11</td>
</tr>
<tr>
<td>Create button</td>
<td>button</td>
<td>12</td>
</tr>
</tbody>
</table>

640 mode

PutDialog640

start

using Strings

dc i'0,0,120,320'
dc i'-1'
dc i'0'
dc i'SaveButP640'
dc i'OpenButP640'
dc i'CloseButP640'
dc i'NextButP640'
dc i'CancelButP640'
dc i'ScrollP640'
dc i'PathP640'
dc i'FilesP640'
dc i'PromptP640'
dc i'EditP640'
dc i'StatTextP640'
dc i'CreateButP640'
dc i'0'

SaveButP640
dc i'1'
dc i'87,204,99,310'
dc i'ButtonItem'
dc i'SaveStr'
dc i'0'
dc i'0'
dc i'0'
OpenButP640
  dc i'2'
  dc i'49,204,61,310'
  dc i'ButtonItem'
  dc i4'OpenStr'
  dc i'0'
  dc i'0'
  dc i4'0'

CloseButP640
  dc i'3'
  dc i'64,204,76,310'
  dc i'ButtonItem'
  dc i4'CloseStr'
  dc i'0'
  dc i'0'
  dc i4'0'

NextButP640
  dc i'4'
  dc i'15,204,27,310'
  dc i'ButtonItem'
  dc i4'DriveStr'
  dc i'0'
  dc i'0'
  dc i4'0'

CancelButP640
  dc i'5'
  dc i'104,204,116,310'
  dc i'ButtonItem'
  dc i4'CancelStr'
  dc i'0'
  dc i'0'
  dc i4'0'

ScrollP640
  dc i'6'
  dc i'26,169,88,194'
  dc i'ScrollBarItem'
  dc i4'0'
  dc i'0'
  dc i'3'
  dc i4'0'

PathP640
  dc i'7'
  dc i'00,10,12,315'
  dc i'UserItem'
  dc i4'0'
  dc i'0'
  dc i'0'
  dc i4'0'

FilesP640
  dc i'8'
  dc i'26,10,88,170'
  dc i'UserItem'
  dc i4'0'
  dc i'0'
  dc i'0'
  dc i4'0'

Standard File dialog templates
PromptP640   dc  i'9'  
               dc  i'88,10,100,200'  
               dc  i'UserItem+$8000'  
               dc  i'0'  
               dc  i'0'  
               dc  i'0'  
               dc  i'0'

EditP640    dc  i'10'  
               dc  i'100,10,118,194'  
               dc  i'EditLine'  
               dc  i'0'  
               dc  i'0'  
               dc  i'0'  
               dc  i'0'

StatTextP640  dc  i'11'  
                dc  i'12,10,22,200'  
                dc  i'StatText+$8000'  
                dc  i'KbFreeStr'  
                dc  i'0'  
                dc  i'0'  
                dc  i'0'  
                dc  i'0'

CreateButP640 dc  i'12'  
                dc  i'29,204,41,310'  
                dc  i'ButtonItem'  
                dc  i'FolderStr'  
                dc  i'0'  
                dc  i'0'  
                dc  i'0'  
                dc  i'0'

end

320 mode

PutDialog320 start 

using Strings 

dc  i'0,0,128,270'  
dc  i'-1' 
dc  i'0' 
dc  i'SaveButP320'  
dc  i'OpenButP320'  
dc  i'CloseButP320'  
dc  i'NextButP320'  
dc  i'CancelButP320'  
dc  i'ScrollP320'  
dc  i'PathP320'  
dc  i'FilesP320'  
dc  i'PromptP320'  
dc  i'EditP320'  
dc  i'StatTextP320'  
dc  i'CreateButP320'  
dc  i'0'
Standard File dialog templates 22-11
See Chapter 6, "Dialog Manager," in Volume 1 for further details.

The bounding rectangle for the Files user item determines how many files may be displayed at one time. You should set the height of this rectangle to 2 plus 10 times the number of files to show. A height of 122 would allow 12 files to be seen.

22-12 Chapter 22: Standard File Operations Tool Set
Using the Standard File Operations Tool Set

This section discusses how the Standard File Operations Tool Set routines fit into the general flow of an application and gives you an idea of which routines you'll need to use under normal circumstances. Each routine is described in detail later in this chapter.

The Standard File Operations Tool Set depends on the presence of the tool sets shown in Table 22-2 and requires that at least the indicated version of each tool set be present.

<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
<th>Minimum version needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01 #01</td>
<td>Tool Locator</td>
<td>1.0</td>
</tr>
<tr>
<td>$02 #02</td>
<td>Memory Manager</td>
<td>1.0</td>
</tr>
<tr>
<td>$03 #03</td>
<td>Miscellaneous Tool Set</td>
<td>1.0</td>
</tr>
<tr>
<td>$04 #04</td>
<td>QuickDraw II</td>
<td>1.0</td>
</tr>
<tr>
<td>$06 #06</td>
<td>Event Manager</td>
<td>1.0</td>
</tr>
<tr>
<td>$0E #14</td>
<td>Window Manager</td>
<td>1.3</td>
</tr>
<tr>
<td>$10 #16</td>
<td>Control Manager</td>
<td>1.3</td>
</tr>
<tr>
<td>$0F #15</td>
<td>Menu Manager</td>
<td>1.3</td>
</tr>
<tr>
<td>$14 #20</td>
<td>LineEdit Tool Set</td>
<td>1.0</td>
</tr>
<tr>
<td>$15 #21</td>
<td>Dialog Manager</td>
<td>1.1</td>
</tr>
</tbody>
</table>

To use the Standard File Operations Tool Set routines, your application must call the SFStartUp routine before making any other Standard File calls. To save memory space, you can choose to have your application make the SFStartUp call only when it needs to present the dialog boxes. Use the LoadOneTool routine in the Tool Locator if you wish to use this method. See the section “SFStartUp” in this chapter for an example.

Use the SFShutDown routine to shut down the Standard File Operations Tool Set after you have finished using it. If you wish, you can unload the tool set with the Tool Locator routine UnLoadOneTool, which will unload the tool set from memory and thus save space.

Important

If you choose to unload the Standard File Operations Tool Set, be sure to reload it with a LoadOneTool call before making the SFStartUp call again.
When the user makes a request to open a file, use the SFGetFile routine in your application to present the standard Open File dialog box and retrieve the file name. SFGetFile allows you to specify where the standard dialog box will be placed on the screen, specify the prompt at the top of the box, and filter the type of files to be displayed in the box. The routine does not allow you to modify the appearance of the box; if you wish to construct your own custom dialog box, use the SFPGetFile routine.

When the user makes a request to save a file, use the SFPutFile routine to present the standard dialog Save File dialog box. SFPutFile allows you to specify where the standard dialog box will be placed on the screen, the prompt at the top of the box, and the maximum number of characters the user may type.

Like SFGetFile, SFPutFile does not allow you to modify the appearance of the box; if you wish to construct your own custom dialog box, use the SFPPutFile routine.
$0117 SFBootInit

Initializes Standard File Operations Tool Set; called only by the Tool Locator.

Warning
An application must never make this call.

Parameters
The stack is not affected by this call. There are no input or output parameters.

Errors
None.

Call must not be made by an application.
$0217  **SFStartUp**

Starts up the Standard File Operations Tool Set for use by an application.

---

**Important**

Your application must make this call before it makes any other Standard File calls.

---

You may choose to have your application call SFStartUp only when it is needed, thus freeing memory for other uses. The number of the Standard File Operations Tool Set is $17, so a typical sequence of calls (in a pseudocode format) might be

```pseudocode
LoadOneTool ($17,$0101)
SFStartUp (appropriate parameters)
SFGetFile (appropriate parameters)
SFShutDown
UnloadOneTool ($17)
```

**Parameters**

**Stack before call**

| previous contents          | Word—ID number of application
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>user ID</td>
<td>Word—Bank $0 starting address for one page of direct-page space</td>
</tr>
<tr>
<td>dPageAddr</td>
<td>← SP</td>
</tr>
</tbody>
</table>

**Stack after call**

| previous contents          | ← SP                          |

**Errors**

None

**C**

```c
extern pascal void SFStartUp(userID, dPageAddr)

Word    userID;
Word    dPageAddr;
```

22-16  **Standard File Operations Tool Set housekeeping routines**
$0317  SFShutDown

Shuts down the Standard File Operations Tool Set. Your application may call SFShutDown immediately after Standard File Operations are completed, thus freeing memory for other uses.

---

**Important**

If your application has started up Standard File Operations, the application must make this call before it quits.

---

The number of the Standard File Operations Tool Set is $17, so a typical sequence of calls (in a pseudocode format) might be:

- LoadOneTool ($17,$0101)
- SFStartUp (appropriate parameters)
- SFGetFile (appropriate parameters)
- SFShutDown
- UnloadOneTool ($17)

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

**Errors**

None

**C**

```pascal
extern pascal void SFShutDown()
```
$0417 \textbf{SFVersion} \\
\textit{Returns the version number of the Standard File Operations Tool Set.}

\textbf{Parameters}

\textbf{Stack before call}

\begin{align*}
\text{previous contents} & \quad \text{wordspace} & \text{Word} & \text{Space for result} \\
\hline
\end{align*}

\textbf{Stack after call}

\begin{align*}
\text{previous contents} & \quad \text{versionInfo} & \text{Word} & \text{Version number of Standard File Operations Tool Set} \\
\hline
\end{align*}

\textbf{Errors} \quad \text{None}

\textbf{C}

\begin{verbatim}
extern pascal Word SFVersion()
\end{verbatim}

$0517 \textbf{SFReset} \\
\textit{Resets the Standard File Operations Tool Set; called only when the system is reset.}

\textbf{Warning}

\textit{An application must never make this call.}

\textbf{Parameters} \quad \text{The stack is not affected by this call. There are no input or output parameters.}

\textbf{Errors} \quad \text{None}

\textbf{C}

\begin{verbatim}
Call must not be made by an application.
\end{verbatim}
$0617  SFStatus

Indicates whether the Standard File Operations Tool Set is active.

SFStatus returns TRUE if SFStartUp has been called and SFShutDown has not been
called. The routine returns FALSE if SFStartUp has not been called at all or if
SFShutDown has been called since the last time SFStartUp was called.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
</tbody>
</table>

Word—Space for result

| ← SP |

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>activeFlag</td>
</tr>
</tbody>
</table>

Word—BOOLEAN; TRUE if Standard File active, FALSE if inactive

| ← SP |

**Errors**

None

**C**

extern pascal Boolean SFStatus()
$0D17 \textbf{SFAllCaps} \\

Allows an application to decide if filenames will be displayed in all uppercase letters or in uppercase and lowercase letters. If \texttt{allCapsFlag} is set to FALSE, the initial letter of the filename will be uppercase and the first letter after a period will be uppercase; all other letters will be lowercase.

\textbf{Parameters}

\textbf{Stack before call}

\begin{verbatim}
previous contents
allCapsFlag
\end{verbatim}

Word—BOOLEAN; TRUE for uppercase filenames, FALSE for mixed

\texttt{← SP}

\textbf{Stack after call}

\begin{verbatim}
previous contents
\end{verbatim}

\texttt{← SP}

\textbf{Errors} \\
None

\textbf{C}

\begin{verbatim}
extern pascal void SFAllCaps(allCapsFlag)

Boolean allCapsFlag;
\end{verbatim}
$0917 \textbf{SFGetFile}

Displays the standard Open File dialog box and returns information about the file selected by the user.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>whereX</td>
</tr>
<tr>
<td>whereY</td>
</tr>
<tr>
<td>promptPtr</td>
</tr>
<tr>
<td>filterProcPtr</td>
</tr>
<tr>
<td>typeListPtr</td>
</tr>
<tr>
<td>replyPtr</td>
</tr>
</tbody>
</table>

Word—INTEGER; X coordinate of upper left corner of dialog box
Word—INTEGER; Y coordinate of upper left corner of dialog box

Long—POINTER to string to display at top of dialog box

Long—POINTER to filter procedure; NIL for none
Long—POINTER to typelist record; NIL to display all files
Long—POINTER to a reply record

$\leftarrow \text{SP}$

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

$\leftarrow \text{SP}$

**Errors**

None

C

extern pascal void SFGetFile(whereX,whereY,promptPtr,
filterProcPtr,typeListPtr,replyPtr)

Integer whereX;
Integer whereY;
Pointer promptPtr;
WordProcPtr filterProcPtr;
Pointer typeListPtr;
SFReplyRecPtr replyPtr;

Standard File Operations Tool Set routines 22-21
You can also use the following alternate form of the call:

```plaintext
extern pascal void SFGetFile(where, promptPtr, filterProcPtr, typeListPtr, replyPtr)
```

More about parameters

The filter procedure pointed to by `filterProcPtr` determines which files will be displayed in the dialog box. Set `filterProcPtr` to NIL to prevent the procedure from being called. The filter procedure is called in full native mode, in the same way as one would call a Pascal function having one long parameter.

The calling sequence inside SFGetFile is as follows:

```plaintext
PushWord #0 ; Space for result
PushLong #DirEntry ; Pointer to directory entry ($27 bytes long)
jsl FilterProc
PopWord Result
```

The procedure must strip the four bytes of the directory entry off the stack and return with the result at the top of the stack. The result indicates what the procedure wants to do with the file, as shown in Table 22-3.

### Table 22-3
Filter procedure results

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>noDisplay</td>
<td>Don't display file</td>
</tr>
<tr>
<td>1</td>
<td>noSelect</td>
<td>Display file but don't allow user to select the file</td>
</tr>
<tr>
<td>2</td>
<td>displaySelect</td>
<td>Display file and allow user to select the file</td>
</tr>
</tbody>
</table>

The directory entry for the file is illustrated in Figure 22-3.
For more information on the fields of the directory entry, see the *Apple IIgs ProDOS 16 Reference*.

The `typeListPtr` parameter points to a record containing a list of file types to display (set the pointer to NIL to display all file types). The list has the form shown in Figure 22-4.

### Figure 22-3
File directory entry

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>storage_type</td>
<td>1 byte</td>
</tr>
<tr>
<td>1</td>
<td>name_length</td>
<td>15 bytes</td>
</tr>
<tr>
<td>2</td>
<td>file_name</td>
<td>1 byte</td>
</tr>
<tr>
<td>3</td>
<td>file_type</td>
<td>2 bytes</td>
</tr>
<tr>
<td>4</td>
<td>key_pointer</td>
<td>2 bytes</td>
</tr>
<tr>
<td>6</td>
<td>blocks_used</td>
<td>2 bytes</td>
</tr>
<tr>
<td>8</td>
<td>EOF</td>
<td>3 bytes</td>
</tr>
<tr>
<td>10</td>
<td>creation</td>
<td>4 bytes</td>
</tr>
<tr>
<td>12</td>
<td>date and</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>time</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>version</td>
<td>1 byte</td>
</tr>
<tr>
<td>18</td>
<td>min_version</td>
<td>1 byte</td>
</tr>
<tr>
<td>20</td>
<td>access</td>
<td>1 byte</td>
</tr>
<tr>
<td>22</td>
<td>aux_type</td>
<td>2 bytes</td>
</tr>
<tr>
<td>24</td>
<td>modification</td>
<td>4 bytes</td>
</tr>
<tr>
<td>26</td>
<td>date and</td>
<td></td>
</tr>
<tr>
<td></td>
<td>time</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>header_point</td>
<td>2 bytes</td>
</tr>
</tbody>
</table>

### Figure 22-4
Typelist record

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>numEntries</td>
<td>Byte—Total number of entries in list</td>
</tr>
<tr>
<td>1</td>
<td>fileType1</td>
<td>Byte—First file type to display</td>
</tr>
<tr>
<td>2</td>
<td>fileType2</td>
<td>Byte—Second file type to display</td>
</tr>
<tr>
<td></td>
<td>lastFileType</td>
<td>Byte—Last file type to display</td>
</tr>
</tbody>
</table>
If you specify both a `filterProcPtr` and a `typeListPtr`, only files of the right file type or types are passed on to the `filterProc` procedure.

The `replyPtr` parameter points to a reply record that has the form shown in Figure 22-5.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><code>good</code></td>
<td>Word—BOOLEAN: TRUE for open, FALSE for cancel</td>
</tr>
<tr>
<td>1</td>
<td><code>fileType</code></td>
<td>Word—ProDOS file type</td>
</tr>
<tr>
<td>2</td>
<td><code>auxFiletype</code></td>
<td>Word—ProDOS auxiliary file type</td>
</tr>
<tr>
<td>3</td>
<td><code>fileName</code></td>
<td>16 bytes—Name of file in prefix 0</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td><code>fullPathName</code></td>
<td>129 bytes—Full pathname of file in prefix 0</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 22-5**
Reply record

When the dialog box is displayed, the files from prefix 0 are shown. When the file is selected, the name of the file is returned in the record pointed to by `replyPtr` and prefix 0 is set to the directory containing the selected file. If the user cancels the operation, prefix 0 is left at whatever directory is being shown at the time.

*Macintosh programmers*: The Disk button works differently from the Drive button in the Macintosh. When a user clicks on the Disk button, Standard File first looks at the disk in the drive the current disk is expected to be in. If the current disk is no longer in that drive, the disk in that drive becomes the current disk. If the current disk is still there, the Disk button moves to the next disk in the ProDOS chain. The Disk button works this way because a user can change disks without the system knowing about it.
$0B17  **SFPGetFile**

Displays a custom Open File dialog box and returns information about the file selected by the user.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>whereX</td>
</tr>
<tr>
<td>whereY</td>
</tr>
<tr>
<td>promptPtr</td>
</tr>
<tr>
<td>filterProcPtr</td>
</tr>
<tr>
<td>typeListPtr</td>
</tr>
<tr>
<td>dlgTempPtr</td>
</tr>
<tr>
<td>dialogHookPtr</td>
</tr>
<tr>
<td>replyPtr</td>
</tr>
</tbody>
</table>

Word—INTEGER; X coordinate of upper left corner of dialog box

Word—INTEGER; Y coordinate of upper left corner of dialog box

Long—POINTER to string to display at top of dialog box

Long—POINTER to filter procedure; NIL for none

Long—POINTER to a typelist record; NIL to display all files

Long—POINTER to a dialog template in memory

Long—POINTER to a routine called every time ModalDialog returns a hit

Long—POINTER to a reply record

**Errors**

None

C

```c
extern pascal void SFPGetFile(whereX, whereY, promptPtr, filterProcPtr, typeListPtr, dlgTempPtr, dialogHookPtr, replyPtr)
```

Integer  whereX;

Integer  whereY;

Pointer  promptPtr;

WordProcPtr  filterProcPtr;

Pointer  typeListPtr;

DlgTempPtr  dlgTempPtr;
```

Standard File Operations Tool Set routines  22-25"
The `dlgTempPtr` and `dialogHookPtr` parameters

SFPGetFile works like SFGetFile, except for the addition of the `dlgTempPtr` and `dialogHookPtr` parameters. The `dlgTempPtr` provides a pointer to a dialog template; the template is a record passed to the Dialog Manager routine GetNewModalDialog. The template contains information about the dialog to be created, including a bounds rectangle and a list of pointers to item templates.

For the Open File dialog box, the item part of the template must include the following items in this exact order:

<table>
<thead>
<tr>
<th>Item</th>
<th>Item type</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open button</td>
<td>buttonItem</td>
<td>1</td>
</tr>
<tr>
<td>Close button</td>
<td>buttonItem</td>
<td>2</td>
</tr>
<tr>
<td>Next button</td>
<td>buttonItem</td>
<td>3</td>
</tr>
<tr>
<td>Cancel button</td>
<td>buttonItem</td>
<td>4</td>
</tr>
<tr>
<td>Scroll bar</td>
<td>scrollbarItem</td>
<td>5</td>
</tr>
</tbody>
</table>

- Note: The Standard File dialog box only allows standard scroll bar operations; it does not allow custom scroll bar routines.

Path          userItem  6
Files         userItem  7
Prompt        userItem  8

For more information and examples of dialog templates, see the section “Standard File Dialog Templates” in this chapter.

The `dialogHookPtr` parameter is a pointer to the routine called by SFPGetFile every time ModalDialog returns an item hit. The routine is passed a pointer to the dialog port and a pointer to the item-hit word. If the dialogHook routine wants to handle the item hit, it should handle it and set the hit to 0. If it wants SFPGetFile to handle the item hit, it should leave it unchanged.

The routine is called as follows:

```
PushLong #DialogPort
PushLong #ItemHit
jsl DialogHook
lda ItemHit
```

Your routine must be certain to strip the two longs (eight bytes) representing the DialogPort and ItemHit pointers off the stack before returning to SFPGetFile.
$0C17  SFPPutFile

Displays a custom Save File dialog box and returns information about the name of the file to be saved.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>whereX</td>
</tr>
<tr>
<td>whereY</td>
</tr>
<tr>
<td>promptPtr</td>
</tr>
<tr>
<td>origNamePtr</td>
</tr>
<tr>
<td>maxlen</td>
</tr>
<tr>
<td>dlgTempPtr</td>
</tr>
<tr>
<td>dialogHookPtr</td>
</tr>
<tr>
<td>replyPtr</td>
</tr>
</tbody>
</table>

- Word—INTEGER; X coordinate of upper left corner of dialog box
- Word—INTEGER; Y coordinate of upper left corner of dialog box
- Long—POINTER to string to display at top of dialog box
- Long—POINTER to string holding name that appears as default
- Word—INTEGER; maximum number of characters user may type
- Long—POINTER to a dialog template in memory
- Long—POINTER to a routine called every time ModalDialog returns a hit
- Long—POINTER to a reply record

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

Errors

None

Standard File Operations Tool Set routines  22-27
extern pascal void SFPPutFile(whereX, whereY, promptPtr, origNamePtr, maxLen, dlgTempPtr, dialogHookPtr, replyPtr)

Integer whereX;
Integer whereY;
Pointer promptPtr;
Pointer origNamePtr;
unsigned int maxLen;
DlgTempPtr dlgTempPtr;
VoidProcPtr dialogHookPtr;
SFReplyRecPtr replyPtr;

You can also use the following alternate form of the call:

extern pascal void SFPPutFile(where, promptPtr, origNamePtr, maxLen, dlgTempPtr, dialogHookPtr, replyPtr)

Point where;
Pointer promptPtr;
Pointer origNamePtr;
unsigned int maxLen;
DlgTempPtr dlgTempPtr;
VoidProcPtr dialogHookPtr;
SFReplyRecPtr replyPtr;
The **dlgTempPtr** and **dialogHookPtr** parameters

This routine works like SFPutFile, except for the addition of the **dlgTempPtr** and **dialogHookPtr** parameters. The **dlgTempPtr** provides a pointer to a dialog template; the template is a record passed to the Dialog Manager routine GetNewModalDialog. The template contains information about the dialog to be created, including a bounds rectangle and a list of pointers to item templates.

The item part of the template must include the following items in this exact order:

<table>
<thead>
<tr>
<th>Item</th>
<th>Item type</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save button</td>
<td>buttonItem</td>
<td>1</td>
</tr>
<tr>
<td>Open button</td>
<td>buttonItem</td>
<td>2</td>
</tr>
<tr>
<td>Close button</td>
<td>buttonItem</td>
<td>3</td>
</tr>
<tr>
<td>Next button</td>
<td>buttonItem</td>
<td>4</td>
</tr>
<tr>
<td>Cancel button</td>
<td>buttonItem</td>
<td>5</td>
</tr>
<tr>
<td>Scroll bar</td>
<td>scrollBarItem</td>
<td>6</td>
</tr>
<tr>
<td>Path</td>
<td>userItem</td>
<td>7</td>
</tr>
<tr>
<td>Files</td>
<td>userItem</td>
<td>8</td>
</tr>
<tr>
<td>Prompt</td>
<td>userItem</td>
<td>9</td>
</tr>
<tr>
<td>Filename</td>
<td>editItem</td>
<td>10</td>
</tr>
<tr>
<td>Free space</td>
<td>statText</td>
<td>11</td>
</tr>
<tr>
<td>Create button</td>
<td>button</td>
<td>12</td>
</tr>
</tbody>
</table>

For more information and examples of dialog templates, see the section “Standard File Dialog Templates” in this chapter and Chapter 6, “Dialog Manager,” in Volume 1 for further details.

The bounding rectangle for the Files user item determines how many files may be displayed. You should set the height of this rectangle to 2 plus 10 times the number of files to show at one time. A height of 122 would allow 12 files to be seen at one time.

The **dialogHookPtr** parameter is a pointer to the routine called by SFPutFile every time ModalDialog returns an item hit. The routine is passed a pointer to the dialog port and a pointer to the item-hit word. If the DialogHook routine wants to handle the item hit, it should handle it and set the hit to 0. If the DialogHook routine wants SFPutFile to handle the item hit, it should leave the hit unchanged.

The routine is called as follows in assembly language:

```
PushLong #DialogPort
PushLong #ItemHit
jsl DialogHook
lda ItemHit
```

Your routine must be certain to strip the two longs (eight bytes) representing the DialogPort and the ItemHit off the stack before returning to SFPutFile.
Displays the standard Save File dialog box and returns information about the name of the file to be saved.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>whereX</th>
<th>whereY</th>
<th>promptPtr</th>
<th>origNamePtr</th>
<th>maxlen</th>
<th>replyPtr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Word—INTEGER; X coordinate of upper left corner of dialog box</td>
<td>Word—INTEGER; Y coordinate of upper left corner of dialog box</td>
<td>Long—POINTER to string to display at top of dialog box</td>
<td>Long—POINTER to string holding name that appears as default</td>
<td>Word—INTEGER; maximum number of characters user may type</td>
<td>Long—POINTER to a reply record</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Errors

None

C

```c
extern pascal void SFPutFile(whereX, whereY, promptPtr, origNamePtr, maxlen, replyPtr)
Integer whereX;
Integer whereY;
Pointer promptPtr;
Pointer origNamePtr;
unsigned int maxlen;
SFRreplyRecPtr replyPtr;
```
You can also use the following alternate form of the call:

```pascal
extern pascal void SFPutFile(where, promptPtr, origNamePtr, maxlen, replyPtr)
Point where;
Pointer promptPtr;
Pointer origNamePtr;
unsigned int maxlen;
SFReplyRecPtr replyPtr;
```

### More about parameters

The `maxlen` parameter specifies the maximum number of characters a user may type. Most applications will use 15 for this value, but if the application wants to add a suffix to the file name, the `maxlen` value must be shortened. Values greater than 15 are not valid.

The `replyPtr` points to the same type of reply record as that of SFGetFile. See Figure 22-5 in the section “SFGetFile” in this chapter.

When the dialog box is first displayed, all the files from prefix 0 are shown. This lets the user know what names are in use and prevents use of a name that already exists. The user can open any directories shown as selectable; files not shown as selectable cannot be opened. When the user clicks on the Save button or presses the Return key, the name of the file is returned in the reply record and prefix 0 is set to the directory containing the selected file. If the user cancels the operation, prefix 0 is restored to its original state.

SFPutFile also checks to see if the file already exists. If it does, SFPutFile displays a dialog asking if it is OK to destroy the existing file.

If the user cancels the operation, prefix 0 is left at whatever directory is being shown at the time.

- **Macintosh programmers**: The Disk button works differently from the Drive button in the Macintosh. When a user clicks on the Disk button, Standard File first looks at the disk in the drive the current disk is expected to be in. If the current disk is no longer in that drive, the disk in that drive becomes the current disk. If the current disk is still there, the Disk button moves to the next disk in the ProDOS chain. The Disk button works this way because a user can change disks without the system knowing about it.

---

Standard File Operations Tool Set routines 22-31
Standard File Operations Tool Set summary

This section briefly summarizes the constants and data structures contained in the Standard File Operations Tool Set. There are no tool set error codes for the Standard File Operations Tool Set.

Important

These definitions are provided in the appropriate Interface file.

Table 22-4
Standard File Operations Tool Set constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter procedure results</td>
<td></td>
<td></td>
</tr>
<tr>
<td>noDisplay</td>
<td>$0000</td>
<td>Don't display file</td>
</tr>
<tr>
<td>noSelect</td>
<td>$0001</td>
<td>Display file, but don't allow user to select it</td>
</tr>
<tr>
<td>displaySelect</td>
<td>$0002</td>
<td>Display file and allow user to select it</td>
</tr>
</tbody>
</table>

Table 22-5
Standard File Operations Tool Set data structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reply record</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>good</td>
<td>$00</td>
<td>Boolean</td>
<td>TRUE for Open, FALSE for Cancel</td>
</tr>
<tr>
<td>fileType</td>
<td>$02</td>
<td>Word</td>
<td>ProDOS file type</td>
</tr>
<tr>
<td>auxFileType</td>
<td>$04</td>
<td>Word</td>
<td>ProDOS aux file type</td>
</tr>
<tr>
<td>fileName</td>
<td>$06</td>
<td>16 bytes</td>
<td>Name of the file in prefix .0</td>
</tr>
<tr>
<td>fullPathname</td>
<td>$16</td>
<td>129 bytes</td>
<td>Full pathname of selected file</td>
</tr>
</tbody>
</table>

Note: The actual assembly-language equates have a lowercase letter o in front of all of the names given in this table.
The **Text Tool Set** provides an interface between Apple II character device drivers, which must be executed in emulation mode, and new applications running in native mode. It also provides a means of redirection of I/O through RAM-based drivers. The Text Tool Set makes it possible to deal with the text screen without switching 65816 processor modes and moving to bank zero. Dispatches to RAM-based drivers will occur in full native mode.

### A preview of the Text Tool Set routines

To introduce you to the capabilities of the Text Tool Set, all of its routines are grouped by function and briefly described in Table 23-1. These routines are described in detail later in this chapter, where they are separated into housekeeping routines (discussed in routine number order) and the rest of the Text Tool Set routines (discussed in alphabetical order).

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Housekeeping routines</strong></td>
<td></td>
</tr>
<tr>
<td>TextBootInit</td>
<td>Initializes the Text Tool Set; called only by the Tool Locator—must not be called by an application</td>
</tr>
<tr>
<td>TextStartUp</td>
<td>Starts up the Text Tool Set for use by an application</td>
</tr>
<tr>
<td>TextShutDown</td>
<td>Shuts down the Text Tool Set when an application quits</td>
</tr>
<tr>
<td>TextVersion</td>
<td>Returns the version number of the Text Tool Set</td>
</tr>
<tr>
<td>TextReset</td>
<td>Resets the Text Tool Set; called only when the system is reset—must not be called by an application</td>
</tr>
<tr>
<td>TextStatus</td>
<td>Indicates whether the Text Tool Set is active</td>
</tr>
</tbody>
</table>

(continued)
<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Text global functions</strong></td>
<td></td>
</tr>
<tr>
<td>SetInGlobals</td>
<td>Sets the global parameters for the input device</td>
</tr>
<tr>
<td>SetOutGlobals</td>
<td>Sets the global parameters for the output device</td>
</tr>
<tr>
<td>SetErrGlobals</td>
<td>Sets the global parameters for the error output device</td>
</tr>
<tr>
<td>GetInGlobals</td>
<td>Returns the current values for the input device global parameters</td>
</tr>
<tr>
<td>GetOutGlobals</td>
<td>Returns the current values for the output device global parameters</td>
</tr>
<tr>
<td>GetErrGlobals</td>
<td>Returns the current values for the error output device global parameters</td>
</tr>
<tr>
<td><strong>I/O directing routines</strong></td>
<td></td>
</tr>
<tr>
<td>SetInputDevice</td>
<td>Sets the input device to a specified type and location</td>
</tr>
<tr>
<td>SetOutputDevice</td>
<td>Sets the output device to a specified type and location</td>
</tr>
<tr>
<td>SetErrorDevice</td>
<td>Sets the error output device to a specified type and location</td>
</tr>
<tr>
<td>GetInputDevice</td>
<td>Returns the type of driver installed as the input device</td>
</tr>
<tr>
<td>GetOutputDevice</td>
<td>Returns the type of driver installed as the output device</td>
</tr>
<tr>
<td>GetErrorDevice</td>
<td>Returns the type of driver installed as the error output device</td>
</tr>
<tr>
<td><strong>Text routines</strong></td>
<td></td>
</tr>
<tr>
<td>InitTextDev</td>
<td>Initializes a specified text device</td>
</tr>
<tr>
<td>CtlTextDev</td>
<td>Passes a control code to a specified text device</td>
</tr>
<tr>
<td>StatusTextDev</td>
<td>Executes a status call to a specified text device</td>
</tr>
<tr>
<td>WriteChar</td>
<td>Combines a specified character with the output global AND mask and OR mask and writes the character to the output text device</td>
</tr>
<tr>
<td>ErrWriteChar</td>
<td>Combines a specified character with the output global AND mask and OR mask and writes the character to the error output text device</td>
</tr>
<tr>
<td>WriteLine</td>
<td>Combines a pointed-to Pascal-type string (first byte of string specifies length) with the output global masks, concatenates a carriage return (for BASIC or RAM-based drivers) or a carriage return and line feed (for Pascal drivers), and then writes the string to the output text device</td>
</tr>
<tr>
<td>ErrWriteLine</td>
<td>Combines a pointed-to Pascal-type string (first byte of string specifies length) with the output global masks and then writes the string to the error output text device</td>
</tr>
<tr>
<td>WriteString</td>
<td>Combines a pointed-to Pascal-type string (first byte of string specifies length) with the output global masks and then writes the string to the output text device</td>
</tr>
<tr>
<td>ErrWriteString</td>
<td>Combines a pointed-to Pascal-type string (first byte of string specifies length) with the output global masks and then writes the string to the error output text device</td>
</tr>
<tr>
<td>TextWriteBlock</td>
<td>Combines a specified character string with the output global masks and then writes the string to the output text device</td>
</tr>
<tr>
<td>ErrWriteBlock</td>
<td>Combines a specified character string with the output global masks and then writes the string to the error output text device</td>
</tr>
<tr>
<td>WriteCString</td>
<td>Combines a pointed-to C string (string terminates with $00) with the output global masks and then writes the string to the output text device</td>
</tr>
<tr>
<td>ErrWriteCString</td>
<td>Combines a pointed-to C string (string terminates with $00) with the output global masks and then writes the string to the error output text device</td>
</tr>
</tbody>
</table>
Table 23-1 (continued)
Text Tool Set routines and their functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadChar</td>
<td>Reads a character obtained from the input text device, combines it with the input global masks, and returns the character on the stack</td>
</tr>
<tr>
<td>ReadLine</td>
<td>Reads a character string from the input text device, combines it with the input global masks, and writes the string to a specified memory location</td>
</tr>
<tr>
<td>TextReadBlock</td>
<td>Reads a block of characters from the input text device, combines it with the input global masks, and writes the block to a specified memory location</td>
</tr>
</tbody>
</table>

Using the I/O directing routines

The I/O directing routines direct I/O from the Text Tool Set to a specific type of character device driver or get information about the directing of a specific I/O driver. The types of character device drivers are listed in Table 23-2.

Table 23-2
Character device driver types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>BASIC device driver</td>
</tr>
<tr>
<td>1</td>
<td>Pascal device driver</td>
</tr>
<tr>
<td>2</td>
<td>RAM-based device driver</td>
</tr>
<tr>
<td>23</td>
<td>Illegal driver type</td>
</tr>
</tbody>
</table>

BASIC device drivers must support the standard Apple II BASIC device driver entry points (INIT, INPUT, and OUTPUT).

*Note:* BASIC devices use the Apple II I/O hooks ($36–$39) in absolute zero page.

Any desk accessories using a Text Tool BASIC device driver should save and restore the global masks, device descriptors, and I/O hooks when entering or leaving the desk accessory.

Pascal device drivers must support the standard Apple II Pascal 1.1 device driver entry points (INIT, READ, WRITE, and status). The optional Pascal 1.1 control entry point is supported by the Text Tool Set but does not necessarily have to be supported by the device. The application must find out if the device supports the optional Pascal entry points.
RAM-based device drivers may be located at any address and in any bank. A RAM-based driver must support five entry points, as follows:

- **RAMDRIVER Base Address**
- **RAMDRIVER Base Address + 3**
- **RAMDRIVER Base Address + 6**
- **RAMDRIVER Base Address + 9**
- **RAMDRIVER Base Address + 12**

<table>
<thead>
<tr>
<th>Address Increment</th>
<th>Entry Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>INIT (initialization) entry point</td>
</tr>
<tr>
<td>3</td>
<td>READ entry point</td>
</tr>
<tr>
<td>6</td>
<td>WRITE entry point</td>
</tr>
<tr>
<td>9</td>
<td>STATUS entry point</td>
</tr>
<tr>
<td>12</td>
<td>CONTROL entry point</td>
</tr>
</tbody>
</table>

RAM-based drivers are called by the Text Tool Set in native mode (m and x bits set to 16 bits) and should return to the Text Tools via an RTI instruction. The Text Tools pass data or ASCII characters to the RAM-based driver via the low-order byte of the 16-bit accumulator.

A RAM-based driver should not make any assumptions about the state of the data bank register or the direct page register. I/O performed by RAM-based drivers should operate on a single-character basis. The Text Tool Set, not the RAM-based driver, manipulates the text with regard to the global masks before passing the text to the RAM-based driver. All Text Tool Set routines that interface to different string types are supported by the tool set and not the device.

### Using the text routines

The following example demonstrates how to set up global parameters, initialize the text devices, send output to the error and output devices, and read input from the input device.

```
list on
instime off
absaddr on
gen off
symbol off
keep testit
65816 on
msb on
mcopy mm.macros
mcopy misc.macros
mcopy tt.macros
test start
```
C2 30  rep  #30
longa on
longi on

_TextStartUp

bcc NoErr0 ; If no error
jmp ToolError ; else

NoErr0 anop

pea $00FF  ; AND mask
pea $0080  ; OR mask

_SetInGlobals

bcc NoErr1 ; If no error
jmp ToolError ; else

NoErr1 anop

pea $00FF  ; AND mask
pea $0080  ; OR mask

_SetOutGlobals

bcc NoErr2 ; If no error
jmp ToolError ; else

NoErr2 anop

pea $00FF  ; AND mask
pea $0080  ; OR mask

_SetErrGlobals

bcc NoErr3 ; If no error
jmp ToolError ; else

NoErr3 anop

pea $0000  ; 0 = basic driver
pea $0000  ; in slot #3
pea $0003  ;

_SetInputDevice

bcc NoErr4 ; If no error
jmp ToolError ; else

Using the text routines 23-5
NoErr4 anop
pea $0000 ; 0 = basic driver
pea $0000 ; in slot #3
pea $0003 ;
/SetOutputDevice
bcc NoErr5 ; If no error
jmp ToolError ; else

NoErr5 anop
pea $0000 ; 0 = basic driver
pea $0000 ; in slot #1
pea $0001 ;
/SetErrorDevice
bcc NoErr6 ; If no error
jmp ToolError ; else

NoErr6 anop
pea $0000 ; Initialize the input device
_InitTextDev
bcc NoErr7 ; If no error
jmp ToolError ; else

NoErr7 anop
pea $0001 ; Initialize the output device
_InitTextDev
bcc NoErr8 ; If no error
jmp ToolError ; else

NoErr8 anop
pea $0002 ; Initialize the error device
_InitTextDev
bcc NoErr9 ; If no error
jmp ToolError ; else

23-6 Chapter 23: Text Tool Set
NoErr9  anop
  pha          ; Space for result
  pea         $0000  ; Don't echo
  _ReadChar
  bcc  NoErr10  ; Wait for any key
  jmp  ToolError ; If no error
NoErr10 anop
  pushlong  #$buffer  ; Pointer to input buffer
  pea        $0080   ; Offset into buffer
  pea        $0100   ; 256 characters is max block size
  pea        $0001   ; Echo characters to output device
  _ReadBlock
  bcc  NoErr11  ; Wait for any key
  jmp  ToolError ; If no error
NoErr11 anop
  pushlong  #$outstring  ; Pointer to the output device string
  _WriteCString
  bcc  NoErr12  ; and send it out
  jmp  ToolError ; If no error
NoErr12 anop
  pushlong  #$ErrString ; Pointer to the error device string
  _ErrWriteCString  ; and send it out
  bcc  NoErr13  ; If no error
  jmp  ToolError ; else
NoErr13 anop
  rts
ToolError anop

pha          ; Error code for display
pushlong    #FailMsg         ; Pointer to error message
_SysFailMgr
outstring   anop

dc          c'This string should go on the screen'
dc          h'0D'            ; A carriage return (basic = auto lf)
dc          h'00'
erstring    anop

dc          c'This string should go on the paper'
dc          h'0D'            ; A carriage return (basic = auto lf)
dc          h'00'
msb         off

FailMsg anop

dc          h'22'            ; Character count
dc          c'A tool call returned with error = '
end
Using the Text Tool Set

This section discusses how the Text Tool Set routines fit into the general flow of an application and gives you an idea of which routines you'll need to use under normal circumstances. Each routine is described in detail later in this chapter.

The Text Tool Set depends on the presence of the tool sets shown in Table 23-3 and requires that at least the indicated version of the tool set be present.

<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
<th>Minimum version needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01  #01</td>
<td>Tool Locator</td>
<td>1.0</td>
</tr>
<tr>
<td>$02  #02</td>
<td>Memory Manager</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Your application should make a TextStartUp call before making any other Text Tool Set calls.

*Note: At the time of publication, the TextStartUp call was not an absolute requirement because the Tool Locator was automatically starting up the Text Tool Set at boot time. However, you should make the call anyway to guarantee that your application remains compatible with all future versions of the system.*

Your application should also make the TextShutDown call when the application quits.

The Text Tool Set has global routines that are used to set or read the current global parameters used by RAM and the Pascal and BASIC text tools. The tool set also has I/O directing routines that direct I/O from the tool set to a specific type of character device driver or get information about directing a specific I/O driver. Finally, the tool set has text routines that interface with any BASIC, Pascal 1.1, or RAM-based character device driver.
$010C  **TextBootInit**

Initialize the Text Tool Set; called only by the Tool Locator.

---

**Warning**

An application must never make this call.

---

TextBootInit sets up the default device parameters as follows:

- Input device type is BASIC.
- Output device type is BASIC.
- Error output device type is BASIC.
- Input device resides in slot #3.
- Output device resides in slot #3.
- Error output device resides in slot #3.
- Global input AND mask is set to $FF.
- Global input OR mask is set to $80.
- Global output AND mask is set to $FF.
- Global output OR mask is set to $80.
- Global error output AND mask is set to $FF.
- Global error output OR mask is set to $80.

---

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

---

**Errors**

None

---

**C**

Call must not be made by an application.
TextStartUp

Starts up the Text Tool Set for use by an application. Your application should make a TextStartUp call before making any other Text Tool Set calls.

Note: At the time of publication, the TextStartUp call was not an absolute requirement because the Tool Locator was automatically starting up the Text Tool Set at boot time. However, you should make the call anyway to guarantee that your application remains compatible with all future versions of the system.

Parameters

The stack is not affected by this call. There are no input or output parameters.

Errors

None

C

extern pascal void TextStartUp()

TextShutDown

Shuts down the Text Tool Set when an application quits.

Important

If your application has started up the Text Tool Set, the application must make this call before it quits.

Parameters

The stack is not affected by this call. There are no input or output parameters.

Errors

None

C

extern pascal void TextShutDown()
$040C \textbf{TextVersion}

Returns the version number of the Text Tool Set.

**Parameters**

**Stack before call**

\[
\begin{array}{c|c}
\text{previous contents} & \text{Word—Space for result} \\
\hline
\text{wordspace} & \leftarrow \text{SP}
\end{array}
\]

**Stack after call**

\[
\begin{array}{c|c}
\text{previous contents} & \text{Word—Version number of Text Tool Set} \\
\hline
\text{versionInfo} & \leftarrow \text{SP}
\end{array}
\]

**Errors**

None

**C**

\[
\text{extern pascal Word TextVersion();}
\]
**TextReset**

Resets the Text Tool Set; called only when the system is reset.

---

**Warning**

An application must never make this call.

---

Resets the device parameters to the default values as follows:

- Input device type is BASIC.
- Output device type is BASIC.
- Error output device type is BASIC.
- Input device resides in slot #3.
- Output device resides in slot #3.
- Error output device resides in slot #3.
- Global input AND mask is set to $FF.
- Global input OR mask is set to $80.
- Global output AND mask is set to $FF.
- Global output OR mask is set to $80.
- Global error output AND mask is set to $FF.
- Global error output OR mask is set to $80.

---

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

---

**Errors**

None

---

Call must not be made by an application.
$060C  TextStatus

Indicates whether the Text Tool Set is active.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Word — Space for result</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
<td></td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Word — BOOLEAN; TRUE if Text Tool Set active, FALSE if inactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>activeFlag</td>
<td></td>
</tr>
</tbody>
</table>

Errors

None

C

extern pascal Boolean TextStatus()
$160C \textbf{CtlTextDev}

Passes a control code to a specified text device. The control codes passed depend on the device used and are outside the scope of this manual.

For Pascal device drivers, this is an optional entry point that may not be supported by a particular Pascal device. Because BASIC devices do not support this routine, an error occurs if this call is made to a BASIC device.

**Parameters**

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceNum</td>
</tr>
<tr>
<td>controlCode</td>
</tr>
</tbody>
</table>

Word—Device to control (0 = input, 1 = output, 2 = error output)

Word—Control code (in low-order byte) to pass to device

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

$0C01 \text{badDevType} \quad \text{Illegal device type}

*Note:* The remaining errors should occur only for Pascal devices.

$0C02 \text{badDevNum} \quad \text{Illegal device number}

$0C03 \text{badMode} \quad \text{Illegal operation}

$0C04 \text{unDefHW} \quad \text{Undefined hardware error}

$0C05 \text{lostDev} \quad \text{Lost device: device no longer on-line}

$0C06 \text{lostFile} \quad \text{File no longer in diskette directory}

$0C07 \text{badTitle} \quad \text{Illegal filename}

$0C08 \text{noRoom} \quad \text{Insufficient space on specified diskette}

$0C09 \text{noDevice} \quad \text{Specified volume not on-line}

$0C0A \text{noFile} \quad \text{Specified file not in directory of specified volume}

$0C0B \text{dupFile} \quad \text{Duplicate file: attempt to rewrite a file when a file of that name already exists}

$0C0C \text{notClosed} \quad \text{Attempt to open file that is already open}
$0C0D  notOpen  Attempt to access a closed file
$0C0E  badFormat  Error in reading real or integer number
$0C0F  ringBuffOFlo  Ring buffer overflow: characters arriving faster than the input buffer can accept them
$0C10  writeProtected  Specified diskette is write protected
$0C40  devErr  Device error: device failed to complete a read or write correctly

extern pascal void CtlTextDev(deviceNum, controlCode);

Word   deviceNum;
Word   controlCode;

Text Tool Set routines
$1F0C  ErrWriteBlock

Combines a specified character string with the output global masks and then writes the string to the error output text device. The string is specified by the parameters \texttt{textPtr + offset}, with the length specified by the \texttt{count} parameter.

**Parameters**

**Stack before call**

\begin{verbatim}
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>textPtr</td>
</tr>
<tr>
<td>offset</td>
</tr>
<tr>
<td>count</td>
</tr>
</tbody>
</table>
\end{verbatim}

- \texttt{Long}—POINTER to start of ASCII text
- \texttt{Word}—Offset within ASCII text
- \texttt{Word}—String length

**Stack after call**

\begin{verbatim}
| previous contents |
\end{verbatim}

- \texttt{SP}

**Errors**

Pascal device errors

Any of the errors $0C02$–$0C40$ as listed in the section "Text Tool Set Summary" at the end of this chapter

**C**

\begin{verbatim}
extern pascal void ErrWriteBlock(textPtr, offset, count)

Pointer   textPtr;
Word      offset;
Word      count;
\end{verbatim}
ErrWriteChar

Combines a specified character with the output global AND mask and OR mask and writes the character to the error output text device.

Parameters

Stack before call

\[
\begin{array}{c|c}
\text{previous contents} & \text{Word—Character to write (in low-order byte)} \\
\hline
\text{theChar} & \leftarrow \text{SP}
\end{array}
\]

Stack after call

\[
\begin{array}{c|c}
\text{previous contents} & \leftarrow \text{SP}
\end{array}
\]

Errors

Pascal device errors

Any of the errors $0C02$–$0C40$ as listed in the section "Text Tool Set Summary" at the end of this chapter

C

\[
\text{extern pascal void } \text{ErrWriteChar}(\text{theChar})
\]

\[
\text{Word } \text{theChar};
\]
$210C  ErrWriteCString

Combines a pointed-to C-type string (string terminates with $00) with the output global masks and then writes the string to the error output text device.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>cStrPtr</td>
</tr>
</tbody>
</table>
| ← SP

Stack after call

| previous contents |
| ← SP

Errors

Pascal device errors

Any of the errors $0C02-$0C40 as listed in the section "Text Tool Set Summary" at the end of this chapter

C

extern pascal void ErrWriteCString(cStrPtr)

Pointer  cStrPtr;
$1B0C \textbf{ErrWriteLine} \\

Combines a pointed-to Pascal-type string (first byte of string specifies length) with the output global masks and then writes the string to the error output text device. For BASIC and RAM-based drivers, the routine concatenates a carriage return to the string. For Pascal drivers, the routine concatenates a carriage return and a line feed to the string.

\textbf{Parameters}

\textbf{Stack before call}

\begin{tabular}{l|l}
\textit{previous contents} & \textbf{Long}—POINTER to Pascal-type string \\
\hline
\textit{strPtr} & \leftarrow SP \\
\end{tabular}

\textbf{Stack after call}

\begin{tabular}{l|l}
\textit{previous contents} & \leftarrow SP \\
\end{tabular}

\textbf{Errors}

Pascal device errors \hspace{1cm} Any of the errors $\text{OC02}-\text{OC40}$ as listed in the section "Text Tool Set Summary" at the end of this chapter

\textbf{C}

\begin{verbatim}
extern pascal void ErrWriteLine(strPtr)

Pointer strPtr;
\end{verbatim}
$1D0C$  **ErrWriteString**

Combines a pointed-to Pascal-type string (first byte of string specifies length) with the output global masks and then writes the string to the error output text device.

### Parameters

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Long—POINTER to Pascal-type string to be written</th>
</tr>
</thead>
<tbody>
<tr>
<td>strPtr</td>
<td>SP</td>
</tr>
</tbody>
</table>

**Stack after call**

| previous contents | SP                                           |

### Errors

Pascal device errors

Any of the errors $0C02–0C40$ as listed in the section "Text Tool Set Summary" at the end of this chapter

C

```c
extern pascal void ErrWriteString(strPtr)
Pointer strPtr;
```
$0E0C  GetErrGlobals

Returns the current values for the error output device global parameters.

Parameters

Stack before call

```
previous contents

wordspace

wordspace

Word—Space for result

Word—Space for result
```

Stack after call

```
previous contents

andMask

orMask

Word—Global AND mask for error output device (low-order byte only)

Word—Global OR mask for error output device (low-order byte only)
```

Errors

None

C

```
extern pascal TxtMaskRec GetErrGlobals()
```
Returns the type of driver installed as the error output device. For BASIC or Pascal device drivers, also returns the slot number; for RAM-based drivers, returns a pointer to the jump table.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
<tr>
<td>longspace</td>
</tr>
</tbody>
</table>

Word—Space for result
Long—Space for result
← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceType</td>
</tr>
<tr>
<td>ptrOrSlot</td>
</tr>
</tbody>
</table>

Word—Error output device type (0 = BASIC, 1 = Pascal, 2 = RAM-based)
Long—Slot number for BASIC or Pascal driver, or POINTER to jump table for RAM-based driver
← SP

Errors

None

C

extern pascal DeviceRec GetErrorDevice()
GetInGlobals

Returns the current values for the input device global parameters.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
<tr>
<td>wordspace</td>
</tr>
</tbody>
</table>

Word—Space for result

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>andMask</td>
</tr>
<tr>
<td>orMask</td>
</tr>
</tbody>
</table>

Word—Global AND mask for input device (low-order byte only)

Word—Global OR mask for input device (low-order byte only)

Errors

None

C

extern pascal TxtMaskRec GetInGlobals()
GetInputDevice

Returns the type of driver installed as the input device. For BASIC or Pascal device drivers, also returns the slot number; for RAM-based drivers, returns a pointer to the jump table.

**Parameters**

<table>
<thead>
<tr>
<th>Stack before call</th>
</tr>
</thead>
<tbody>
<tr>
<td>previous contents</td>
</tr>
<tr>
<td>wordspace</td>
</tr>
<tr>
<td>longspace</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack after call</th>
</tr>
</thead>
<tbody>
<tr>
<td>previous contents</td>
</tr>
<tr>
<td>deviceType</td>
</tr>
<tr>
<td>ptrOrSlot</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Errors**

None

**C**

```c
extern pascal DeviceRec GetInputDevice()
```
$0D0C      GetOutGlobals

Returns the current values for the output device global parameters.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
</tbody>
</table>
| wordspace         |  Word—Space for result  Word—Space for result  ← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>
| andMask           |  Word—Global AND mask for output device (low-order byte only)
| orMask            |  Word—Global OR mask for output device (low-order byte only)  ← SP

Errors

None

C

extern pascal TxtMaskRec GetOutGlobals()
$130C  GetOutputDevice

Returns the type of driver installed as the output device. For BASIC or Pascal device drivers, also returns the slot number; for RAM-based drivers, returns a pointer to the jump table.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Word—Space for result</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td></td>
</tr>
<tr>
<td>longspace</td>
<td>Long—Space for result</td>
</tr>
<tr>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Word—Output device type (0 = BASIC, 1 = Pascal, 2 = RAM-based)</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceType</td>
<td>Long—Slot number for BASIC or Pascal driver, or POINTER to jump</td>
</tr>
<tr>
<td>ptrOrSlot</td>
<td>table for RAM-based driver</td>
</tr>
<tr>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

Errors

None

C

extern pascal DeviceRec GetOutputDevice()
$150C \textbf{InitTextDev} \\
Initializes a specified text device.

\textbf{Parameters}

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceNum</td>
</tr>
</tbody>
</table>

Word—Device to initialize (0 = input, 1 = output, 2 = error output) \leftarrow SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

\leftarrow SP

\textbf{Errors}

$0C01 \hspace{1em} \text{badDevType} \hspace{1em} \text{Illegal device type}

\textbf{C}

extern pascal void InitTextDev(deviceNum)

Word \hspace{1em} deviceNum;
ReadChar

Reads a character obtained from the input text device, combines it with the input global masks, and returns the character on the stack. If echoFlag is set to a value of $0001, the character is also written to the output device.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Word—Space for result</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
<td>echoFlag</td>
</tr>
<tr>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Word—Character (in low-order byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>← SP</td>
</tr>
</tbody>
</table>

Errors

Pascal device errors

Any of the errors $0C02-$0C40 as listed in the section “Text Tool Set Summary” at the end of this chapter

C

```c
extern pascal Word ReadChar(echoFlag)

Word echoFlag;
```

The echo-flag word

The values for echoFlag are illustrated in Figure 23-1.

```
+--------------+
<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved: set to 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

echo

Echo characters to output device = 1
Don't echo characters = 0
```

Figure 23-1

Character echo-flag word
$240C  **ReadLine**

Reads a character string from the input text device, combines it with the input global masks, and writes the string to the memory location starting at *bufferPtr*. The character string is terminated when one of the following conditions is met:

- An End-of-line (EOL) character is received.
- The count of characters received is equal to the maximum line length specified by `maxCount`.

If `echoFlag` is set to a value of $0001, the characters are also written to the output device. The `echoFlag` is illustrated in Figure 23-1 in the section "ReadChar" in this chapter.

The count of characters received is returned on the stack.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wordspace</code></td>
</tr>
<tr>
<td><code>bufferPtr</code></td>
</tr>
<tr>
<td><code>maxCount</code></td>
</tr>
<tr>
<td><code>eolChar</code></td>
</tr>
<tr>
<td><code>echoFlag</code></td>
</tr>
</tbody>
</table>

- **Word**—Space for result
- **Long**—POINTER to start of buffer where characters will be written
- **Word**—Maximum line length
- **Word**—End-of-line character in low byte
- **Word**—Echo or don’t echo characters to output device (see Figure 23-1)

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>charCount</code></td>
</tr>
</tbody>
</table>

- **Word**—Count of characters received

**Errors**

- Pascal device errors

Any of the errors $0C02-$0C40 as listed in the section "Text Tool Set Summary" at the end of this chapter

---

23-30  **Text Tool Set routines**
extern pascal Word ReadLine(bufferPtr, maxCount, eolChar, echoFlag)

Pointer bufferPtr;

Word maxCount;

Word eolChar;

Word echoFlag;
$0B0C  SetErrGlobals

Sets the global parameters for the error output device.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Word—Global AND mask for error output device (low-order byte only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>andMask</td>
<td></td>
</tr>
<tr>
<td>orMask</td>
<td>Word—Global OR mask for error output device (low-order byte only)</td>
</tr>
</tbody>
</table>

Stack after call

| previous contents | ← SP |

Errors

None

C

extern pascal void SetErrGlobals(andMask, orMask)
Word andMask;
Word orMask;

You can also use the following alternate form of the call:

extern pascal void SetErrGlobals(mask)
TxtMaskRec mask;
$110C SetErrorDevice

Sets the error output device to a specified type and location. The routine returns an error if the deviceType specified is greater than 2.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceType</td>
<td>Word—Error output device (0 = BASIC, 1 = Pascal, 2 = RAM based)</td>
</tr>
<tr>
<td>ptrOrSlot</td>
<td>Long—Slot number for BASIC or Pascal driver, or POINTER to jump table for RAM-based driver</td>
</tr>
</tbody>
</table>

Stack after call

| previous contents | ← SP |

Errors

| $0C01 badDevType Illegal device type |

C

extern pascal void SetErrorDevice(deviceType, ptrOrSlot)

Word deviceType;

LongWord ptrOrSlot;

You can also use the following alternate form of the call:

extern pascal void SetErrorDevice(deviceRec)

DeviceRec deviceRec;
$090C  SetInGlobals

Sets the global parameters for the input device.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>andMask</td>
</tr>
<tr>
<td>orMask</td>
</tr>
</tbody>
</table>

Word—Global AND mask for input device (low-order byte only)
Word—Global OR mask for input device (low-order byte only)

Stack after call

| previous contents |

<- SP

Errors

None

C

text pascal void SetInGlobals(andMask, orMask)

Word    andMask;
Word    orMask;

You can also use the following alternate form of the call:

text pascal void SetInGlobals(mask)

TxtMaskRec    mask;
$0F0C  **SetInputDevice**

Sets the input device to a specified type and location. The routine returns an error if the *deviceType* is greater than 2.

**Parameters**

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceType</td>
</tr>
<tr>
<td>ptrOrSlot</td>
</tr>
</tbody>
</table>

- **Word**—Input device type (0 = BASIC, 1 = Pascal, 2 = RAM based)
- **Long**—Slot number for BASIC or Pascal driver, or POINTER to jump table for RAM-based driver

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

Errors

$0C01  badDevType    Illegal device type

C

```c
extern pascal void SetInputDevice(deviceType, ptrOrSlot)

Word    deviceType;
LongWord ptrOrSlot;
```

You can also use the following alternate form of the call:

```c
extern pascal void SetInputDevice(DeviceRec)

DeviceRec  deviceRec;
```

(continued)
Assembly-language examples

**BASIC example**

```
PEA 0000 ; BASIC type
PEA 000 ; in slot 3
PEA 003
  _SetInputDevice
```

**RAM-based example**

```
PEA 002 ; RAM-based type
PEA label/256 ; RAM driver location
PEA label
  _SetInputDevice
```

23-36  Text Tool Set routines
SetOutGlobals

Sets the global parameters for the output device.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>andMask</th>
<th>Word—Global AND mask for output device (low-order byte only)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>orMask</td>
<td>Word—Global OR mask for output device (low-order byte only)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

| previous contents  | ← SP    |

Errors

None

C

extern pascal void SetOutGlobals(andMask, orMask)

Word andMask;

Word orMask;

You can also use the following alternate form of the call:

extern pascal void SetOutGlobals(mask)

TxtMaskRec mask;
$100C  **SetOutputDevice**

Sets the output device to a specified type and location. The routine returns an error if the *deviceType* specified is greater than 2.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>deviceType</th>
<th>Word—Output device type (0 = BASIC, 1 = Pascal, 2 = RAM based)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ptrOrSlot</td>
<td>Long—Slot number for BASIC or Pascal driver, or POINTER to jump table for RAM-based driver</td>
</tr>
<tr>
<td>← SP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Stack after call**

| previous contents | ← SP |
|                  |      |

**Errors**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$0C01</td>
<td>badDevType</td>
<td>Illegal device type</td>
</tr>
</tbody>
</table>

C

```c
extern pascal void SetOutputDevice(deviceType, ptrOrSlot)

Word    deviceType;
LongWord ptrOrSlot;
```

You can also use the following alternate form of the call:

```c
extern pascal void SetOutputDevice(deviceRec)

DeviceRec deviceRec;
```
$170C StatusTextDev

Executes a status call to a specified text device. The routine returns an error if the device is not ready.

Warning
BASIC devices do not support this routine.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>Previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceNum</td>
</tr>
<tr>
<td>requestCode</td>
</tr>
</tbody>
</table>

Word—Device for inquiry (0 = input, 1 = output, 2 = error output)
Word—Request code (in low-order byte) to pass to deviceNum

Stack after call

| Previous contents |

Errors

Pascal device errors
Any of the errors $0C02-$0C40 as listed in the section "Text Tool Set Summary" at the end of this chapter

C

extern pascal void StatusTextDev(deviceNum, requestCode)

Word    deviceNum;
Word    requestCode;

Text Tool Set routines 23-39
**$230C  TextReadBlock**

Reads a block of characters from the input text device, combines them with the input global masks, and writes the block to the memory location at `bufferPtr + offset`.

If `echoFlag` is set to a value of $0001, the characters are also written to the output device. The echoFlag word is illustrated in Figure 23-1 in the section “ReadChar” in this chapter.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bufferPtr</code> Long—POINTER to start of block</td>
</tr>
<tr>
<td><code>offset</code> Word—Offset to start of block to be read</td>
</tr>
<tr>
<td><code>blockSize</code> Word—Size of block to be read</td>
</tr>
<tr>
<td><code>echoFlag</code> Word—Echo or don’t echo characters to output device (see Figure 23-1)</td>
</tr>
</tbody>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

Pascal device errors Any of the errors $0C02–$0C40 as listed in the section “Text Tool Set Summary” at the end of this chapter

**C**

```
extern pascal void TextReadBlock(bufferPtr, offset, blockSize, echoFlag);
```

- Pointer `bufferPtr`;
- Word `offset`;
- Word `blockSize`;
- Word `echoFlag`;

23-40  Text Tool Set routines
$1E0C \textbf{TextWriteBlock}

Combines a specified character string with the output global masks and then writes the string to the output text device. The string is specified by the \texttt{textPtr + offset} parameters, with the length specified by the \texttt{count} parameter.

**Parameters**

**Stack before call**

- \texttt{previous contents}
  - \texttt{textPtr} — Long—POINTER to start of ASCII text
  - \texttt{offset} — Word—Offset within ASCII text
  - \texttt{count} — Word—String length

\text{← SP}

**Stack after call**

- \texttt{previous contents}
  - \text{← SP}

**Errors**

Pascal device errors

Any of the errors \$0C02-$0C40 as listed in the section “Text Tool Set Summary” at the end of this chapter

**C**

```c
extern pascal void TextWriteBlock(textPtr, offset, count);

Pointer textPtr;
Word offset;
Word count;
```
$180C  \textbf{WriteChar}

Combines a specified character with the output global AND mask and OR mask and writes the character to the output text device.

**Parameters**

**Stack before call**

\[
\begin{array}{c|c}
\text{previous contents} & \text{theChar} \rightarrow \text{Word} - \text{Character to write (in low-order byte)} \\
\end{array}
\]

**Stack after call**

\[
\begin{array}{c|c}
\text{previous contents} & \leftarrow \text{SP} \\
\end{array}
\]

**Errors**

Pascal device errors  Any of the errors $0C02$–$0C40$ as listed in the section “Text Tool Set Summary” at the end of this chapter

\begin{verbatim}
C
extern pascal void WriteChar(theChar)
Word theChar;
\end{verbatim}
WriteCString

Combines a pointed-to C-type string (string terminates with $00) with the output global masks and then writes the string to the output text device.

Parameters

Stack before call

\[
\begin{array}{l}
\text{previous contents} \\
\hline
- cStrPtr \\
\hline
\end{array}
\]

\textbf{Long}—POINTER to C-type string to be written

\textbf{SP}

Stack after call

\[
\begin{array}{l}
\text{previous contents} \\
\hline
\end{array}
\]

\textbf{SP}

Errors

Pascal device errors

Any of the errors $0C02-$0C40 as listed in the section "Text Tool Set Summary" at the end of this chapter

C

\text{extern} \text{ pascal} \text{ void} \text{WriteCString(cStrPtr)}

\text{Pointer} \quad \text{cStrPtr;}

Text Tool Set routines 23-43
$1A0C \textbf{WriteLine} \\

Combines a pointed-to Pascal-type string (first byte of string specifies length) with the output global masks and then writes the string to the output text device. For BASIC and RAM-based drivers, the routine concatenates a carriage return to the string. For Pascal drivers, the routine concatenates a carriage return and a line feed to the string.

\textbf{Parameters}

\textbf{Stack before call}

\begin{verbatim}
previous contents
-- strPtr
← SP
\end{verbatim}

\textbf{Stack after call}

\begin{verbatim}
previous contents
← SP
\end{verbatim}

\textbf{Errors}

Pascal device errors \hspace{2cm} Any of the errors $0C02$-$0C40$ as listed in the section "Text Tool Set Summary" at the end of this chapter

\textbf{C}

\begin{verbatim}
extern pascal void WriteLine(strPtr)

Pointer strPtr;
\end{verbatim}
$1C0C WriteString

Combines a pointed-to Pascal-type string (first byte of string specifies length) with the output global masks and then writes the string to the output text device.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>strPtr</td>
</tr>
</tbody>
</table>

Long--POINTER to Pascal-type string to be written

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

← SP

Errors

Pascal device errors

Any of the errors $0C02–$0C40 as listed in the section "Text Tool Set Summary" at the end of this chapter

C

extern pascal void WriteString(strPtr)

Pointer strPtr;
Text Tool Set summary

This section briefly summarizes the constants and tool set errors contained in the Text Tool Set. There are no predefined data structures for the Text Tool Set.

Important

These definitions are provided in the appropriate interface file.

Table 23-4
Text Tool Set constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Echo flag values</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>noEcho</td>
<td>$0000</td>
<td>Don’t echo characters to output device</td>
</tr>
<tr>
<td>echo</td>
<td>$0001</td>
<td>Echo characters to output device</td>
</tr>
<tr>
<td><strong>Device numbers</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>input</td>
<td>$0000</td>
<td>Input device</td>
</tr>
<tr>
<td>output</td>
<td>$0001</td>
<td>Output device</td>
</tr>
<tr>
<td>errorOutput</td>
<td>$0002</td>
<td>Error output device</td>
</tr>
<tr>
<td><strong>Device types</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>basicType</td>
<td>$0000</td>
<td>Basic device type</td>
</tr>
<tr>
<td>pascalType</td>
<td>$0001</td>
<td>Pascal device type</td>
</tr>
<tr>
<td>ramBased</td>
<td>$0002</td>
<td>RAM-based device type</td>
</tr>
<tr>
<td>Code</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>$0C01</td>
<td>badDevType</td>
<td>Illegal device type</td>
</tr>
<tr>
<td>$0C02</td>
<td>badDevNum</td>
<td>Illegal device number</td>
</tr>
<tr>
<td>$0C03</td>
<td>badMode</td>
<td>Illegal operation</td>
</tr>
<tr>
<td>$0C04</td>
<td>undefHW</td>
<td>Undefined hardware error</td>
</tr>
<tr>
<td>$0C05</td>
<td>lostDev</td>
<td>Lost device: device no longer on-line</td>
</tr>
<tr>
<td>$0C06</td>
<td>lostFile</td>
<td>File no longer in diskette directory</td>
</tr>
<tr>
<td>$0C07</td>
<td>badTitle</td>
<td>Illegal filename</td>
</tr>
<tr>
<td>$0C08</td>
<td>noRoom</td>
<td>Insufficient space on specified diskette</td>
</tr>
<tr>
<td>$0C09</td>
<td>noDevice</td>
<td>Specified volume not on-line</td>
</tr>
<tr>
<td>$0C0A</td>
<td>noFile</td>
<td>Specified file not in directory of specified volume</td>
</tr>
<tr>
<td>$0C0B</td>
<td>dupFile</td>
<td>Duplicate file, attempt to rewrite a file when a file of that name already exists</td>
</tr>
<tr>
<td>$0C0C</td>
<td>notClosed</td>
<td>Attempt to open file that is already open</td>
</tr>
<tr>
<td>$0C0D</td>
<td>notOpen</td>
<td>Attempt to access a closed file</td>
</tr>
<tr>
<td>$0C0E</td>
<td>badFormat</td>
<td>Error in reading real or integer number</td>
</tr>
<tr>
<td>$0C0F</td>
<td>ringBuffOFlo</td>
<td>Ring buffer overflow: characters arriving faster than the input buffer can accept them</td>
</tr>
<tr>
<td>$0C10</td>
<td>writeProtected</td>
<td>Specified diskette is write-protected</td>
</tr>
<tr>
<td>$0C40</td>
<td>dev2rr</td>
<td>Device error: device failed to complete a read or write correctly</td>
</tr>
</tbody>
</table>

Note: With the exception of badDevType, the errors should occur only for Pascal devices.
Chapter 24

Tool Locator

The Tool Locator is the tool set that allows your application to use tool sets without knowing whether the tool sets are in RAM or ROM.

In this chapter, we provide the information you need to use the existing tool sets. If you are writing your own tool set, however, you'll need to understand more about how the Tool Locator works. We have provided most of that information in Appendix A, "Writing Your Own Tool Set," as a special subject; please refer to that appendix for more information.

A preview of the Tool Locator routines

To introduce you to the capabilities of the Tool Locator, all Tool Locator routines are grouped by function and briefly described in Table 24-1. These routines are described in detail later in this chapter, where they are separated into housekeeping routines (discussed in routine number order) and the rest of the Tool Locator routines (discussed in alphabetical order).
<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Housekeeping routines</strong></td>
<td></td>
</tr>
<tr>
<td>TLBootInit</td>
<td>Initializes the Tool Locator and all other ROM-based tool sets—must not be called by an application</td>
</tr>
<tr>
<td>TLStartUp</td>
<td>Starts up the Tool Locator for use by an application</td>
</tr>
<tr>
<td>TLShutDown</td>
<td>Shuts down the Tool Locator when an application quits</td>
</tr>
<tr>
<td>TLVersion</td>
<td>Returns the version number of the Tool Locator</td>
</tr>
<tr>
<td>TLReset</td>
<td>Resets the Tool Locator and all other ROM-based tool sets when the system is reset—must not be called by an application</td>
</tr>
<tr>
<td>TLStatus</td>
<td>Indicates whether the Tool Locator is active</td>
</tr>
<tr>
<td><strong>Tool locating routines</strong></td>
<td></td>
</tr>
<tr>
<td>LoadTools</td>
<td>Ensures that specified system tool sets are available and have specified minimum version numbers</td>
</tr>
<tr>
<td>LoadOneTool</td>
<td>Ensures that a specified system tool set is available and has a specified minimum version number</td>
</tr>
<tr>
<td>UnloadOneTool</td>
<td>Unloads a specified tool set from memory</td>
</tr>
<tr>
<td>GetFuncPtr</td>
<td>Returns an entry in the function pointer table for a specified function in a specified tool set</td>
</tr>
<tr>
<td>GetTSPtr</td>
<td>Returns the pointer to the function pointer table of a specified tool set</td>
</tr>
<tr>
<td>SetTSPtr</td>
<td>Installs the pointer to a function pointer table in the appropriate tool pointer table</td>
</tr>
<tr>
<td>GetWAP</td>
<td>Gets the pointer to the work area for a specified tool set</td>
</tr>
<tr>
<td>SetWAP</td>
<td>Sets the pointer to the work area for a specified tool set</td>
</tr>
<tr>
<td>TLMountVolume</td>
<td>Displays on the Super Hi-Res display a simulated dialog box that your application can use to ask the user to mount a volume</td>
</tr>
<tr>
<td>TLTextMountVolume</td>
<td>Displays on the 40-column text screen a simulated dialog box that your application can use to ask the user to mount a volume</td>
</tr>
<tr>
<td>SaveTextState</td>
<td>Saves the state of the text screen and forces the hardware to display the text screen regardless of the display mode in use</td>
</tr>
<tr>
<td>RestoreTextState</td>
<td>Restores the state of the text screen from a specified handle and disposes of the handle</td>
</tr>
<tr>
<td>MessageCenter</td>
<td>Allows applications to communicate with each other</td>
</tr>
</tbody>
</table>
Using the Tool Locator

This section discusses how some of the Tool Locator routines fit into the general flow of an application and gives you an idea of which routines you'll need to use under normal circumstances. Each routine is described in detail later in this chapter.

The Tool Locator does not depend on the presence of any of the other tool sets; rather, all other tool sets depend on the Tool Locator.

The first Tool Locator call your application must make is TLStartUp. TLStartUp starts the Tool Locator mechanism that allows the other tool sets to be found. Conversely, when you quit your application, you must make the TLShutDown call just before the application quits.

Your application ensures that the tool sets it needs are available—and that they meet specified minimum version requirements—by making a LoadTools call. If, for example, you wanted to use the LineEdit Tool Set routine LETextBox2, which is available only in versions 2.0 and later of the tool set, you would specify 2.0 as the minimum version for the LineEdit Tool Set in the LoadTools call. The LoadTools routine loads all the tool sets that meet the minimum version requirements of those specified in the routine, thus making them available for the life of the application.

RAM-based tool sets are loaded only when the application requests they be loaded. Thus, if you want to load a tool set just before your application uses it, you can use the LoadOneTool routine (this routine also checks for the minimum tool set version). When you are finished with the tool set, you can unload it by using the UnloadOneTool routine.

The routines TLMountVolume and TLTextMountVolume can help if the boot disk containing the specified tool set (in the TOOLS subdirectory of the SYSTEM directory) is not available when the tool set is requested. TLMountVolume provides a mechanism your application can use to display a prompting message and OK and Cancel buttons on the Super Hi-Res display. TLTextMountVolume performs the same function (with a message only, no buttons) for the 40-column text display.

You can use the SaveTextState and RestoreTextState routines to save and restore the state of the text screen. You can use the MessageCenter call to communicate with other applications.

Normally, your application won't need to use the GetWap, SetWap, GetTSPtr, SetTSPtr, and GetFuncPtr calls; those calls are used only when you are writing your own tool set (see Appendix A, "Writing Your Own Tool Set," for more details).
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLBootInit</td>
<td>Initializes the Tool Locator and all other ROM-based tool sets; called only by the system startup firmware.</td>
</tr>
<tr>
<td><strong>Warning</strong></td>
<td>An application must never make this call.</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td>The stack is not affected by this call. There are no input or output parameters.</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>Call must not be made by an application.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLStartUp</td>
<td>Starts up the Tool Locator for use by an application.</td>
</tr>
<tr>
<td><strong>Important</strong></td>
<td>Your application must make this call before it makes any other tool calls.</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td>The stack is not affected by this call. There are no input or output parameters.</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>extern pascal void TLStartUp()</td>
</tr>
</tbody>
</table>

24-4 Tool Locator housekeeping routines
$0301 TLShutDown
Shuts down the Tool Locator when an application quits.

Important
Your application must make this call just before it quits.

Parameters
The stack is not affected by this call. There are no input or output parameters.

Errors
None

C
extern pascal void TLShutDown()

$0401 TLVersion
Returns the version number of the Tool Locator.

Parameters

Stack before call

```
previous contents
wordspace
```
Word—Space for result
← SP

Stack after call

```
previous contents
versionInfo
```
Word—Version number of Tool Locator
← SP

Errors
None

C
extern pascal Word TLVersion()
**TLReset**

Resets the Tool Locator and all other ROM-based tool sets when the system is reset; called only by the system firmware.

---

**Warning**

An application must never make this call.

---

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

**Errors**

None

**C**

Call must not be made by an application.

---

**TLStatus**

Indicates whether the Tool Locator is active.

---

**Parameters**

Stack before call

| previous contents | wordspace | Word—Space for result
|-------------------|-----------|-----------------------|

Stack after call

| previous contents | activeFlag | Word—BOOLEAN; TRUE (Tool Locator is always active)
|-------------------|------------|-----------------------|

**Errors**

None

**C**

extern pascal Boolean TLStatus()

---

24-6 Tool Locator housekeeping routines
$0B01  **GetFuncPtr**  
Returns an entry in the function pointer table for a specified function in a specified tool set.

*Note:* The word containing *funcNum* and *tsNum* is the same as the number the Tool Locator normally uses to locate the routine. For example, the appropriate number for GetFuncPtr is $0B01.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
<tr>
<td>userOrSystem</td>
</tr>
<tr>
<td>funcNum</td>
</tr>
</tbody>
</table>

| Long—Space for result |
| Word—$0000 = system tool set; $8000 = user tool set |
| Byte—Function number | Byte—Tool set number |

← SP

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>fptEntry</td>
</tr>
</tbody>
</table>

| Long—Entry in function pointer table |

← SP

**Errors**

| $0001 | toolNotFoundErr | Specified tool set not found |
| $0002 | funcNotFoundErr | Specified routine not found |

**C**

```c
extern pascal Pointer GetFuncPtr(userOrSystem, funcTSNum)

Word  userOrSystem;
Word  funcTSNum;
```
GetTSPtr

Returns the pointer to the function pointer table of a specified tool set.

*Note:* This call is normally used only if you are writing your own tool set. See Appendix A, "Writing Your Own Tool Set," for more information.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>--- longspace ---</td>
</tr>
<tr>
<td>--- userOrSystem ---</td>
</tr>
<tr>
<td>--- tsNum ---</td>
</tr>
</tbody>
</table>

- **Long**—Space for result
- **Word**—$0000 = system tool set; $8000 = user tool set
- **Word**—Tool set number of tool set whose pointer is to be returned

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>--- fptPtr ---</td>
</tr>
</tbody>
</table>

- **Long**—POINTER to function pointer table of tool set

**Errors**

| $0001 toolNotFoundErr |

Specified tool set not found

**C**

```c
extern pascal Pointer GetTSPtr(userOrSystem, tsNum)
```

```c
Word userOrSystem;
Word tsNum;
```
$0C01 \textbf{GetWAP}

Gets the pointer to the work area for a specified tool set.

\textbf{Note}: This call is normally used only if you are writing your own tool set. See Appendix A, "Writing Your Own Tool Set," for more information.

\section*{Parameters}

\subsection*{Stack before call}

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{longspace}</td>
</tr>
<tr>
<td>\textbf{userOrSystem}</td>
</tr>
<tr>
<td>\textbf{tsNum}</td>
</tr>
</tbody>
</table>

- \textbf{Long}—Space for result
- \textbf{Word}—$0000 = \text{system tool set}; 8000 = \text{user tool set}
- \textbf{Word}—Number of tool set

\subsection*{Stack after call}

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{waptPtr}</td>
</tr>
</tbody>
</table>

- \textbf{Long}—POINTER to work area of tool set

\subsection*{Errors}

- $0001$  \text{toolNotFoundErr}  Specified tool set not found

\subsection*{C}

\begin{verbatim}
extern pascal Pointer GetWAP(userOrSystem, tsNum)
\end{verbatim}

\begin{verbatim}
Word userOrSystem;
Word tsNum;
\end{verbatim}
$0F01 LoadOneTool

Ensures that a specified system tool set is available (loading it from disk if necessary) and has at least a specified minimum version number. If the minimum version of the tool is not available in the TOOLS subdirectory of the SYSTEM directory in the boot volume, an error occurs.

 tô Note: See the section "LoadTools" in this chapter for a list of tool set numbers.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>toolNumber</td>
</tr>
<tr>
<td>minVersion</td>
</tr>
</tbody>
</table>

Word—INTEGER; tool set number of tool set to load
Word—Minimum version number of tool set needed

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

Errors

$0001 toolNotFoundErr Specified tool set not found
$0010 toolVersionErr Specified minimum version not found
System Loader errors Returned unchanged
ProDOS errors Returned unchanged

C

extern pascal void LoadOneTool(toolNumber, minVersion)
Word toolNumber;
Word minVersion;

24-10 Tool Locator routines
LoadTools

Ensures that specified system tool sets are available and checks that the tool sets have at least a specified **minimum version number**. The call needs, as input, a pointer to a tool table. The tool table lists the total number of tool sets, the number of each tool set needed, and the minimum acceptable version number of each tool set.

The structure of the tool table is illustrated in Figure 24-1. The numbers of the tool sets are given in Table 24-2.

**Important**

RAM-based tools are loaded from the TOOLS subdirectory of the SYSTEM directory. Each tool set is a load file of type SBA and is named after its decimal tool set number (Tool 23 is in a file named TOOL023, and so on).

**Parameters**

**Stack before call**

```
previous contents
```

```
-- toolTablePtr --

Long—POINTER to table of tool set numbers
```

```
<- SP
```

**Stack after call**

```
previous contents
```

```
<- SP
```

**Errors**

- `$0001` toolNotFoundErr Specified tool set not found
- `$0010` toolVersionErr Specified minimum version not found
- System Loader errors Returned unchanged
- ProDOS errors Returned unchanged

```
extern pascal void LoadTools(toolTablePtr)

Pointer toolTablePtr;
```

(continued)
Tool table

The form the tool table must take is shown in Figure 24-1.

<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Count</th>
<th>Minimum version of tool set</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01</td>
<td></td>
<td>$0103</td>
</tr>
<tr>
<td>$02</td>
<td></td>
<td>$0201</td>
</tr>
<tr>
<td>$03</td>
<td></td>
<td>$0100</td>
</tr>
</tbody>
</table>

Thus, in assembly language the table looks like this:

```assembly
dc i'NumToolsRequired'
dc i'ToolNum1, MinVersion'
dc i'ToolNum2, MinVersion'
...
dc i'ToolNumN, MinVersion'
```

The format of the `MinVersion` word is as follows:

High-order byte = Major version  
Low-order byte = Minor version  

Thus, a value of $0201 means a major version of 2 and a minor version of 1, and is commonly referred to as "Version 2.1."
Tool set numbers

The tool set numbers of all of the tool sets are listed in Table 24-2.

<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01</td>
<td>#01 Tool Locator</td>
</tr>
<tr>
<td>$02</td>
<td>#02 Memory Manager</td>
</tr>
<tr>
<td>$03</td>
<td>#03 Miscellaneous Tool Set</td>
</tr>
<tr>
<td>$04</td>
<td>#04 QuickDraw II</td>
</tr>
<tr>
<td>$05</td>
<td>#05 Desk Manager</td>
</tr>
<tr>
<td>$06</td>
<td>#06 Event Manager</td>
</tr>
<tr>
<td>$07</td>
<td>#07 Scheduler</td>
</tr>
<tr>
<td>$08</td>
<td>#08 Sound Tool Set</td>
</tr>
<tr>
<td>$09</td>
<td>#09 Apple Desktop Bus Tool Set</td>
</tr>
<tr>
<td>$0A</td>
<td>#10 SANE Tool Set</td>
</tr>
<tr>
<td>$0B</td>
<td>#11 Integer Math Tool Set</td>
</tr>
<tr>
<td>$0C</td>
<td>#12 Text Tool Set</td>
</tr>
<tr>
<td>$0D</td>
<td>#13 Reserved for internal use</td>
</tr>
<tr>
<td>$0E</td>
<td>#14 Window Manager</td>
</tr>
<tr>
<td>$0F</td>
<td>#15 Menu Manager</td>
</tr>
<tr>
<td>$10</td>
<td>#16 Control Manager</td>
</tr>
<tr>
<td>$11</td>
<td>#17 System Loader</td>
</tr>
<tr>
<td>$12</td>
<td>#18 QuickDraw II Auxiliary</td>
</tr>
<tr>
<td>$13</td>
<td>#19 Print Manager</td>
</tr>
<tr>
<td>$14</td>
<td>#20 LineEdit Tool Set</td>
</tr>
<tr>
<td>$15</td>
<td>#21 Dialog Manager</td>
</tr>
<tr>
<td>$16</td>
<td>#22 Scrap Manager</td>
</tr>
<tr>
<td>$17</td>
<td>#23 Standard File Operations Tool Set</td>
</tr>
<tr>
<td>$18</td>
<td>#24 Not available</td>
</tr>
<tr>
<td>$19</td>
<td>#25 Note Synthesizer</td>
</tr>
<tr>
<td>$1A</td>
<td>#26 Note Sequencer</td>
</tr>
<tr>
<td>$1B</td>
<td>#27 Font Manager</td>
</tr>
<tr>
<td>$1C</td>
<td>#28 List Manager</td>
</tr>
</tbody>
</table>

Tool Locator routines 24-13
$1501 \textbf{MessageCenter}

Allows applications to communicate with each other.

\textbf{Important}

This routine is available only in Version 2.1 or later of the Tool Locator.

The message types are administered by Developer Technical Support at Apple Computer.

\textbf{Parameters}

\textbf{Stack before call}

\begin{tabular}{|c|c|}
\hline
\multicolumn{2}{|c|}{\textit{previous contents}} \\
\hline
\textit{action} & \textbf{Word}—1 = add, 2 = get, 3 = delete (see Table 24-3) \\
\hline
\textit{type} & \textbf{Word}—Message ID number \\
\hline
\textit{messageHandle} & \textbf{Long}—HANDLE to message \\
\hline
\end{tabular}

\textit{← SP}

\textbf{Stack after call}

\begin{tabular}{|c|}
\hline
\multicolumn{1}{|c|}{\textit{previous contents}} \\
\hline
\textit{← SP} \\
\end{tabular}

\textbf{Errors}

$0111 \text{ messNotFoundErr} \quad \text{Specified message not found}

\textbf{C}

\begin{verbatim}
extern pascal void MessageCenter(action, type, messageHandle) 
Word action; 
Word type; 
Handle messageHandle;
\end{verbatim}

24-14 \tool Locator routines
More about messages

The action codes the routine will recognize are shown in Table 24-3.

Table 24-3
Message Center action codes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>addMessage</td>
<td>Adds the specified message to the message center data. The type is the ID of the message being added; any message already in the message center with this type is deleted. This message does not alter the messageHandle.</td>
</tr>
<tr>
<td>2</td>
<td>getMessage</td>
<td>Returns the specified message from the message center. If there is no message with the specified ID, an error occurs and the messageHandle is not altered. The messageHandle block can be of any size; the routine will resize the block to fit the data being returned.</td>
</tr>
<tr>
<td>3</td>
<td>deleteMessage</td>
<td>Deletes a specified message from the message center. If the message does not appear in the message center, an error occurs. The messageHandle is not used for this call, and any value may be passed.</td>
</tr>
</tbody>
</table>

The actual data in the message must begin with six bytes of reserved space, as follows:

- **MessNext**: LONG Handle of next message
- **MessType**: WORD Type of this message
- **MessData**: block Message-specific data

The only message defined at the time of publication is type 1. That message type is used to pass file information from one application to another. For example, a Finder-type application needs to tell other applications what documents it should open or print. In this case, the message data is set up as follows:

- **MessNext**: LONG
- **MessType**: WORD 1; file info type
- **MessData**: block 0 = open the following files
- 1 = print the following files
- str 'name1'
- str 'full pathname 2'
- ...
- BYTE 0

Tool Locator routines 24-15
$1401 \textbf{RestoreTextState}

Restores the state of the text screen from a specified handle and disposes of the handle. The state of the text screen can be saved with a SaveTextState call; see the section "SaveTextState" in this chapter.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>(\textbf{Long} - \text{HANDLE} ) to state record to restore</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\textbf{stateHandle} )</td>
<td>(\leftarrow \text{SP} )</td>
</tr>
</tbody>
</table>

**Stack after call**

| previous contents | \(\leftarrow \text{SP} \)                                      |

**Errors**

None

**C**

```
extern pascal void RestoreTextState(stateHandle)
Handle stateHandle;
```
$1301 \textbf{SaveTextState} \\
Saves the state of the text screen and forces the hardware to display the text screen regardless of the display mode in use. The routine does not initialize the Text Tool Set.

\begin{itemize}
\item \textbf{Important} \hspace{1cm} This routine is available only in Version 2.1 or later of the Tool Locator.
\end{itemize}

\textbf{Parameters}

\begin{itemize}
\item \textbf{Stack before call}
\end{itemize}

\begin{itemize}
\item \textit{previous contents}
\item -- \hspace{1cm} \text{Long}--Space for result
\end{itemize}

\begin{itemize}
\item \text{Long}\hspace{1cm} \text{Long-Space for result}
\item \text{previous contents}
\item -- \hspace{1cm} \text{Long-HANDLE to state record (see Table 24-4)}
\item \text{stateHandle}
\item \text{SP}
\end{itemize}

\begin{itemize}
\item \textbf{Errors} \hspace{1cm} None
\end{itemize}

\begin{itemize}
\item \textbf{C}
\end{itemize}

\begin{itemize}
\item extern pascal Handle SaveTextState()
\end{itemize}

(continued)
The SaveTextState record

The structure of the record in which the SaveTextState information is saved is private. At the time of publication, the various entities saved included those shown in Table 24-4.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StateReg</td>
<td>Language card information</td>
</tr>
<tr>
<td>80Col</td>
<td>40 or 80 columns</td>
</tr>
<tr>
<td>NewVideo</td>
<td>New video register</td>
</tr>
<tr>
<td>Text</td>
<td>Text state soft switch</td>
</tr>
<tr>
<td>Mix</td>
<td>Split mode soft switch</td>
</tr>
<tr>
<td>Page2</td>
<td>Page 1 or page 2</td>
</tr>
<tr>
<td>HiRes</td>
<td>HiRes or LoRes</td>
</tr>
<tr>
<td>AltCharSet</td>
<td>Character set</td>
</tr>
<tr>
<td>80Vid</td>
<td>State of 80 store</td>
</tr>
<tr>
<td>Cursor</td>
<td>Internal cursor variable</td>
</tr>
<tr>
<td>c3ROM</td>
<td>State of the slot 3 firmware</td>
</tr>
<tr>
<td>OutputDevice</td>
<td>Text tools output device</td>
</tr>
<tr>
<td>OutGlobals</td>
<td>Text tools output globals</td>
</tr>
<tr>
<td>InputDevice</td>
<td>Text tools input device</td>
</tr>
<tr>
<td>InGlobals</td>
<td>Text tools input globals</td>
</tr>
<tr>
<td>Screen Memory in bank $0</td>
<td></td>
</tr>
<tr>
<td>Screen Memory in bank $1</td>
<td></td>
</tr>
<tr>
<td>Screen Memory in bank $E0</td>
<td></td>
</tr>
<tr>
<td>Screen Memory in bank $E1</td>
<td></td>
</tr>
<tr>
<td>Screen holes for Slot 3</td>
<td></td>
</tr>
</tbody>
</table>

Table 24-4

The various entities saved included those shown in Table 24-4.
SetTSPtr

Installs the pointer to a function pointer table in the appropriate tool pointer table (TPT).

Note: This call is normally used only if you are writing your own tool set. See Appendix A, “Writing Your Own Tool Set,” for more information.

If the TPT is not yet in RAM, the routine copies it to RAM. (Memory for the TPT is obtained from the Memory Manager.) If there is not enough room in the TPT for the new entry, the TPT is moved to a bigger chunk of memory. Likewise, the Work Area Pointer Table (WAPT) is expanded, if necessary (memory for the expansions is obtained from the Memory Manager).

If the new pointer table has any 0 entries, old entries are moved from the old pointer table to the new pointer table. This feature makes it possible, for example, to patch just a portion of a tool set rather than replacing the tool set entirely.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>userOrSystem</td>
</tr>
<tr>
<td>tsNum</td>
</tr>
<tr>
<td>fptPtr</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Errors

$0001  toolNotFoundErr  Specified tool set not found

C

extern pascal void SetTSPtr(userOrSystem, tsNum, fptPtr)

Word  userOrSystem;

Word  tsNum;

Pointer  fptPtr;
$0D01 SetWAP

Sets the pointer to the work area for a specified tool set.

- **Note:** This call is normally used only if you are writing your own tool set. See Appendix A, “Writing Your Own Tool Set,” for more information.

### Parameters

#### Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>userOrSystem</td>
</tr>
<tr>
<td>tsNum</td>
</tr>
<tr>
<td>waptPtr</td>
</tr>
</tbody>
</table>

- **Word**—$0000 = system tool set; $8000 = user tool set
- **Word**—Tool set number of tool set
- **Long**—POINTER to work area of tool set

#### Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

← SP

### Errors

<table>
<thead>
<tr>
<th>$0001 toolNotFoundErr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specified tool set not found</td>
</tr>
</tbody>
</table>

### C

```c
extern pascal void SetWAP(userOrSystem, tsNum, waptPtr)
```

```c
Word  userOrSystem;
Word  tsNum;
Pointer  waptPtr;
```
**$1101**

**TLMountVolume**

Displays on the Super Hi-Res display a simulated dialog box that your application can use to ask the user to mount a volume. The routine also displays two buttons.

*Note:* The box is not really a dialog box because it is not under the control of the Dialog Manager (a RAM-based tool set that might not be active).

The text displayed must be supplied by the application (and can therefore be easily translated into other languages). The contents of the screen under the box are saved before the box is drawn and are restored after the user responds. The button may be chosen by the user either by clicking on it with the mouse button or by pressing the Return key for button 1 or the Esc (escape) key for button 2.

---

**Important**

Your application must make sure that the text passed in the strings fits in the area provided.

---

### Parameters

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><em>wordspace</em></td>
<td><strong>Word</strong>—Space for result</td>
</tr>
<tr>
<td><em>whereX</em></td>
<td><strong>Word</strong>—INTEGER; upper left X coordinate for box</td>
</tr>
<tr>
<td><em>whereY</em></td>
<td><strong>Word</strong>—INTEGER; upper left Y coordinate for box</td>
</tr>
<tr>
<td><em>line1Ptr</em></td>
<td><strong>Long</strong>—POINTER to Pascal-type string to appear at top of box</td>
</tr>
<tr>
<td><em>line2Ptr</em></td>
<td><strong>Long</strong>—POINTER to Pascal-type string to appear just below line 2</td>
</tr>
<tr>
<td><em>but1Ptr</em></td>
<td><strong>Long</strong>—POINTER to Pascal-type string to appear inside button 1</td>
</tr>
<tr>
<td><em>but2Ptr</em></td>
<td><strong>Long</strong>—POINTER to Pascal-type string to appear inside button 2</td>
</tr>
</tbody>
</table>

← **SP**

---

Tool Locator routines 24-21
Stack after call

| previous contents | whichButton | Word—Button number chosen; Return = 1, Esc = 2

← SP

Errors

None

C

extern pascal Word TLMountVolume(whereX, whereY, line1Ptr, line2Ptr, but1Ptr, but2Ptr)
Integer whereX;
Integer whereY;
Pointer line1Ptr;
Pointer line2Ptr;
Pointer but1Ptr;
Pointer but2Ptr;

You can also use the following alternate form of the call:

extern pascal Word TLMountVolume(where, line1Ptr, line2Ptr, but1Ptr, but2Ptr)
Point where;
Pointer line1Ptr;
Pointer line2Ptr;
Pointer but1Ptr;
Pointer but2Ptr;
Displays on the 40-column text screen a simulated dialog box that your application can use to ask the user to mount a volume. The routine also displays two buttons.

*Note:* The box is not really a dialog box because it is not under the control of the Dialog Manager (a RAM-based tool set that might not be active).

The text displayed must be supplied by the application (and can therefore be easily translated into other languages). The contents of the screen under the box are saved before the box is drawn and restored after the user responds. The button is chosen by the user pressing the Return key for button 1 or the Esc (escape) key for button 2.

---

**Important**

Your application must make sure that the text passed in the strings fits in the area provided.

---

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wordspace</code></td>
</tr>
<tr>
<td><code>line1Ptr</code></td>
</tr>
<tr>
<td><code>line2Ptr</code></td>
</tr>
<tr>
<td><code>button1Ptr</code></td>
</tr>
<tr>
<td><code>button2Ptr</code></td>
</tr>
</tbody>
</table>

- **Word**—Space for result
- **Long**—POINTER to Pascal-type string to appear at top of box
- **Long**—POINTER to Pascal-type string to appear just below line 2
- **Long**—POINTER to Pascal-type string to appear inside button 1
- **Long**—POINTER to Pascal-type string to appear inside button 2

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>whichButton</code></td>
</tr>
</tbody>
</table>

- **Word**—Button number chosen; Return = 1, Esc = 2

---

Tool Locator routines 24-23
Errors
None

C

extern pascal Word TLTextMountVolume(line1Ptr, line2Ptr, but1Ptr, but2Ptr)

Pointer line1Ptr;
Pointer line2Ptr;
Pointer but1Ptr;
Pointer but2Ptr;

Tool Locator routines
$1001  **UnloadOneTool**

Unloads a specified tool set from memory. See Table 24-2 in the section “LoadTools” in this chapter for the tool set numbers.

The tool set is left in a restartable state; that is, the purge level of all of the tool set’s memory blocks is set to 3 so that the tool set may be restarted quickly. See Chapter 12, “Memory Manager,” in Volume 1 for more information on purge levels.

**Parameters**

**Stack before call**

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>toolNumber</td>
</tr>
</tbody>
</table>
```

Word—INTEGER; tool set number of tool set to unload

```
<- SP
```

**Stack after call**

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>
```

```
<- SP
```

**Errors**

$0001  toolNotFoundErr  Specified tool set not found

**C**

```c
extern pascal void UnloadOneTool(toolNumber)
Word    toolNumber;
```
Tool Locator summary

This section briefly summarizes the constants and tool set error codes contained in the Tool Locator. There are no predefined data structures for the Tool Locator.

Important

These definitions are provided in the appropriate interface file.

Table 24-5
Tool Locator constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message center action codes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addMessage</td>
<td>$0001</td>
<td>Add message</td>
</tr>
<tr>
<td>getMessage</td>
<td>$0002</td>
<td>Get message</td>
</tr>
<tr>
<td>deleteMessage</td>
<td>$0003</td>
<td>Delete message</td>
</tr>
<tr>
<td>Message center type values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fileInfoType</td>
<td>$0001</td>
<td></td>
</tr>
<tr>
<td>TLMountVolume button values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mvReturn</td>
<td>$0001</td>
<td>Equivalent of dialog OK button</td>
</tr>
<tr>
<td>mvEscape</td>
<td>$0002</td>
<td>Equivalent of dialog Cancel button</td>
</tr>
<tr>
<td>Tool set specification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sysTool</td>
<td>$0000</td>
<td>System tool set</td>
</tr>
<tr>
<td>userTool</td>
<td>$8000</td>
<td>User tool set</td>
</tr>
</tbody>
</table>

Table 24-6
Tool Locator error codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0001</td>
<td>toolNotFoundErr</td>
<td>Specified tool set not found</td>
</tr>
<tr>
<td>$0002</td>
<td>funcNotFoundErr</td>
<td>Specified routine not found</td>
</tr>
<tr>
<td>$0110</td>
<td>toolVersionErr</td>
<td>Specified minimum version not found</td>
</tr>
<tr>
<td>$0111</td>
<td>messNotFoundErr</td>
<td>Specified message not found</td>
</tr>
</tbody>
</table>
The **Window Manager** is a tool set for dealing with windows on the Apple IIGS screen. The screen represents a working surface or **desktop**; graphic objects appear on the desktop and can be manipulated with a mouse. A **window** is an object on the desktop in which information, such as a document or a picture, is presented. Windows can be any size or shape, and there can be one or many of them, depending on the application.

Windows allow the application to control more information than the screen can display at one time. The name **window** is used because the user sees through the window into a larger area, as illustrated in Figure 25-1.
A preview of the Window Manager routines

To introduce you to the capabilities of the Window Manager, all Window Manager routines are grouped by function and briefly described in Table 25-1. These routines are described in detail later in this chapter, where they are separated into housekeeping routines (discussed in routine number order) and the rest of the Window Manager routines (discussed in alphabetical order).

Table 25-1
Window Manager routines and their functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Housekeeping routines</strong></td>
<td></td>
</tr>
<tr>
<td>WindBootInit</td>
<td>Initializes the Window Manager; called only by the Tool Locator—must not be called by an application</td>
</tr>
<tr>
<td>WindStartUp</td>
<td>Starts up the Window Manager for use by an application</td>
</tr>
<tr>
<td>WindShutDown</td>
<td>Shuts down the Window Manager when an application quits</td>
</tr>
<tr>
<td>WindVersion</td>
<td>Returns the version number of the Window Manager</td>
</tr>
<tr>
<td>WindReset</td>
<td>Resets the Window Manager; called only when the system is reset—must not be called by an application</td>
</tr>
<tr>
<td>WindStatus</td>
<td>Indicates whether the Window Manager is active</td>
</tr>
<tr>
<td><strong>Initialization and termination routines</strong></td>
<td></td>
</tr>
<tr>
<td>Desktop</td>
<td>Controls the addition of regions to and subtraction of regions from the desktop and controls the current desktop pattern</td>
</tr>
<tr>
<td>NewWindow</td>
<td>Creates a specified window as specified by its parameters, adds it to the window list, and returns a pointer to the new window's GrafPort</td>
</tr>
<tr>
<td>CloseWindow</td>
<td>Removes a specified window from the screen, disposes of all controls associated with that window, and deletes the window from the window list</td>
</tr>
<tr>
<td>WindNewRes</td>
<td>Closes the Window Manager's GrafPort and opens a new GrafPort in the other Super Hi-Res resolution</td>
</tr>
<tr>
<td><strong>Window record and global access routines</strong></td>
<td></td>
</tr>
<tr>
<td>GetWMgrPort</td>
<td>Returns a pointer to the Window Manager's port</td>
</tr>
<tr>
<td>SetWindowIcons</td>
<td>Sets the icon font for the Window Manager</td>
</tr>
<tr>
<td>SetWRefCon</td>
<td>Sets a value that is inside a specified window record and is reserved for the application's use</td>
</tr>
<tr>
<td>GetWRefCon</td>
<td>Returns a value from a specified window's record that was passed to either NewWindow or SetWRefCon by the application</td>
</tr>
<tr>
<td>SetWTitle</td>
<td>Changes the title of a specified window to a specified title and redraws the window</td>
</tr>
<tr>
<td>GetWTitle</td>
<td>Returns the pointer to a specified window's title</td>
</tr>
<tr>
<td>SetFrameColor</td>
<td>Sets the color of a specified window's frame</td>
</tr>
<tr>
<td>GetFrameColor</td>
<td>Returns the color of a specified window's frame</td>
</tr>
<tr>
<td>FrontWindow</td>
<td>Returns a pointer to the first visible window in the window list (that is, the active window)</td>
</tr>
</tbody>
</table>

25-2 Chapter 25: Window Manager
Table 25-1 (continued)
Window Manager routines and their functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window record and global access routines</td>
<td></td>
</tr>
<tr>
<td>GetFirstWindow</td>
<td>Returns a pointer to the first window in the window list (the window may not be the active window)</td>
</tr>
<tr>
<td>GetNextWindow</td>
<td>Returns a pointer to the next window in the window list after a specified window; returns NIL if the specified window is the last window in the window list</td>
</tr>
<tr>
<td>GetWKind</td>
<td>Indicates whether a specified window is a system window or an application window</td>
</tr>
<tr>
<td>SetWFrame</td>
<td>Sets the bit flag that describes a specified window’s frame type</td>
</tr>
<tr>
<td>GetWFrame</td>
<td>Returns the bit flag that describes a specified window’s frame type</td>
</tr>
<tr>
<td>GetStructRgn</td>
<td>Returns a handle to a specified window’s structure region</td>
</tr>
<tr>
<td>GetContentRgn</td>
<td>Returns a handle to a specified window’s content region</td>
</tr>
<tr>
<td>GetUpdateRgn</td>
<td>Returns a handle to a specified window’s update region</td>
</tr>
<tr>
<td>GetDefProc</td>
<td>Returns a pointer to the routine that is called to draw, hit test, and otherwise define a window’s frame and behavior</td>
</tr>
<tr>
<td>SetDefProc</td>
<td>Sets the pointer to the routine that is called to draw, hit test, and otherwise define a window’s frame and behavior</td>
</tr>
<tr>
<td>GetWControls</td>
<td>Returns the handle to the first control in the window’s control list</td>
</tr>
<tr>
<td>SetZoomRect</td>
<td>Sets the rectangle to be used as the content’s zoomed or unzoomed size for a specified window</td>
</tr>
<tr>
<td>GetZoomRect</td>
<td>Returns a pointer to the rectangle to be used as the content’s zoomed or unzoomed size for a specified window</td>
</tr>
<tr>
<td>GetSysWFlag</td>
<td>Indicates whether a specified window is a system or an application window</td>
</tr>
<tr>
<td>SetSysWindow</td>
<td>Marks a specified window as a system window</td>
</tr>
<tr>
<td>GetContentOrigin</td>
<td>Returns the values used by TaskMaster to set the origin of the window’s GrafPort when handling an update event</td>
</tr>
<tr>
<td>SetContentOrigin</td>
<td>Sets the origin of the window’s GrafPort when handling an update event</td>
</tr>
<tr>
<td>SetContentOrigin2</td>
<td>Sets the origin of the window’s GrafPort when handling an update event and allows the application to scroll or not scroll the window’s content region</td>
</tr>
<tr>
<td>SetOriginMask</td>
<td>Specifies the mask used to put the horizontal origin on a grid</td>
</tr>
<tr>
<td>StartDrawing</td>
<td>Makes a specified window the current port and sets its origin</td>
</tr>
<tr>
<td>GetDataSize</td>
<td>Returns the height and width of the data area of a specified window</td>
</tr>
<tr>
<td>SetDataSize</td>
<td>Sets the height and width of the data area of a specified window</td>
</tr>
<tr>
<td>GetMaxGrow</td>
<td>Returns the maximum values to which a specified window’s content region can grow</td>
</tr>
<tr>
<td>SetMaxGrow</td>
<td>Sets the maximum values to which a specified window’s content region can grow</td>
</tr>
<tr>
<td>GetScroll</td>
<td>Returns the number of pixels by which TaskMaster will scroll the content region when the user selects the arrows on window frame scroll bars</td>
</tr>
<tr>
<td>SetScroll</td>
<td>Sets the number of pixels by which TaskMaster will scroll the content region when the user selects the arrows on window frame scroll bars</td>
</tr>
</tbody>
</table>

(continued)
Table 25-1 (continued)
Window Manager routines and their functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Window record and global access routines</strong></td>
<td></td>
</tr>
<tr>
<td>GetPage</td>
<td>Returns the number of pixels by which TaskMaster will scroll the content region when the user selects the page regions on window frame scroll bars</td>
</tr>
<tr>
<td>SetPage</td>
<td>Sets the number of pixels by which TaskMaster will scroll the content region when the user selects the page regions on window frame scroll bars</td>
</tr>
<tr>
<td>GetContentDraw</td>
<td>Returns the pointer to the routine that draws the content region of a specified window</td>
</tr>
<tr>
<td>SetContentDraw</td>
<td>Sets the pointer to the routine to draw the content region of a specified window</td>
</tr>
<tr>
<td><strong>Information bar routines</strong></td>
<td></td>
</tr>
<tr>
<td>GetInfoDraw</td>
<td>Returns the pointer to the routine that draws the information bar for a specified window</td>
</tr>
<tr>
<td>SetInfoDraw</td>
<td>Sets the pointer to the routine that draws the information bar for a specified window</td>
</tr>
<tr>
<td>GetInfoRefCon</td>
<td>Returns the value associated with the draw information bar routine for a specified window</td>
</tr>
<tr>
<td>SetInfoRefCon</td>
<td>Sets the value associated with the draw information bar routine for a specified window</td>
</tr>
<tr>
<td>GetRectInfo</td>
<td>Sets the information rectangle to the coordinates of the information bar rectangle</td>
</tr>
<tr>
<td>StartInfoDrawing</td>
<td>Allows an application to draw or hit test outside of its information bar definition procedure</td>
</tr>
<tr>
<td>EndInfoDrawing</td>
<td>Puts the Window Manager back into a global coordinate system</td>
</tr>
<tr>
<td><strong>Window shuffling routines</strong></td>
<td></td>
</tr>
<tr>
<td>SelectWindow</td>
<td>Makes a specified window the active window</td>
</tr>
<tr>
<td>HideWindow</td>
<td>Makes a specified window invisible</td>
</tr>
<tr>
<td>ShowWindow</td>
<td>Makes a specified window visible if it was invisible and then draws the window</td>
</tr>
<tr>
<td>ShowHide</td>
<td>Shows or hides a window</td>
</tr>
<tr>
<td>BringToFront</td>
<td>Brings a specified window to the front of all other windows and redraws the windows as necessary but does not do any highlighting or unhighlighting</td>
</tr>
<tr>
<td>SendBehind</td>
<td>Changes the position of a specified window, redrawing any exposed windows</td>
</tr>
<tr>
<td><strong>Window drawing routines</strong></td>
<td></td>
</tr>
<tr>
<td>HiliteWindow</td>
<td>Highlights or unhighlights a specified window</td>
</tr>
<tr>
<td>RefreshDesktop</td>
<td>Redraws the entire desktop and all the windows</td>
</tr>
</tbody>
</table>
### Table 25-1 (continued)

**Window Manager routines and their functions**

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User interaction routines</strong></td>
<td></td>
</tr>
<tr>
<td>FindWindow</td>
<td>Indicates which part of which window, if any, the cursor was in when the user pressed the mouse button</td>
</tr>
<tr>
<td>DragWindow</td>
<td>Pulls around a dotted outline of a specified window, following the movements of the mouse until the mouse button is released</td>
</tr>
<tr>
<td>GrowWindow</td>
<td>Pulls around a grow image of a specified window, following the movements of the mouse until the mouse button is released</td>
</tr>
<tr>
<td>TrackGoAway</td>
<td>Tracks the mouse until the mouse button is released, highlighting the go-away region as long as the mouse location remains inside it and unhighlighting it when the mouse moves outside it</td>
</tr>
<tr>
<td>TrackZoom</td>
<td>Tracks the mouse until the mouse button is released, highlighting the zoom region as long as the mouse location remains inside it and unhighlighting it when the mouse moves outside it</td>
</tr>
<tr>
<td>TaskMaster</td>
<td>Calls GetNextEvent and looks in the event part of the task record to see if it can handle the event</td>
</tr>
<tr>
<td><strong>Window sizing and positioning routines</strong></td>
<td></td>
</tr>
<tr>
<td>MoveWindow</td>
<td>Moves a specified window to another part of the screen without affecting its size</td>
</tr>
<tr>
<td>SizeWindow</td>
<td>Enlarges or shrinks the port rectangle of a specified window's GrafPort to a specified width and height</td>
</tr>
<tr>
<td>ZoomWindow</td>
<td>Switches the size and position of a specified window between its current size and position and its maximum size</td>
</tr>
<tr>
<td>WindDragRect</td>
<td>Pulls a dotted outline of a specified rectangle around the screen, following the movements of the mouse until the mouse button is released</td>
</tr>
<tr>
<td><strong>Update region routines</strong></td>
<td></td>
</tr>
<tr>
<td>InvalRect</td>
<td>Accumulates a specified rectangle into the update region of the window whose GrafPort is the current port</td>
</tr>
<tr>
<td>InvalRgn</td>
<td>Accumulates a specified region into the update region of the window whose GrafPort is the current port</td>
</tr>
<tr>
<td>ValidRect</td>
<td>Removes a specified rectangle from the update region of the window whose GrafPort is the current port and tells the Window Manager to cancel any updates accumulated for that rectangle</td>
</tr>
<tr>
<td>ValidRgn</td>
<td>Removes a specified region from the update region of the window whose GrafPort is the current port and tells the Window Manager to cancel any updates accumulated for that region</td>
</tr>
<tr>
<td>BeginUpdate</td>
<td>Replaces the visible region of the window's GrafPort with the intersection of the visible region and the update region and then sets the window's update region to an empty region</td>
</tr>
<tr>
<td>EndUpdate</td>
<td>Restores the normal visible region of a specified window's GrafPort that was changed by a BeginUpdate call</td>
</tr>
<tr>
<td><strong>Miscellaneous routines</strong></td>
<td></td>
</tr>
<tr>
<td>PinRect</td>
<td>Pins a specified point inside a specified rectangle</td>
</tr>
<tr>
<td>CheckUpdate</td>
<td>Looks from top to bottom in the window list for a visible window that needs updating (that is, for a window whose update region is not empty)</td>
</tr>
</tbody>
</table>

* A preview of the Window Manager routines 25-5
Window frames and controls

There are two kinds of predefined window frames, document and alert. The alert window is used by the Dialog Manager and is explained in Chapter 6, “Dialog Manager,” in Volume 1. The document window, which is used by the Window Manager, is explained in this section. The two types of window frames are illustrated in Figure 25-2.

A document window may have any or all of the standard window controls, as listed below. The only restriction is that if there is a close or zoom box there must also be a title bar, and common sense would dictate that there only be a zoom box if there is a size box, although this is not a requirement. The standard controls include the following:

- **Title bar**, a rectangle at the top of the window that contains the window’s title, may hold the close and zoom boxes, and can be a drag region for moving the window
- **Close box**, a small region in the title bar that the user can select to remove the window from the screen
- **Zoom box**, a small region in the title bar that the user can select to make the window its maximum size and to return it to its previous size and position
- **Right scroll bar**, which the user selects to scroll vertically through the data in the window
- **Bottom scroll bar**, which the user selects to scroll horizontally through the data in the window
- **Size box**, a small region in the lower right corner of the window; the user can drag the size box to change the size of the window
- **Information bar**, a place in which an application can display some information that won’t be affected by the scroll bars
These standard controls, which can be used only for document windows and may not be added to alert windows, are illustrated in Figure 25-3.

Figure 25-3
Standard window controls

Some possible document window combinations are illustrated in Figure 25-4.

Figure 25-4
Sample document windows
You can either use the standard window types or create your own window types (see the section “Defining Your Own Windows” in this chapter). Some windows—such as the window the Dialog Manager creates to display an alert—may be created indirectly for you when you use other parts of the Toolbox. Windows created either directly or indirectly by an application are collectively called application windows. Another class of windows, called system windows, consists of windows in which desk accessories are displayed.

The Window Manager's main function is to keep track of overlapping windows. You can draw in any window without running over onto windows in front of it. You can move windows to different places on the screen, change their planes (front-to-back order), or change their sizes—all without concern for how the various windows overlap. The Window Manager keeps track of any newly exposed areas and provides a convenient mechanism with which you can ensure that they are properly redrawn.

You can also easily set up your application so that mouse actions cause the following standard responses inside a document window (or similar responses inside other windows):

- Clicking anywhere in an inactive window makes it the active window by bringing it to the front and highlighting it.
- Clicking inside the close box of the active window closes the window. Depending on the application, this may mean that the window disappears altogether or that a representation of the window (such as an icon) is left on the desktop.
- Dragging anywhere inside the title bar of a window (except in a close or zoom box) pulls an outline of the window across the screen; releasing the mouse button moves the window to the new location. If the window isn't the active window, it becomes the active window unless the Apple key was also held down. A window can never be moved completely off the screen; by convention, it can't be moved such that the visible area of the title bar is less than four pixels square.
- Dragging inside the size box of the active window changes the size of the window.
Window regions

Every window has a content region—the area in which your application draws—and a frame region—the outline of the entire window plus any standard window controls. Together, the content and frame regions make up the structure region.

The content region is bounded by the rectangle you specify when you create the window (that is, the rectangle specified in the portRect field of the window's GrafPort). This region is where your application presents information to the user.

A window may also have any of the following regions within the window frame:

- Go-away region, a close box in the active window. Clicking in this region closes the window.
- Drag region, the title bar. Dragging in this region pulls an outline of the window across the screen, moves the window to a new location, and makes it the active window (unless it was already the active window or unless the Apple key was held down).
- Grow region, the size box. Dragging in this region pulls the lower right corner of an outline of the window across the screen with the window's origin fixed, resizes the window, and makes it the active window (unless it was already the active window or unless the Apple key was held down).
- Zoom region, the zoom box in the active window. Clicking in this region toggles from the current position and size to a maximum size and back again.

Clicking in any region of an inactive window makes it the active window.

Data and content areas and scroll bars

Windows act like a microfiche machine. What is seen in the window's content region is like what is seen on the viewer. Similarly, the window's data area is what the microfiche is to the viewer. Through the content region, the user can see part of the data area unless the content region is large enough to view the entire data area.

Scroll bars are the devices used for scrolling the data area through the content region and showing the relationship between the data area and content region. Because scroll bars are handled by the Control Manager, the Control Manager must be loaded and started up before scroll bars can be used in windows. The following paragraphs explain how standard window scroll bars act in relationship to windows.

The scroll bar is like a reduced cross section of the work area. The scroll thumb has the same ratio to the page region as the content region has to the data area, as illustrated in Figure 25-5.
The size and zoom boxes are used to increase or decrease the amount of the data area displayed at one time. When the window is moved, the data area is moved with it so the view in the content region remains the same.

**Using the Window Manager**

This section discusses how the Window Manager routines fit into the general flow of an application and gives you an idea of which routines you'll need to use under normal circumstances. Each routine is described in detail later in this chapter.

The Window Manager depends on the presence of the tool sets shown in Table 25-2 and requires that at least the indicated version of each tool set be present.

<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
<th>Minimum version needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01 #01</td>
<td>Tool Locator</td>
<td>1.2</td>
</tr>
<tr>
<td>$02 #02</td>
<td>Memory Manager</td>
<td>1.2</td>
</tr>
<tr>
<td>$03 #03</td>
<td>Miscellaneous Tool Set</td>
<td>1.2</td>
</tr>
<tr>
<td>$04 #04</td>
<td>QuickDraw II</td>
<td>1.2</td>
</tr>
<tr>
<td>$06 #06</td>
<td>Event Manager</td>
<td>1.0</td>
</tr>
</tbody>
</table>
The first Window Manager call that your application must make is WindStartUp. Conversely, when you quit your application, you must make the WindShutDown call. Where appropriate in your program, use NewWindow to create any windows you need.

There are two ways to handle user input in relation to windows. You can poll the user via TaskMaster, which will handle most events that deal with standard user interfaces (see the section "Using TaskMaster" in this chapter).

If you are not using TaskMaster, you must poll for events by calling GetNextEvent in the Event Manager. Whenever your application receives an update event, the application should respond as follows:

1. Call the BeginUpdate routine. This routine temporarily replaces the visible region of the window's GrafPort with the intersection of the visible region and the update region. It then clears the update region for that window.
2. Draw the window contents.
3. Call the EndUpdate routine to restore the actual visible region.

Activate events for dialog and alert windows are handled by the Dialog Manager. In response to activate or inactivate events for windows created directly by your application, you might take actions such as the following:

- Inactivate controls in inactive window and activate controls in active windows.
- Remove the highlighting or blinking cursor from text being edited when the window becomes inactive and restore it when the window becomes active.
- Enable or disable a menu or certain menu items appropriately to match what the user can do when windows become active or inactive.

If you are not using TaskMaster and a mouse-down event occurs, the application should call FindWindow to see if the button was pressed inside a window. The following are results from FindWindow and the standard actions to take:

- **wlnMenuBar**: Mouse-down somewhere outside the desktop. If you have not subtracted any area from the desktop, there is a good chance the button was pressed in the system menu bar. Call the Menu Manager routine MenuSelect.
- **wlnDrag**: Mouse-down in a window’s drag region; it may or may not be the active window. Call DragWindow.
- **wlnContent**: Mouse-down in window’s content region. Call SelectWindow if the window is not the active window. Otherwise, handle the event according to your application.
- **wlnGoAway**: Mouse-down in active window’s go-away region. Call TrackGoAway. If the routine returns TRUE, you may want to give the user the opportunity to save the window; then call CloseWindow or HideWindow.
- **wlnZoom**: Mouse-down in active window’s zoom region. Call TrackZoom. If TrackZoom returns TRUE, call ZoomWindow.
- **wlnGrow**: Mouse-down in active window’s grow region. Call GrowWindow.
Using TaskMaster

TaskMaster is a procedure that can handle many standard functions. When TaskMaster is called instead of GetNextEvent; the sequence of events is as follows:

1. TaskMaster calls GetNextEvent.

2. If there isn't an event ready, TaskMaster returns 0.

   If an event is ready, TaskMaster looks at it and tries to handle it.

3. If TaskMaster can't handle the event that is ready, it returns the event code to the application. The application can handle the event as if it had come from GetNextEvent.

   If TaskMaster can handle the event, it calls standard routines to try to complete the task. For example, if the user presses the mouse button in an active window's zoom region, TaskMaster detects it, calls TrackZoom, calls ZoomWindow (if the user actually selects the zoom region), and returns no event.

Sometimes TaskMaster can handle an event only up to a point. If the user presses the mouse in the active window's content region, TaskMaster detects it but won't be able to go any further, so it returns winContent, which tells the application the mouse button is down in the active window's content region.

We provide TaskMaster for two reasons. First, it should help you get an application running as quickly as possible and still allow you to take advantage of the standard user interface. TaskMaster should be usable by even the most advanced applications, although some alternate algorithms may have to be used to get the desired results.

Second, TaskMaster provides upward compatibility in the years to come. If an application is using TaskMaster, a modification to TaskMaster to take advantage of some new feature will not adversely affect the application; in fact, your application may be using the new feature without any modification on your part!

When calling TaskMaster, you pass a pointer to a TaskMaster record, TaskRec. The beginning of the record is the same as an event record. When TaskMaster calls GetNextEvent, it passes the provided pointer so that the event record part of TaskRec is set by GetNextEvent. The structure of the task record is shown in Figure 25-6.
The `wmtaskMask` is used by your application to tell TaskMaster about functions you would like it to ignore. The `wmtaskMask` is defined as shown in Figure 25-7.
Figure 25-7
The wmtaskMask bit flag

25-14 Chapter 25: Window Manager
Important
At the time of publication, bits 31-16 must be set to 0. In fact, TaskMaster will return an error if they are not. Because these bits will mask off as yet unknown features, applications will continue to run even when the new features are added.

Window Manager icon font
The standard document window definition uses a font to draw the close and zoom boxes, and their highlighted states, in a window's title. If you would like to use different icons, you can replace the default font. To replace the icon font, or just to get the handle to the current font, call SetWindowIcons. The format of the font is shown in Table 25-3.

Table 25-3
Window Manager Icon font

<table>
<thead>
<tr>
<th>Character</th>
<th>Icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Close box</td>
</tr>
<tr>
<td>1</td>
<td>Highlighted close and zoom boxes</td>
</tr>
<tr>
<td>2</td>
<td>Zoom box</td>
</tr>
<tr>
<td></td>
<td>(same character for both)</td>
</tr>
</tbody>
</table>

Window record
The Window Manager keeps all the information it requires for its operations on a particular window in a window record. The record contains the information the Window Manager needs to manage windows. The complete window record is accessed directly only by the Window Manager. Your application can directly access only the part of the window record illustrated in Figure 25-8.

Not allowing direct access to the entire window record has advantages and disadvantages. Access to window information is slower if calls have to be made to the Window Manager. However, the delay could only be measured in milliseconds and can't be seen on the screen. On the plus side, future Window Managers won't be tied to an older, possibly inadequate, record structure. The chances of improving the current Window Manager without affecting existing applications and of maintaining compatibility across future hardware are greatly improved by allowing the window record to change.
Many Window Manager calls need as input a window pointer that is returned from NewWindow. That pointer points to the window’s GrafPort.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>wNext</td>
<td>Long—POINTER to next window in window list</td>
</tr>
<tr>
<td>1</td>
<td>port</td>
<td>170 bytes—Window’s GrafPort; returned window pointers point to this</td>
</tr>
<tr>
<td>2</td>
<td>wPadding</td>
<td>16 bytes—Reserved for future use</td>
</tr>
<tr>
<td>3</td>
<td>wStructRgn</td>
<td>Long—HANDLE to window’s entire region; frame plus content</td>
</tr>
<tr>
<td>4</td>
<td>wContRgn</td>
<td>Long—HANDLE to window’s content region</td>
</tr>
<tr>
<td>5</td>
<td>wUpdateRgn</td>
<td>Long—HANDLE to region that needs redrawing</td>
</tr>
<tr>
<td>6</td>
<td>wControls</td>
<td>Long—HANDLE to application’s first control in content region</td>
</tr>
<tr>
<td>7</td>
<td>wFrameCtrls</td>
<td>Long—HANDLE to frame’s first control</td>
</tr>
<tr>
<td>8</td>
<td>wFrame</td>
<td>Word—Bit flag that defines window</td>
</tr>
</tbody>
</table>

Figure 25-8
Window record

The settings for the wFrame parameter are described in the section “NewWindow” in this chapter.
Windows and GrafPorts

It's easy for your application to use windows; to the application, a window is a GrafPort the application can draw into with QuickDraw II routines. When you create a window, you specify a RECT data structure that becomes the port rectangle of the GrafPort in which the window contents will be drawn. The bit map for this GrafPort, its pen pattern, and other characteristics are the same as the default values set by QuickDraw II. These characteristics apply whenever the application draws in the window, and they can easily be changed with QuickDraw II routines.

There is, however, more to a window than just the GrafPort in which the application draws. The other part of a window is called the window frame because it usually surrounds the rest of the window. For drawing window frames, the Window Manager creates a GrafPort that has the entire screen as its port rectangle.

Window frame colors and patterns

In addition to the standard window types and controls, the color of the window and controls can be selected. Each 4-bit color is an index into either the default color table or a color table pointed to by the wColor field in the NewWindow parameter list (see Table 25-8 in the section "NewWindow" in this chapter). See Chapter 16, "QuickDraw II," for more information about color tables.

The color table for document and alert windows is shown in Figure 25-9.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>nameColor</td>
<td>Word—Color of window frame and alert frame</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bits 15-8 = 0 Bits 7-4 = Outline color Bits 3-0 = 0</td>
</tr>
<tr>
<td>1</td>
<td>titleColor</td>
<td>Word—Color of inactive title bar, inactive title, and active title</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bits 15-12 = 0 Bits 11-8 = Inactive title bar color Bits 7-4 = Inactive title bar color Bits 3-0 = Color of title, close, and zoom boxes</td>
</tr>
<tr>
<td>2</td>
<td>titleColor</td>
<td>Word—Color and pattern of title bar</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bits 15-8 = Solid, 5000 = Dither, 5000 = Lined Bits 7-4 = Pattern color</td>
</tr>
<tr>
<td>3</td>
<td>titleColor</td>
<td>Word—Color of size box and alert frame's middle outline</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bits 15-12 = Color of alert frame's middle outline Bits 7-4 = Interior color of size box when not selected Bits 3-0 = Interior color of size box when selected</td>
</tr>
<tr>
<td>4</td>
<td>titleColor</td>
<td>Word—Color of information bar and alert frame's inside outline</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bits 15-12 = Color of alert frame's inside outline Bits 7-4 = Interior color of information bar Bits 3-0 = 0</td>
</tr>
</tbody>
</table>

Figure 25-9
Document and alert window color table
Figures 25-10 through 25-14 show how these colors are used.

**Figure 25-10**
Window frame color (`frameColor`)

**Figure 25-11**
Window title color (`titleColor`)

25-18  Chapter 25: Window Manager
**Figure 25-12**
Window title bar color (*tBarColor*)

**Figure 25-13**
Window size box and alert window's middle outline color (*growColor*)
Figure 25-14
Window information bar and alert window’s inside outline color (infoColor)

Use the SetFrameColor routine to set the color table a window should use and the GetFrameColor routine to get a pointer to the window’s current color table.

How a window is drawn

When a window is drawn or redrawn, the window frame is drawn first, followed by the window contents.

To draw the window frame, the Window Manager manipulates regions of the Window Manager port as necessary to ensure that only what should be drawn is drawn. It then calls the window definition procedure with a request that the window frame be drawn. The window definition procedure is either within the Window Manager or in the application for custom windows (see the section “Defining Your Own Windows” in this chapter).

To draw the window contents, the Window Manager generates an update event. Whenever your application receives an update event, the application should respond as follows:

1. Call the BeginUpdate routine. This routine temporarily replaces the visible region of the window’s GrafPort with the intersection of the visible region and the update region. It then clears the update region for that window.
2. Draw the window contents.
3. Call the EndUpdate routine to restore the actual visible region.

Update events are issued for the frontmost window first and the hindmost last.

25-20   Chapter 25: Window Manager
**Draw content routine**

When the NewWindow call is used to open a new window, the Window Manager checks the \textit{wContDefProc} field. If that field is nonzero, the value is considered to be the address of a routine in your application that will draw the window's content region.

The \textit{wContDefProc} field must be set if you want to use window frame scroll bars (which are the scroll bars TaskMaster creates). TaskMaster will scroll the content and call \textit{wContDefProc} to update the uncovered area when the user performs a scrolling action. \textit{wContDefProc} could be considered a control action procedure.

The \textit{wContDefProc} field might be useful even if you are not using window frame scroll bars. TaskMaster can handle your update events if \textit{wContDefProc} is set. TaskMaster will call BeginUpdate, \textit{wContDefProc}, and EndUpdate.

There are no inputs or outputs to your draw content routine.

\textit{Note:} Use the QuickDraw II routine \textit{GetPort} to obtain the current window pointer.

Draw what is needed in the content and perform an RTL to exit. Remember that the content will have already been erased using the window port's background pattern and that the visible region is set to the area needing to be redrawn.

---

**Warning**

Do not change ports or perform a QuickDraw II SetOrigin call while in your draw content routine.

---

**Draw information bar routine**

If the \textit{fInfo} bit (bit 4) is set to 1 in the \textit{wFrameBits} field of the NewWindow parameter list, the window will have an information bar that appears just above the content region. The width of the information bar is same as the width of the window, and the height of the information bar is specified by the \textit{wInfoHeight} field in the NewWindow parameter list.

The Window Manager draws the information bar, but it is up to the application to draw any information inside the bar. Your application can do this by storing the address of a draw information bar routine in the \textit{wInfoDefProc} field of the NewWindow parameter list (and you must also set the \textit{fInfo} bit of \textit{wFrame}). When the standard window frame definition procedure draws the empty information bar, it will also call the procedure pointed to by \textit{wInfoDefProc}.

---

How a window is drawn 25-21
The inputs to your routine will be as follows:

- `infoBarPtr` — Long—POINTER to RECT data structure specifying enclosing rectangle
- `infoData` — Long—`wInfoRefCon` value from NewWindow parameter list
- `theWindowPtr` — Long—POINTER to window’s GrafPort
- `RTL` — 3 bytes—RTL address

An assembly-language example of a draw information bar routine that prints a string looks like this:

```assembly
InfoDefProc  START
  theWindow   equ   6
  InfoData    equ   theWindow+4
  InfoBar     equ   InfoData+4
  phd          ; Save the current direct page
  tsc          ; Switch to direct page in stack
  tcd

  ; --- Position the pen at the text starting point ---------------------------
  ldy  #left_side              ; (Where left_side equals 2)
  lda  [InfoBar],y             ; Get the left side of the information bar,
  clc
  adc  #20                     ; plus a tab over, to get
  pha

  ;
  ldy  #top_side               ; (Where top_side equals 0)
  lda  [InfoBar],y             ; Get the top side of the information bar,
  clc
  adc  #10                     ; plus enough to vertically center the text, to
  pha

  ; _MoveTo                      ; get a starting Y position (pass to _MoveTo)

  ; Move the pen to the starting point.
```

25-22 Chapter 25: Window Manager
; --- Print the text on the information bar ---------------------------------
  pea infoStrg|-16               ; Pass high word of string
  pea infoStrg                   ; Pass low word of string
  _DrawString                    ; Print the string

; --- All done, now clean up stack and return to Window Manager -----------
  ply                            ; Get original direct page back
  lda 2,s                        ; Move return down over input parameters
  sta <14                        ; Works because stack and direct page are equal
  lda 0,s                        ;
  sta <12
  tsc                            ; Now move stack pointer over input parameters
  clc
  adc $12                        ; Number of bytes of input parameters
  tcs                            ; New stack
  tya                            ; Restore original direct page
  tcd
  rtl                            ; Back to Window Manager

infoStrg dc '8',c'MyTitle'

END

The example takes some liberties, such as assuming the color and writing mode of the pen when the text is written. When entered, the current port is the Window Manager's. You may change the pen location, color, and writing mode without saving the original port state. However, that's as much as you should do without first saving the port state and then restoring it on exit.

Another liberty the example takes is when the text is centered vertically. You should make QuickDraw calls to find font height, to find the InfoBar height, and then to actually center the text. You should always use InfoBar as offsets into the information bar interior because the height could be different from time to time.

---

**Important**

Do not change the current port's clipRgn or visRgn fields unless you save and restore the original value.
Making a window active: activate events

A number of Window Manager routines change the state of a window from inactive to active or from active to inactive. For each such change, the Window Manager generates an activate event, passing along the window pointer in the event message. The activeFlag bit in the modifiers field of the event record is set if the window has become active; it is cleared if it has become inactive.

When the Event Manager finds out from the Window Manager that an activate event has been generated, it passes the event on to the application (via the GetNextEvent function). Activate events have the highest priority of any type of event.

Usually when one window becomes active another becomes inactive and vice versa, so activate events are most commonly generated in pairs. When this happens, the Window Manager generates first the event for the window becoming inactive and then the event for the window becoming active. Sometimes only a single activate event is generated—such as when there's only one window in the window list or when the active window is permanently discarded (because it no longer exists).

Activate events for dialog and alert windows are handled by the Dialog Manager. In response to activate or inactivate events for windows created directly by your application, you might take actions such as the following:

- Inactivate controls in inactive window and activate controls in active windows.
- Remove the highlighting or blinking cursor from text being edited when the window becomes inactive and restore it when the window becomes active.
- Enable or disable a menu or certain menu items to match what the user can do when windows become active or inactive.
Defining your own windows

You may want to define your own type of window, such as a round or hexagonal window; QuickDraw and the Window Manager allow you to define your own window shape.

To define your own type of window, you must write a window definition procedure that defines the appearance and behavior of the window. Then, when the Window Manager needs to do something, it calls your routine and not its own.

You pass the address to the NewWindow call in the wFrameDefProc parameter (see the section "NewWindow" in this chapter). The inputs to your routine are

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>varCode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Word—Operation to be performed</td>
</tr>
<tr>
<td>-- theWindowPtr --</td>
<td>Long—POINTER to window’s GrafPort</td>
</tr>
<tr>
<td>-- params --</td>
<td>Long—Parameters used by some messages</td>
</tr>
<tr>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>outcomeFlag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Long—Returned parameter flag</td>
</tr>
<tr>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

The varCode parameters are shown in Table 25-4.

Table 25-4

The varCode parameters for custom windows

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>wDraw</td>
<td>Draw window frame</td>
</tr>
<tr>
<td>1</td>
<td>wHit</td>
<td>Tell what region the cursor was in when the mouse button was pressed</td>
</tr>
<tr>
<td>2</td>
<td>wCalcRgn</td>
<td>Calculate wStructRgn and wContRgn</td>
</tr>
<tr>
<td>3</td>
<td>wNew</td>
<td>Do any additional window initialization</td>
</tr>
<tr>
<td>4</td>
<td>wDispose</td>
<td>Take any additional disposal actions</td>
</tr>
<tr>
<td>5</td>
<td>wGrow</td>
<td>Draw window’s grow image</td>
</tr>
</tbody>
</table>
What you can expect in response to each of the codes is described immediately following the \textit{wContDefProc} example.

Your routine must strip off the three input parameters and return via RTL. So the shell of your \textit{wContDefProc} routine might be as follows:

\begin{verbatim}
MyWindow  START
  lda  12,s     ; Get varCode
  asl a
  tax
  lda >actions,x
  pha
  rts         ; Go to action handler

actions   dc i2'draw_wind-1' ; Routine to draw window's frame
          dc i2'test_hit-1'  ; Routine to find a window region
                         ; at a given point
          dc i2'calc_rgns-1' ; Compute window's \texttt{wStructRgn} and \texttt{wContRgn}
          dc i2'init_wind-1' ; Do additional initialization
          dc i2'kill_wind-1'  ; Do additional disposal
END

draw_wind  START
  ...
  (code that draws window frame)
  jmp  exit
END

test_hit   START
  ...
  (code that finds area of the window in which the point in params is located)
  jmp  exit
END

calc_rgns  START
  ...
  (code that computes the window's \texttt{wStructRgn} and \texttt{wContRgn})
  jmp  exit
END
\end{verbatim}
init_wind  START

...  

(code that performs additional initialization)

jmp exit

END

kill_wind  START

(code that performs additional disposal)

jmp exit

END

exit  START

lda 2, s ; Move return address
sta 12, s
lda 1, s
sta 11, s
tsc ; Strip off input parameters
sec
sbc #10
tcs
rtl ; Return to Window Manager

END

wDraw: draw a window frame

Your routine should draw in the current GrafPort, which will be the Window Manager port. The Window Manager will request this operation only if the window is visible.

The structure of the params parameter is as follows:

<table>
<thead>
<tr>
<th>wDrawFrame</th>
<th>$00</th>
<th>Draw window's entire frame.</th>
</tr>
</thead>
<tbody>
<tr>
<td>wInGoAway</td>
<td>$01</td>
<td>Draw go-away region.</td>
</tr>
<tr>
<td>wInZoom</td>
<td>$02</td>
<td>Draw zoom region.</td>
</tr>
<tr>
<td>Bit 31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Draw frame or region as highlighted.</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Draw frame or region as unhighlighted.</td>
<td></td>
</tr>
</tbody>
</table>

Thus, valid params values are as follows:

$00000000  Draw entire window frame as an inactive window.
$08000000  Draw entire window frame as an active window.
$00000001  Draw go-away region as unhighlighted.
$08000001  Draw go-away region as highlighted.
$00000002  Draw zoom region as unhighlighted.
$08000002  Draw zoom region as highlighted.
**wHit: find what region a point is in**

The `params` parameter specifies the point to check. The vertical coordinate is in the low-order word and the horizontal coordinate in the high-order word. The Window Manager requests this operation only if the window is visible. Your routine should determine where the point is in your window and then return one of the following:

- `wNoHit`: 0 Not on the window at all
- `winContent`: 19 In window's content region
- `winDrag`: 20 In window's drag (title bar) region
- `winGrow`: 21 In window's size box region
- `winGoAway`: 22 In window's go-away (close box) region
- `winZoom`: 23 In window's zoom (zoom box) region
- `winInfo`: 24 In window's information bar
- `winFrame`: 27 In window, but not in any of the above areas

Usually, `wNoHit` means the given point isn't anywhere within the window, but this is not necessarily so.

**wCalcRgns: calculate a window's regions**

Your routine should calculate the window's entire region, place it in the `wStructRgn`, and place the content region in `wContRgn` based on the current GrafPort's port rectangle. The Window Manager requests this operation only if the window is visible.

**Warning**

When you calculate regions for your window, do not alter the `clipRgn` or `visRgn` fields of the window's GrafPort.

**wNew: perform additional initialization**

After initializing fields appropriately when creating a new window, the Window Manager sends the message `wNew` to your routine. This gives your routine a chance to perform any initialization it may require. For example, because the structure of the window record is not documented, you may want to allocate your own record structure, initialize it, and store its pointer via a `SetWRefCon` call.
wDispose: remove a window

The Window Manager's CloseWindow and DisposeWindow procedures send this message so your routine can carry out any additional actions required when disposing of the window. The routine might, for example, release space that was allocated by the initialize routine. The routine is called before all controls in the wControls and wFrameCtrls lists are removed via a KillControls call to the Control Manager, the port is closed, and the window record freed. Return 0 to continue closing or 1 to abort closing.

wGrow: draw the outline of a window

The *params* parameter is a pointer to a RECT data structure defining a rectangle. Your routine should draw an outline image of your window that would fit the specified rectangle. The Window Manager requests this operation repeatedly as the user drags inside the grow region. Your routine should use the GrafPort's current pen pattern and pen mode.

Origin movement

This section describes in detail how the origin of a window can change and what the effects of that change are. To benefit from the following discussion, you should already be familiar with QuickDraw II's explanation of ports and boundsRect.

The origin of a window is what allows data to be scrolled and drawing to occur in the proper place after a scroll. In Figure 25-15, the gray area is a screen with the pixel in its upper left corner being 0,0 (coordinates are shown here as Y,X). The window port appears on the screen at 65,50 to 85,80. These points are called **global coordinates**. To draw the house, the X coordinate of the left side would be 60; that is, it would be 10 pixels inside the window port.

![Figure 25-15](image.png)

*Figure 25-15*

Window origin
However, a window port has its own coordinates, called **local coordinates**. Figure 25-16 shows what really happens when a window is created. Although the window is still at 65,50 to 85,80 on the screen, the local coordinates of the window are 0,0 to 20,30. Notice that the window's height and width remain the same. Also notice that the window is now called the port rectangle and the screen the bounds rectangle.

To draw the left wall of this house, you would pass the X coordinate of 10 to QuickDraw II to draw a single vertical line. QuickDraw II would then subtract the X origin of the boundsRect to determine where on the screen to actually draw the line. So, 10 minus -50 is 60. The global coordinate is 60; the local was 10. You will always work with local coordinates; that way, if the window is moved, its coordinate system remains the same. This explains what happens to the horizontal axis; the same thing happens with the vertical axis.

The Window Manager changes the coordinates of the bounds rectangle when the window is moved. If the window is moved one pixel to the left, the boundsRect would become -65,-49. When the coordinate of 10 is passed to QuickDraw II, it computes the global coordinate of 59 (10 minus -49). Thus, even though the house is drawn on another place on the screen, it is drawn in the same place in the window, and the application doesn't have to make any changes.

![Figure 25-16](image)

**Figure 25-16**
Window moving and origins
In the previous simple examples, everything had origins of 0,0. However, one of the powerful features of windows is their ability to scroll to show more data than the screen allows. In the next example, the window has not moved, but the user has scrolled the picture using the bottom scroll bar (see Figure 25-17). When the user moved the thumb on the scroll bar, the rectangle (0,10,20,30) was scrolled (moved) to (0,0,20,20). Then the origin of the window was changed to 0,10, and the exposed rectangle on the right side was redrawn in an update event.

After the scrolling occurred, the application would pass the coordinate of 10 to draw the left side of the house and QuickDraw II would compute 10 minus -40 to get the global coordinate of 50. Now, a minor problem arises: the Window Manager needs every window to have an origin of 0,0 for it to move, grow, and overlap windows. This feature may eventually change, but for now, whenever the Window Manager is called, the origin of every window must be 0,0.

![Figure 25-17](image)

Scrolling and window origins

For example, whenever TaskMaster calls your update routine, it switches to the window's port and sets its origin; in this example it would be SetOrigin(0,10). Then you can draw in the window's local coordinates. When you have finished drawing and return to TaskMaster it performs a SetOrigin(0,0).

Even though changing the origin does not change the screen, any drawing outside of your update routine without setting the origin would have undesirable results. Drawing your house with the origin still at 0,0 would produce two houses, one shifted 10 pixels to the right of the other. To draw outside of your update routine, you need to first set the origin either yourself or through a StartDrawing call and then perform a SetOrigin(0,0) to put it back.

In short, when drawing outside of your update routine, you must perform a StartDrawing call before drawing and a SetOrigin(0,0) when you are finished drawing. This is also true when you are performing a hit test in your content region; the event position must be converted to local coordinates.
$010E  \textbf{WindBootInit}

Initializes the Window Manager; called only by the Tool Locator.

\textit{Warning}
An application must never make this call.

\textbf{Parameters}  The stack is not affected by this call. There are no input or output parameters.

\textbf{Errors}  None

C  Call must not be made by an application.

$020E  \textbf{WindStartUp}

Starts up the Window Manager for use by an application.

\textit{Important}
Your application must make this call before it makes any other Window Manager calls.

\textbf{Parameters}

\textbf{Stack before call}
\begin{tabular}{l|l}
\textit{previous contents} & \textbf{Word}—ID number of application \\
\hline
\textbf{userID} & \textbf{SP}
\end{tabular}

\textbf{Stack after call}
\begin{tabular}{l|l}
\textit{previous contents} & \textbf{SP}
\end{tabular}

\textbf{Errors}  None

C  extern pascal void WindStartUp(userID)

\hspace{1cm} \text{Word} \hspace{1cm} \text{userID;}

25-32  Window Manager housekeeping routines
$030E  WindShutDown

Shuts down the Window Manager when an application quits.

Important
If your application has started up the Window Manager, the application must make this call before it quits.

Parameters
The stack is not affected by this call. There are no input or output parameters.

Errors
None

C
extern pascal void WindShutDown()

$040E  WindVersion

Returns the version number of the Window Manager.

Parameters

Stack before call

\[
\begin{array}{c|c}
\text{previous contents} & \text{wordspace} \\
\text{wordspace} & \text{Word—Space for result} \\
\end{array}
\]

← SP

Stack after call

\[
\begin{array}{c|c}
\text{previous contents} & \text{versioninfo} \\
\text{versioninfo} & \text{Word—Version number of Window Manager} \\
\end{array}
\]

← SP

Errors
None

C
extern pascal Word WindVersion()
$050E  **WindReset**

Resets the Window Manager; called only when the system is reset.

---

**Warning**

An application must never make this call.

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

**Errors**

None

C

Call must not be made by an application.

---

$060E  **WindStatus**

Indicates whether the Window Manager is active.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>wordspace</th>
<th>Word—Space for result</th>
</tr>
</thead>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>activeFlag</th>
<th>Word—BOOLEAN; TRUE if Window Manager active, FALSE if inactive</th>
</tr>
</thead>
</table>

**Errors**

None

C

extern pascal Boolean WindStatus()
$1E0E \textbf{BeginUpdate}

Replaces the visible region of the window's GrafPort with the intersection of the visible region and the update region and then sets the window's update region to an empty region.

Call BeginUpdate when an update event occurs for the window. You would then usually draw the entire content region, although it suffices to draw only the visible region. In either case, only the parts of the window that require updating and are visible will actually be drawn on the screen. Every call to BeginUpdate must be balanced by a call to the EndUpdate routine, as follows:

1. Call BeginUpdate.
2. Draw the window contents.
3. Call the EndUpdate routine to restore the actual visible region.

**Parameters**

Stack before call

previous contents

--- theWindowPtr ---

\text{Long—POINTER to window's GrafPort}

\leftarrow \text{SP}

Stack after call

--- previous contents ---

\leftarrow \text{SP}

**Errors**

None

**C**

\text{extern pascal void BeginUpdate(theWindowPtr)}

\text{GrafPortPtr theWindowPtr;}

Window Manager routines 25-35
$240E  BringToFront

Brings a specified window to the front of all other windows and redraws the windows as necessary but does not do any highlighting or unhighlighting. Normally you won't have to call this procedure; you should call SelectWindow to make a window active, and SelectWindow takes care of bringing the window to the front. If you do call BringToFront, however, remember to call HilliteWindow to make any necessary highlighting changes.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Long—POINTER to window's GrafPort</th>
</tr>
</thead>
<tbody>
<tr>
<td>theWindowPtr</td>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

| previous contents | ← SP                             |

Errors

None

C

extern pascal void BringToFront(theWindowPtr)

GrafPortPtr theWindowPtr;

25-36  Window Manager routines
$0A0E  CheckUpdate

Looks from top to bottom in the window list for a visible window that needs updating (that is, for a window whose update region is not empty). CheckUpdate is normally called only by the Event Manager, and doesn't need to be called by an application.

If a window with something in its update region is found, an update event for that window is stored in the event and the routine returns TRUE. If it doesn't find such a window, it returns FALSE.

Parameters

Stack before call

```
  previous contents
  
  wordspace
  
  theEventPtr
  
  Word—Space for result

  Long—POINTER to an event record

  ← SP
```

Stack after call

```
  previous contents
  
  updateFlag
  
  Word—BOOLEAN; TRUE if update event found, otherwise FALSE

  ← SP
```

Errors

None

C

```c
extern pascal Boolean CheckUpdate(theEventPtr)
EventRecordPtr  theEventPtr;
```
CloseWindow

Removes a specified window from the screen, disposes of all controls associated with that window, and deletes the window from the window list. The routine releases the memory occupied by all data structures associated with the window, including the memory taken up by the window record if it was allocated by NewWindow. Call this routine when you're done with a window.

Any update events for the window are discarded. If the deleted window was the frontmost window, the window behind it (if any) is highlighted and an appropriate activate event is generated.

Warning
If you allocated memory yourself and stored a handle to it in the wRefCon field, CloseWindow won't know about it—you must release the memory before calling CloseWindow.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to window's GrafPort

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

Errors
None

C

extern pascal void CloseWindow(theWindowPtr)

GrafPortPtr theWindowPtr;

25-38 Window Manager routines
$0C0E  Desktop

Controls the addition of regions to and subtraction of regions from the desktop and controls the current desktop pattern. The values for the `deskTopOp` and `dtParam` parameters are shown in Table 25-5.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
<tr>
<td>deskTopOP</td>
</tr>
<tr>
<td>dtParam</td>
</tr>
</tbody>
</table>

Long—Space for result if necessary

Word—Operation to perform

Long—Parameter needed for operation

<- SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>retParam</td>
</tr>
</tbody>
</table>

Long—Possible return parameter (see Table 25-5)

<- SP

Errors

None

C

extern pascal Pointer Desktop(deskTopOP, dtParam)

Word    deskTopOP;

LongWord dtParam;

(continued)
<table>
<thead>
<tr>
<th>Operation</th>
<th>Operation number</th>
<th>Description and dtParam and retParam values</th>
</tr>
</thead>
<tbody>
<tr>
<td>FromDesk</td>
<td>0</td>
<td>Subtract region from desktop region. DTParam—HANDLE to region to be subtracted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RetParam—Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The region passed in dtParam will be subtracted from the current desktop region. When the Window Manager</td>
</tr>
<tr>
<td></td>
<td></td>
<td>redraws the desktop, it will not draw the region subtracted. Windows will not cover the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>subtracted region and will appear to move underneath the region. For example, the Menu Manager</td>
</tr>
<tr>
<td></td>
<td></td>
<td>subtracts a region from the desktop for the system menu bar, which is why windows move under the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system menu bar. FromDesk can be called repeatedly to remove additional areas from the desktop.</td>
</tr>
<tr>
<td>ToDesk</td>
<td>1</td>
<td>Add region to current desktop region. DTParam—HANDLE to region to be added</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RetParam—Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When the Window Manager redraws the desktop, it will also draw the region added. ToDesk can be called</td>
</tr>
<tr>
<td></td>
<td></td>
<td>repeatedly to add additional areas to the desktop.</td>
</tr>
<tr>
<td>GetDesktop</td>
<td>2</td>
<td>Return handle to current desktop region. DTParam—Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RetParam—HANDLE to desktop region</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The handle returned is the actual handle to the desktop region; any modifications to this region will</td>
</tr>
<tr>
<td></td>
<td></td>
<td>change the desktop. Using the handle, you can add, subtract, or XOR a region; find intersections;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and perform other region operations. This call can also be used for restoring the desktop to its</td>
</tr>
<tr>
<td></td>
<td></td>
<td>original shape after modifying it for some temporary use (see SetDesktop for restoring the desktop).</td>
</tr>
</tbody>
</table>

**Warning**

Do not free the handle of the desktop region; that will cause the Window Manager to crash.
Table 25-5 (continued)
Window Manager Desktop routine operations and parameters

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operation number</th>
<th>Description and dtParam and retParam values</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetDesktop</td>
<td>3</td>
<td>Set handle to desktop region.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>dtParam</em>—HANDLE to new desktop region</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>retParam</em>—Same as <em>param</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td>After SetDesktop is called, the new handle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>will be used by the Window Manager for the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>handle of the desktop region. It is not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>necessary to call SetDesktop if you have</td>
</tr>
<tr>
<td></td>
<td></td>
<td>called GetDesktop and modified the region</td>
</tr>
<tr>
<td></td>
<td></td>
<td>because GetDesktop returns the actual</td>
</tr>
<tr>
<td></td>
<td></td>
<td>desktop handle.</td>
</tr>
<tr>
<td>GetDeskPat</td>
<td>4</td>
<td>Return current desktop pattern.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>dtParam</em>—Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>retParam</em>—Current desktop pattern (see Table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25-6)</td>
</tr>
<tr>
<td>SetDeskPat</td>
<td>5</td>
<td>Set new desktop pattern. The desktop is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>redrawn with the new pattern.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>dtParam</em>—New desktop pattern (see Table 25-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>retParam</em>—Not used</td>
</tr>
<tr>
<td>GetVisDesktop</td>
<td>6</td>
<td>Return desktop, less any windows.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>dtParam</em>—Handle to region that will be set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to the visible desktop</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>retParam</em>—Not used</td>
</tr>
</tbody>
</table>

Warning

SetDesktop overwrites the current desktop region handle. Therefore, it is up to the application to free the original handle or save it and restore it later.

...
<table>
<thead>
<tr>
<th>Operation</th>
<th>Operation number</th>
<th>Description and dtParam and retParam values</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackGroundRgn</td>
<td>7</td>
<td>Maintain visible desktop region.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>dtParam</em>—Handle to a region</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>retParam</em>—Not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The region passed will be set to the desktop region less any windows. The region is automatically updated when windows are added, removed, sized, and moved. This operation provides applications an easy way of drawing objects directly on the desktop. A possible sequence for drawing objects on the desktop might be as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>; Space for result (not used)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>; Operation number 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>; Pass address of my routine</td>
</tr>
<tr>
<td></td>
<td></td>
<td>; that will draw the desktop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>; Result not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>; Open a port for my desktop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>; Space for result (not used)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>; Operation number 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>; Pass handle of my desktop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>; port's visRgn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>; Result not used</td>
</tr>
</tbody>
</table>

25-42 Window Manager routines
Table 25-5 (continued)
Window Manager Desktop routine operations and parameters

<table>
<thead>
<tr>
<th>Operation number</th>
<th>Description and wParam and reParam values</th>
</tr>
</thead>
<tbody>
<tr>
<td>After this code is executed, the routine MyDeskDraw will be called by the Window Manager whenever the desktop needs to be drawn. The preceding code passed the value of the visRgn field of a port to the Window Manager. The Window Manager will use that value to compute the visible desktop. Then, when an application wants to draw an object on the desktop, it can switch to MyDeskPort and draw. All drawing will be clipped to the current visible desktop.</td>
<td></td>
</tr>
</tbody>
</table>

Note: The address of the routine passed to the Desktop routine for operation SetDeskPat (operation 5) is still a routine that is called with the current port being the Window Manager's with its clip region set to the visible desktop. To draw whenever you want, you'll have to use your own port and your own visible region.

Desktop patterns

There are no inputs to or outputs from the routine called to draw the desktop. The current port will be the Window Manager's, and the clipping region will be set to the area needing to be drawn. Your routine should exit via an RTL.

Warning

The current direct page and data bank are not defined on entry to your routine. When you exit your routine, the direct page and data bank must be the same as they were on entry.

The desktop pattern is determined by a long value, as shown in Table 25-6.

Table 25-6
Desktop patterns

<table>
<thead>
<tr>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$00</td>
<td>Address of routine that will be called to draw desktop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$80</td>
<td>Address of pattern to be used for desktop (see Chapter 16, &quot;QuickDraw II&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$40</td>
<td>00</td>
<td>00 = Solid desktop pattern</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>01 = Dithered desktop pattern</td>
<td></td>
</tr>
<tr>
<td></td>
<td>02</td>
<td>02 = Horizontally striped desktop pattern</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>High nibble = Pattern’s foreground color</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Low nibble = Pattern’s background color</td>
<td></td>
</tr>
</tbody>
</table>

Window Manager routines 25-43
$1AOE DragWindow

Pulls around a dotted outline of a specified window, following the movements of the mouse until the mouse button is released. When the button is released, DragWindow calls MoveWindow to move the specified window to the location to which it was dragged. The window will be dragged and moved in its current plane.

When there is a mouse-down event in the drag region of the specified window and TaskMaster is not being used, the application should call DragWindow with startX, startY equal to the point where the mouse button was pressed (in global coordinates, as stored in the where field of the event record).

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word—Drag resolution, zero for default (see Table 25-7)</td>
</tr>
<tr>
<td>Word—Starting X coordinate of cursor, in global coordinates</td>
</tr>
<tr>
<td>Word—Starting Y coordinate of cursor, in global coordinates</td>
</tr>
<tr>
<td>Word—Grace buffer around bounds</td>
</tr>
<tr>
<td>Long—POINTER to RECT structure for cursor boundary, NIL for default</td>
</tr>
<tr>
<td>Long—POINTER to window’s GrafPort</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Errors

None

C

extern pascal void DragWindow(grid, startX, startY, grace, boundsRectPtr, theWindowPtr)

Word grid;
Integer startX;
Integer startY;
Word grace;
Rect *boundsRectPtr;
GrafPortPtr theWindowPtr;

25-44 Window Manager routines
You can also use the following alternate form of the call:

```pascal
extern pascal void DragWindow(grid,start,grace,boundsRectPtr,theWindowPtr)
Word grid;
Point start;
Word grace;
Rect *boundsRectPtr;
GrafPortPtr theWindowPtr;
```

**Parameter description**

**grid**

The allowed horizontal resolution movement, as shown in Table 25-7. The *grid* parameter is provided to speed up window moves by eliminating the need for bit shifting, if the *grid* value is the correct value. If *grid* is passed as 0, a default value will be used. The defaults are 4 for 320 mode and 8 for 640 mode. The only allowed values are 1, 2, 4, 8, 16, 32, 64, 128, and so on.

**Table 25-7**

<table>
<thead>
<tr>
<th>Value</th>
<th>Window movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Default value used.</td>
</tr>
<tr>
<td>1</td>
<td>Window can be positioned at any horizontal position.</td>
</tr>
<tr>
<td>2</td>
<td>Window can only be moved a multiple of 2 pixels horizontally.</td>
</tr>
<tr>
<td>4</td>
<td>Window can only be moved a multiple of 4 pixels horizontally.</td>
</tr>
<tr>
<td>8</td>
<td>Window can only be moved a multiple of 8 pixels horizontally.</td>
</tr>
</tbody>
</table>

The only allowed values are 0, 1, 2, 4, 8, 16, 32, 64, 128, and so on.

**startY and startX**

Indicate where the mouse button was pressed, in global coordinates, as stored in the *where* field of the event record. This point is used with the tracked cursor position to compute the movement delta.

(continued)
The distance, in pixels, that you will allow the cursor to move away from the bounds rectangle before the dragged outline should be snapped back to its starting position. TaskMaster uses 8 for this value. The bounds rectangle is expanded by the value of grace to compute the slop rectangle that is passed to DragRect as the `slopRect` parameter. See the section "DragRect" in Chapter 4, "Control Manager," in Volume 1 for more information.

Pointer to a RECT data structure, in global coordinates, that is passed to DragRect as the `limitRect` parameter. See the section "DragRect" in Chapter 4, "Control Manager," in Volume 1 for more information. If NIL is passed for the pointer, the bounds of the desktop, minus 4 all around, will be used.
**EndInfoDrawing**

Puts the Window Manager back into a global coordinate system. Call this routine after a StartInfoDrawing call and before any other calls to the Window Manager.

**Warning**

Calling any Window Manager routine between a StartInfoDrawing call and an EndInfoDrawing call may result in system failure.

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

**Errors**

None

**C**

```c
extern pascal void EndInfoDrawing();
```

---

**EndUpdate**

Restores the normal visible region of a specified window's GrafPort that was changed by a BeginUpdate call.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to window's GrafPort

| ← SP |

**Stack after call**

| previous contents |
| ← SP |

**Errors**

None

**C**

```c
extern pascal void EndUpdate(theWindowPtr);
GrafPortPtr theWindowPtr;
```

Window Manager routines 25-47
FindWindow

Tells which part of which window, if any, the cursor was in when the user pressed the mouse button. If it was pressed in a window, the whichWindowPtr parameter is set to the window port pointer; otherwise, it's set to NIL.

When a mouse-down event occurs, the application should, if not using TaskMaster, call FindWindow with pointX and pointY equal to the point where the cursor was when the user pressed the mouse button (in global coordinates, as stored in the where field of the event record).

Parameters

Stack before call

| previous contents |  
|-------------------|---
| workspace         |  
| whichWindowPtr    |  
| pointX            |  
| pointY            |  

Stack after call

| previous contents |  
|-------------------|---
| location          |  

Errors

None

C

```
extern pascal Word FindWindow(whichWindowPtr,pointX,pointY)

GrafPortPtr *whichWindowPtr;
Integer pointX;
Integer pointY;

You can also use the following alternate form of the call:

extern pascal Word FindWindow(whichWindowPtr,point)

GrafPortPtr *whichWindowPtr;
Point point;
```

Window Manager routines
Mouse-down event location information

The *location* returned by FindWindow is one of the constants shown in Table 25-8.

<table>
<thead>
<tr>
<th>Word</th>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000</td>
<td>wNoHit</td>
<td>Not in the window at all</td>
</tr>
<tr>
<td>$0010</td>
<td>wInDesk</td>
<td>In the desktop area</td>
</tr>
<tr>
<td>$0011</td>
<td>wInMenuBar</td>
<td>In the system menu bar</td>
</tr>
<tr>
<td>$0013</td>
<td>wInContent</td>
<td>In window's content region</td>
</tr>
<tr>
<td>$0014</td>
<td>wInDrag</td>
<td>In window's drag (title bar) region</td>
</tr>
<tr>
<td>$0015</td>
<td>wInGrow</td>
<td>In window's size box region</td>
</tr>
<tr>
<td>$0016</td>
<td>wInGoAway</td>
<td>In window's go-away (close box) region</td>
</tr>
<tr>
<td>$0017</td>
<td>wInZoom</td>
<td>In window's zoom (zoom box) region</td>
</tr>
<tr>
<td>$0018</td>
<td>wInInfo</td>
<td>In window's information bar</td>
</tr>
<tr>
<td>$0019</td>
<td>wInSpecial</td>
<td>In special menu item bar (see Chapter 13, &quot;Menu Manager,&quot; in Volume 1)</td>
</tr>
<tr>
<td>$001A</td>
<td>wInDeskItem</td>
<td>Desk accessory selected from the Apple menu</td>
</tr>
<tr>
<td>$001B</td>
<td>wInFrame</td>
<td>In window, but not in any of the areas defined in this table</td>
</tr>
<tr>
<td>$001C</td>
<td>wInactMenu</td>
<td>Inactive menu item selected</td>
</tr>
<tr>
<td>$8xxx</td>
<td>wInSysWindow</td>
<td>In a system window</td>
</tr>
</tbody>
</table>

Window Manager routines 25-49
FrontWindow

Returns a pointer to the first visible window in the window list (that is, the active window). If there are no visible windows, the routine returns NIL.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
</tbody>
</table>

Long—Space for result

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to active window’s GrafPort; NIL if none visible

← SP

Errors

None

C

extern pascal GrafPortPtr FrontWindow()
GetContentDraw

Returns the pointer to the routine that draws the content region of a specified window. TaskMaster calls this routine when it gets an update event for that window. See the section "Draw Content Routine" in this chapter for more information about the draw routine.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>contentProcPtr</td>
</tr>
</tbody>
</table>

Errors

None

C

extern pascal VoidProcPtr GetContentDraw(theWindowPtr)

GrafPortPtr theWindowPtr;
$3EOE  GetContentOrigin

Returns the values used by TaskMaster to set the origin of the window's GrafPort when handling an update event. The values are also used to compute scroll bars in the window frame.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

Long—Space for result
Long--POINTER to window's GrafPort
← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>origin</td>
</tr>
</tbody>
</table>

Long—POINT; low word = Y origin, high word = X origin
← SP

Errors

None

C

extern pascal Long GetContentOrigin(theWindowPtr)

Pointer theWindowPtr;

Note: C Pascal-type functions do not deal properly with data structures returned on the stack. The Long result returned by this call can be passed to any calls requiring a point as a parameter. You cannot use the C dot operator to access the individual Y and X coordinates within the value returned by this call.
$2F0E \textbf{GetContentRgn}

Returns a handle to a specified window's content region. See the section "Window Regions" in this chapter for a definition of the content region.

\textbf{Parameters}

\textbf{Stack before call}

\begin{tabular}{l|l}
\hline
\textit{previous contents} & \textbf{Long}—Space for result \\
\hline
\textit{longspace} & \textbf{Long}—POINTER to window's GrafPort \\
\hline
\textit{theWindowPtr} & \leftarrow \text{SP} \\
\hline
\end{tabular}

\textbf{Stack after call}

\begin{tabular}{l|l}
\hline
\textit{previous contents} & \textbf{Long}—HANDLE to window's content region \\
\hline
\textit{wContHandle} & \leftarrow \text{SP} \\
\hline
\end{tabular}

\textbf{Errors}

None

\textbf{C}

\begin{verbatim}
extern pascal RgnHandle GetContentRgn( theWindowPtr )
GrafPortPtr theWindowPtr;
\end{verbatim}
$400E GetDataSize

Returns the height and width of the data area of a specified window. The data area is the total amount of data that can be viewed in a window through resizing or scrolling.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- longspace --</td>
</tr>
<tr>
<td>-- theWindowPtr  --</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- dataSize --</td>
</tr>
</tbody>
</table>

Errors
None

C
extern pascal Longword GetDataSize(theWindowPtr)
GrafPortPtr theWindowPtr;
GetDefProc

Returns a pointer to the routine that is called to draw, hit test, and otherwise define a window's frame and behavior.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

- Long—Space for result
- Long—POINTER to window's GrafPort

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wDefProcPtr</td>
</tr>
</tbody>
</table>

- Long—POINTER to window's definition procedure

Errors
None

C

extern pascal LongProcPtr GetDefProc(theWindowPtr)
GrafPortPtr theWindowPtr;
GetFirstWindow

Returns a pointer to the first window in the Window Manager's window list. The returned window may not be the active window (see the section "FrontWindow" in this chapter). Every window in the window list, whether visible or not, can be accessed if you call GetFirstWindow and then call the GetNextWindow routine to run down the remainder of the window list.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
<tr>
<td>Long--Space for result</td>
</tr>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstWindowPtr</td>
</tr>
<tr>
<td>Long--POINTER to first window, or NIL</td>
</tr>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Errors

None

C

extern pascal GrafPortPtr GetFirstWindow()
$100E \textbf{GetFrameColor}

Returns the color of a specified window's frame. See the section "Window Frame Colors and Patterns" in this chapter for a definition of the color table.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>colorPtr</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

\textbf{Long}—POINTER to five-word table to be filled with window's color table

\textbf{Long}—POINTER to window's GrafPort

\leftarrow SP

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

\leftarrow SP

**Errors**

None

**C**

\texttt{extern pascal void GetFrameColor(colorPtr, theWindowPtr)}

\texttt{WindColorPtr colorPtr;}

\texttt{GrafPortPtr theWindowPtr;}

Window Manager routines 25-57
GetInfoDraw

Returns the pointer to the routine that draws the information bar for a specified window. If the window has an information bar routine, the standard window definition procedure calls that routine whenever the window's frame needs to be drawn. See the section "Draw Information Bar Routine" in this chapter for more information about that routine.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- longspace</td>
</tr>
<tr>
<td>-- theWindowPtr</td>
</tr>
</tbody>
</table>

Long—Space for result

Long—POINTER to window's GrafPort

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- infoDrawPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to routine to draw information bar

← SP

Errors

None

C

extern pascal VoidProcPtr GetInfoDraw(theWindowPtr)

GrafPortPtr theWindowPtr;
Returns the value of a specified window's \texttt{winInfoRefCon} field (the value associated with the draw information bar routine). The field is reserved for application use.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{longspace}</td>
</tr>
<tr>
<td>\texttt{theWindowPtr}</td>
</tr>
</tbody>
</table>

\texttt{Long}—Space for result

\texttt{Long}—POINTER to window's GrafPort

\texttt{SP}

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{infoRefCon}</td>
</tr>
</tbody>
</table>

\texttt{Long}—Value passed to draw information bar routine

\texttt{SP}

**Errors**

None

**C**

\begin{verbatim}
extern pascal LongWord GetInfoRefCon(theWindowPtr)
GrafPortPtr theWindowPtr;
\end{verbatim}
$420E \textbf{GetMaxGrow} \\
Returns the maximum values to which a specified window's content region can grow.

\textbf{Parameters}

\textbf{Stack before call}

\begin{itemize}
  \item \textit{previous contents}
  \item \textit{longspace} \quad \textbf{Long}—Space for result
  \item \textit{theWindowPtr} \quad \textbf{Long}—POINTER to window's GrafPort
\end{itemize}

\textbf{Stack after call}

\begin{itemize}
  \item \textit{previous contents}
  \item \textit{maxGrow} \quad \textbf{Long}—Low word = \textit{maxHeight}; high word = \textit{maxWidth}
\end{itemize}

\textbf{Errors} \quad \textbf{None}

\textbf{C}

\begin{verbatim}
extern pascal LongWord GetMaxGrow(theWindowPtr)
GrafPortPtr theWindowPtr;
\end{verbatim}
GetNextWindow

Returns a pointer to the next window in the window list after a specified window; returns NIL if the specified window is the last window in the window list.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

| Long—Space for result |
| Long—POINTER to window's GrafPort |

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextWindowPtr</td>
</tr>
</tbody>
</table>

| Long—POINTER to next window's GrafPort in list; NIL if last window |

Errors

None

C

extern pascal GrafPortPtr GetNextWindow(theWindowPtr)

GrafPortPtr theWindowPtr;
$460E \hspace{1cm} \textbf{GetPage}

Returns the number of pixels by which TaskMaster will scroll the content region when the user selects the page regions on window frame scroll bars.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

- **Long**—Space for result
- **Long**—POINTER to window's GrafPort

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>pageAmount</td>
</tr>
</tbody>
</table>

- **Long**—Low word = vertical amount; high word = horizontal amount

**Errors**

None

**C**

```c
extern pascal LongWord GetPage(theWindowPtr)

GrafPortPtr theWindowPtr;
```
GetRectInfo

Sets the information rectangle to the coordinates of the information bar rectangle. If there is no information bar in the specified window, the coordinates of the RECT data structure pointed to by infoRectPtr will all be 0. The coordinate system will be local to the window's frame; that is, 0,0 will be the upper left corner of the window. The coordinates can be used to set the position of objects that will be drawn in the information bar.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>infoRectPtr</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
</tr>
</tbody>
</table>

Errors

None

C

extern pascal void GetRectInfo(infoRectPtr, theWindowPtr)

Rect *infoRectPtr;
GrafPortPtr theWindowPtr;
GetScroll

Returns the number of pixels by which TaskMaster will scroll the content region when the user selects the arrows on window frame scroll bars.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- longspace --</td>
</tr>
<tr>
<td>-- theWindowPtr --</td>
</tr>
</tbody>
</table>

Long—Space for result

Long—POINTER to window's GrafPort

<- SP

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- scrollAmount --</td>
</tr>
</tbody>
</table>

Long—Low word = vertical amount; high word = horizontal amount

<- SP

**Errors**

None

**C**

```c
extern pascal LongWord GetScroll(theWindowPtr);
GrafPortPtr theWindowPtr;
```

25-64 Window Manager routines
GetStructRgn

Returns a handle to a specified window's structure region. See the section "Window Regions" in this chapter for a definition of the structure region.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- longspace</td>
</tr>
<tr>
<td>-- theWindowPtr</td>
</tr>
</tbody>
</table>

Long—Space for result
Long—POINTER to window's GrafPort
← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- wStructHandle</td>
</tr>
</tbody>
</table>

Long—HANDLE to window's structure region
← SP

Errors

None

C

extern pascal RgnHandle GetStructRgn(theWindowPtr)
GrafPortPtr theWindowPtr;
**GetSysWFlag**

Indicates whether a specified window is a system window or an application window.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

- **Word**—Space for result
- **Long**—POINTER to window's GrafPort

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysFlag</td>
</tr>
</tbody>
</table>

- **Word**—BOOLEAN; TRUE if system window, FALSE if application window

**Errors**

None

**C**

```c
extern pascal Boolean GetSysWFlag(theWindowPtr)
GrafPortPtr theWindowPtr;
```
$300E \textbf{GetUpdateRgn}

Returns a handle to a specified window's update region. See the section "BeginUpdate" in this chapter for an explanation of how the update region is used.

\textbf{Parameters}

\textbf{Stack before call}

\begin{center}
\begin{tabular}{|c|c|}
\hline
\multicolumn{2}{|c|}{\textit{previous contents}} \\
\hline
\hline
\textit{longspace} & \textbf{Long}—Space for result \\
\hline
\textit{theWindowPtr} & \textbf{Long}—POINTER to window's GrafPort \\
\hline
\end{tabular}
\end{center}

\textbf{Stack after call}

\begin{center}
\begin{tabular}{|c|c|}
\hline
\multicolumn{2}{|c|}{\textit{previous contents}} \\
\hline
\hline
\textit{wUpdateHandle} & \textbf{Long}—HANDLE to window's update region \\
\hline
\end{tabular}
\end{center}

\textbf{Errors}

None

\textbf{C}

\begin{verbatim}
extern pascal RgnHandle GetUpdateRgn\(\textit{theWindowPtr}\)
GrafPortPtr \textit{theWindowPtr};
\end{verbatim}
GetWControls

Returns the handle to the first control in the window's control list. The window's control list is the list of controls created by the application with calls to NewControl in the Control Manager.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

- **Long**—Space for result
- **Long**—POINTER to window's GrafPort

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>controlHandle</td>
</tr>
</tbody>
</table>

- **Long**—HANDLE to first control in window's control list; NIL if none

Errors

None

C

```c
extern pascal CtlRecHndl GetWControls(theWindowPtr)

GrafPortPtr theWindowPtr;
```
GetWFrame

Returns the bit flag that describes a specified window's frame type. See the discussion of the wFrame field in the section "NewWindow" in this chapter for the definition of the bits in the flag.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td></td>
</tr>
<tr>
<td>theWindowPtr</td>
<td></td>
</tr>
</tbody>
</table>

| previous contents      |          |
| wFrame                 |          |

Word—Space for result

Long—POINTER to window's GrafPort

← SP

Stack after call

| previous contents      |          |
| wFrame                 |          |

Word—Bit flag with window's frame type

← SP

Errors

None

C

extern pascal Word GetWFrame(theWindowPtr)

GrafPortPtr theWindowPtr;
$2B0E  GetWKind

Indicates whether a specified window is a system window or an application window.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

Word—Space for result
Long—PCINTER to window's GrafPort
← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>windowKind</td>
</tr>
</tbody>
</table>

Word—Bit 15 is 0 for application window, 1 for system window
← SP

Errors

None

C

extern pascal Word GetWKind(theWindowPtr)
GrafPortPtr theWindowPtr;

25-70  Window Manager routines
$200E \textbf{GetWMgrPort}

Returns a pointer to the Window Manager's port.

**Parameters**

**Stack before call**

\begin{center}
\begin{tabular}{l|l}
\textit{previous contents} & \\
\hline
\textit{longspace} & \textbf{Long—Space for result} \\
\end{tabular}
\end{center}

\textit{← SP}

**Stack after call**

\begin{center}
\begin{tabular}{l|l}
\textit{previous contents} & \\
\hline
\textit{wPortPtr} & \textbf{Long—POINTER to Window Manager's GrafPort} \\
\end{tabular}
\end{center}

\textit{← SP}

**Errors**

None

**C**

\texttt{extern pascal GrafPortPtr GetWMgrPort()}

Window Manager routines 25-71
$290E  GetWRefCon

Returns a value from a specified window's record that was passed to either NewWindow or SetWRefCon by the application. The wRefCon field is reserved for use by the application.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Longspace</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

Long—Space for result
Long—POINTER to window's GrafPort

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wRefCon</td>
</tr>
</tbody>
</table>

Long—Current value of wRefCon field

Errors

None

C

extern pascal longword GetWRefCon(theWindowPtr)

GrafPortPtr theWindowPtr;
$0EOE  GetWTitle

Returns the pointer to a specified window's title. The string pointed to by titlePtr is a Pascal-type string.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

Long—Space for result
Long—POINTER to window's GrafPort
← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>titlePtr</td>
</tr>
</tbody>
</table>

Long—POINTER to window's title string
← SP

Errors

None

C

extern pascal Pointer GetWTitle(theWindowPtr)
GrafPortPtr theWindowPtr;
GetZoomRect

Returns a pointer to the rectangle to be used as the content's zoomed or unzoomed size for a specified window. If the zoom flag is set in the frame flag (see the section "GetWFrame" in this chapter), then \texttt{wZoomSizePtr} points to a RECT data structure that contains the window's last size and position. Otherwise, \texttt{wZoomSizePtr} points to a RECT data structure that contains the size and position of the window's content region (port) the next time the window is zoomed by a call to ZoomWindow.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\hspace{1em}</td>
</tr>
<tr>
<td>\hspace{1em} \hspace{1em}</td>
</tr>
</tbody>
</table>

\hspace{1em} --- \texttt{theWindowPtr} \hspace{1em} \texttt{\rightarrow SP}

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\hspace{1em}</td>
</tr>
</tbody>
</table>

\hspace{1em} --- \texttt{wZoomSizePtr} \hspace{1em} \texttt{\rightarrow SP}

Errors

None

C

\begin{verbatim}
extern pascal Rect * GetZoomRect (theWindowPtr)
GrafPortPtr theWindowPtr;
\end{verbatim}
$1B0E GrowWindow

Pulls around a grow image of a specified window, following the movements of the mouse until the mouse button is released. The grow image for a document window is a dotted outline of the entire window plus the lines delimiting the title bar, size box, and scroll bar areas. See Figure 25-18.

When there's a mouse-down event in the size box region of the specified window, the application should call GrowWindow with startY and startX equal to the point where the mouse button was pressed (in global coordinates, as stored in the where field of the event record).

Parameters

Stack before call

```
| previous contents |
|--|--
| iongspace | -- |
| minWidth |
| minHeight |
| startX |
| startY |
| theWindowPtr |
```

- **Long**—Space for result
- **Word**—Minimum width of content region
- **Word**—Minimum height of content region
- **Word**—Starting X coordinate of cursor, in global coordinates
- **Word**—Starting Y coordinate of cursor, in global coordinates
- **Long**—POINTER to window's GrafPort

Stack after call

```
| previous contents |
|--|--
| newSize |
```

- **Long**—High word = new width; low word = new height

Errors

None
C

extern pascal Longword GrowWindow(minWidth, minHeight, startX, startY, theWindowPtr)
Word    minWidth;
Word    minHeight;
Integer  startX;
Integer  startY;
GrafPortPtr theWindowPtr;

You can also use the following alternate form of the call:

extern pascal Longword GrowWindow(minWidth, minHeight, start, theWindowPtr)
Word    minWidth;
Word    minHeight;
Point    start;
GrafPortPtr theWindowPtr;

Grow image

Figure 25-18 illustrates the grow image for a document window that contains both scroll bars. In general, the grow image is defined in the window definition function to appropriately show that the window's size will change.

Figure 25-18
Grow Image of a window

25-76     Window Manager routines
Your application should subsequently call the SizeWindow routine to change the port rectangle of the window’s GrafPort to the new one outlined by the grow image. The sizeRect parameter specifies limits, in pixels, on the height (vertical measurement) and width (horizontal measurement) of what will be the new port rectangle. The top coordinate of sizeRect is the minimum vertical measurement, the left coordinate is the minimum horizontal measurement, the bottom coordinate is the maximum vertical measurement, and the right coordinate is the maximum horizontal measurement.

GrowWindow returns the actual size for the new port rectangle as outlined by the grow image when the mouse button is released. The high-order word of the long is the horizontal measurement in pixels; the low-order word is the vertical measurement. A return value of 0 indicates that the size is the same as that of the current port rectangle.
$120E  HideWindow

Makes a specified window invisible. If the window is the frontmost window and there's a window behind it, HideWindow also unhighlights the window, brings the window behind it to the front, highlights that window, and generates appropriate activate events. If the specified window is already invisible, HideWindow has no effect.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to window's GrafPort

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Errors

None

C

extern pascal void HideWindow(theWindowPtr)

GrafPortPtr theWindowPtr;

25-78  Window Manager routines
HiliteWindow

Highlights or unhighlights a specified window, depending on the value of a specified parameter. If \texttt{fHiliteFlag} is \texttt{TRUE}, this routine highlights the window. If \texttt{fHiliteFlag} is \texttt{FALSE}, HiliteWindow unhighlights the window. The exact way a window is highlighted and unhighlighted depends on its window definition procedure.

Normally you won’t have to call this routine because you should call SelectWindow to make a window active and SelectWindow takes care of the necessary highlighting changes. To conform with the Apple \textit{Human Interface Guidelines}, don’t highlight a window that isn’t the active window.

**Parameters**

**Stack before call**

\begin{verbatim}
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>fHiliteFlag</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>
\end{verbatim}

**Stack after call**

\begin{verbatim}
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>
\end{verbatim}

\begin{verbatim}
| SP |
\end{verbatim}

**Errors**

None

\texttt{C}

\begin{verbatim}
extern pascal void HiliteWindow(fHiliteFlag, theWindowPtr)

Boolean fHiliteFlag;
GrafPortPtr theWindowPtr;
\end{verbatim}
$3A0E \textbf{InvalRect} \\
Accumulates a specified rectangle into the update region of the window whose 
GrafPort is the current port. This tells the Window Manager that the rectangle has 
changed and must be updated. The rectangle is given in local coordinates and is then 
clipped by the Window Manager to the window's content region.

\underline{Important} 
This routine changes the coordinates you give it. Save the coordinates if you 
need to restore them later.

\textbf{Parameters}

\textbf{Stack before call}

\begin{itemize}
  \item \texttt{previous contents}
  \item \texttt{badRectPtr}
  \item Long-POINTER to RECT data structure of rectangle to be added
\end{itemize}

\texttt{← SP}

\textbf{Stack after call}

\begin{itemize}
  \item \texttt{previous contents}
\end{itemize}

\texttt{← SP}

\textbf{Errors}

None

\textbf{C}

\begin{verbatim}
extern pascal void InvalRect(badRectPtr)
Rect *badRectPtr;
\end{verbatim}
$3B0E \textbf{InvalRgn}

Accumulates a specified region into the update region of the window whose GrafPort is the current port. This tells the Window Manager that the region has changed and must be updated. The region is given in local coordinates and is then clipped by the Window Manager to the window's content region.

\begin{itemize}
  \item \textbf{Important:}
  This routine changes the coordinates you give it. Save the coordinates if you need to restore them later.
\end{itemize}

\section*{Parameters}

\subsection*{Stack before call}

\begin{verbatim}
  previous contents
  \hspace{1cm}badRgnHandle \hspace{1cm}Long-HANDLE to region to be added to update region
  \hspace{1cm}← SP
\end{verbatim}

\subsection*{Stack after call}

\begin{verbatim}
  previous contents
  ← SP
\end{verbatim}

\section*{Errors}

None

\begin{verbatim}
  \textbf{C}
  \textbf{external} \textbf{pascal} \textbf{void} InvalRgn(badRgnHandle)
  RgnHandle badRgnHandle;
\end{verbatim}
**MoveWindow**

Moves a specified window to another part of the screen without affecting the window's size. The upper left corner of the window's port rectangle is moved to the screen point $newY$, $newX$. The local coordinates of the window's top left corner remain the same.

### Parameters

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$newX$</td>
</tr>
<tr>
<td>$newY$</td>
</tr>
<tr>
<td>$theWindowPtr$</td>
</tr>
</tbody>
</table>

- **Word**—New X coordinate of content region's upper left corner (global)
- **Word**—New Y coordinate of content region's upper left corner (global)
- **Long**—POINTER to window's GrafPort

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

- **SP**

### Errors

None

**C**

```c
extern pascal void MoveWindow($newX$, $newY$, $theWindowPtr$);
```

Integer $newX$;

Integer $newY$;

GrafPortPtr $theWindowPtr$;

You can also use the following alternate form of the call:

```c
extern pascal void MoveWindow($newPoint$, $theWindowPtr$);
```

Point $newPoint$;

GrafPortPtr $theWindowPtr$;

---

25-82 Window Manager routines
NewWindow

Creates a specified window as specified by its parameters, adds it to the window list, and returns a pointer to the new window's GrafPort. NewWindow allocates space for the structure and content regions of the window and asks the window definition function to calculate those regions.

Important
NewWindow does not set the current port, but many routines require that a current port exist. Use the QuickDraw II routine SetPort to set the current port.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>longspace</td>
</tr>
<tr>
<td>paramListPtr</td>
</tr>
</tbody>
</table>

Long—Space for result
Long—POINTER to parameter list (see Table 25-9)

← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>theWindow</td>
</tr>
</tbody>
</table>

Long—POINTER to window's GrafPort; NIL if error

← SP

Errors

$0E01  paramLenErr  First word of parameter list is the wrong size
$0E02  allocateErr  Unable to allocate memory for window record

C

extern pascal GrafPortPtr NewWindow(paramListPtr)

ParamListPtr  paramListPtr;

(continued)
NewWindow parameter list

The NewWindow parameter list is shown, and each parameter is briefly described, in Table 25-9.

Table 25-9
NewWindow parameter list

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>paramLength</td>
<td>Word</td>
<td>Total number of bytes in parameter table, including the paramLength parameter itself</td>
</tr>
<tr>
<td>wFrameBits</td>
<td>Word</td>
<td>Bit flag that describes window frame type</td>
</tr>
<tr>
<td>wTitle</td>
<td>Long</td>
<td>Pointer to window's title</td>
</tr>
<tr>
<td>wRefCon</td>
<td>Long</td>
<td>Reserved for application use</td>
</tr>
<tr>
<td>wZoom</td>
<td>4 words</td>
<td>RECT specifying size and position of content when zoomed</td>
</tr>
<tr>
<td>wColor</td>
<td>Long</td>
<td>Pointer to window's color table</td>
</tr>
<tr>
<td>wYOrigin</td>
<td>Word</td>
<td>Vertical offset of content region from data area</td>
</tr>
<tr>
<td>wXOrigin</td>
<td>Word</td>
<td>Horizontal offset of content region from data area</td>
</tr>
<tr>
<td>wDataH</td>
<td>Word</td>
<td>Height of entire document</td>
</tr>
<tr>
<td>wDataW</td>
<td>Word</td>
<td>Width of entire document</td>
</tr>
<tr>
<td>wMaxH</td>
<td>Word</td>
<td>Maximum height of content allowed by GrowWindow</td>
</tr>
<tr>
<td>wMaxW</td>
<td>Word</td>
<td>Maximum width of content allowed by GrowWindow</td>
</tr>
<tr>
<td>wScrollVer</td>
<td>Word</td>
<td>Number of pixels to scroll content vertically when user clicks arrow</td>
</tr>
<tr>
<td>wScrollHor</td>
<td>Word</td>
<td>Number of pixels to scroll content horizontally when user clicks arrow</td>
</tr>
<tr>
<td>wPageVer</td>
<td>Word</td>
<td>Number of pixels to scroll content vertically for page</td>
</tr>
<tr>
<td>wPageHor</td>
<td>Word</td>
<td>Number of pixels to scroll content horizontally for page</td>
</tr>
<tr>
<td>wInfoRefCon</td>
<td>Long</td>
<td>Value passed to draw information bar routine</td>
</tr>
<tr>
<td>wInfoHeight</td>
<td>Word</td>
<td>Height of information bar</td>
</tr>
<tr>
<td>wFrameDefProc</td>
<td>Long</td>
<td>Pointer to window definition procedure; NIL for standard</td>
</tr>
<tr>
<td>wInfoDefProc</td>
<td>Long</td>
<td>Pointer to routine that draws the interior of the information bar</td>
</tr>
<tr>
<td>wContDefProc</td>
<td>Long</td>
<td>Pointer to routine that draws the interior of the content region</td>
</tr>
<tr>
<td>wPosition</td>
<td>4 words</td>
<td>RECT specifying window's starting position and size</td>
</tr>
<tr>
<td>wPlane</td>
<td>Long</td>
<td>Window's starting plane</td>
</tr>
<tr>
<td>wStorage</td>
<td>Long</td>
<td>Pointer to memory to use for window record</td>
</tr>
</tbody>
</table>

Each of these parameters is described in more detail in the following paragraphs.

**paramLength:** Total number of bytes in parameter table, including the paramLength parameter itself. Use labels in code to calculate the values, which are used mainly for error checking. Most errors with NewWindow occur because of typing errors occurring when the parameter list is being created. The problem can be compounded if the assembler or compiler skips fields because of the typing errors but does not generate an error.

**wFrameBits:** Window frame type, as shown in Figure 25-19 (each bit flag is described in more detail after the illustration).
At the time of publication, these fields should be 111 or 000.

See text describing the alert.

Alert-type frame (no size and close boxes, no info, title, or scroll bars) = 1

No alert-type frame = 0

Vertical scroll bar = 1

No vertical scroll bar = 0

Horizontal scroll bar = 1

No horizontal scroll bar = 0

Size box = 1

No size box = 0

GrowWindow and ZoomWindow don't change origin = 1

Origin changes = 0

Zoom box on title bar = 1

No zoom box = 0

Title bar is a drag region = 1

No drag region = 0

Activate window and return wlnContent = 1

Activate window and don't return wlnContent = 0

Currently visible = 1

Window is invisible = 0

Information bar = 1

No information bar = 0

Control's state is independent of window's state = 1

Control's state is the same as window's state = 0

Record was allocated = 1

Record was allocated by application = 0

Not zoomed = 0

Not zoomed = 0

ScrollWindow window frame type

(continued)
/*
Title: If this bit is set to 1, the window has a title bar as part of the window frame.

fClose: If this bit is set to 1, the window has a close box as part of the title bar. The window must have a title bar to have a close box.

fAlert: If this flag is set to 1, it indicates to the Dialog Manager that it should draw an alert window. The fInfo, fZoom, fFlex, fGrow, fBScroll, fRScroll, fClose, and fTitle flags should all be set to 0.

fRScroll: If this bit is set to 1, the window has a right (vertical) scroll bar as part of the window frame.

fBScroll: If this bit is set to 1, the window has a bottom (horizontal) scroll bar as part of the window frame.

fGrow: If this bit is set to 1, the window has a size box as part of the window frame.

• Note: If fGrow is set to 1, fBScroll and fRScroll must also be set to 1; to have a window frame size box, you must have at least one window frame scroll bar. The fields should be either 111 or 000.

fFlex: If this bit is set to 1, the data height and width are flexible, which means that GrowWindow and ZoomWindow will not change the window's origin as needed.

fZoom: If this bit is set to 1, the window has a zoom box as part of the title bar. The window must have a title bar to have a zoom box.

fMove: If this bit is cleared to 0, the window's title bar is not considered a drag region and therefore the window cannot be moved.

fQContent: If this bit is set to 1 and there is a button-down event inside an inactive window's content, the window will be selected and a wlnContent message will be returned by TaskMaster. This feature is useful if you would like to act on any button down in the content, even if it was also used to activate the window.

fVis: If this bit is set to 1, the window is visible.

fInfo: If this bit is set to 1, wlnfoHeight and wlnfoDefProc should be given values.

fCtrlTie: When the window is inactive (unhighlighted), its controls are also considered inactive without regard for the active state of the control. Whenever an activate event is received for the window, you should redraw all of the controls for the window to make sure the controls appear in their proper states.

fAllocated: If this flag is set when CloseWindow is called, the window record will be freed. Normally you never have to set or read this flag.

fZoomed: This flag is not used if fZoom is 0.

fHilited: This flag will be set by NewWindow, so whatever value you provide will be ignored.
*/
**wTitle:** Pointer to window's title. If the window has no title bar, this value can be 0. The first byte in the string should be the length of the string followed by the ASCII characters of the title. The title string should always include a space as the first and last character of the string.

**wRefCon:** Application-defined reference value. This value is reserved for application use and can be any value.

**wZoom:** RECT data structure specifying size and position of the content region when the window is zoomed. If the bottom side of the rectangle is 0, a default RECT will be used. The default is set so the window uses the entire screen.

**wColor:** Pointer to window's color table. This is the color table used to draw the window's frame. NIL uses the default color table.

**wYOrigin:** Vertical offset of content region from data area. This value is the vertical value passed to SetOrigin when TaskMaster is used to draw inside the content region. It is also used to compute the right (or vertical) scroll bar. Set **wYOrigin** to 0 if you are not using window frame scroll bars.

**wXOrigin:** Horizontal offset of content region from data area. This value is the horizontal value passed to SetOrigin when TaskMaster is used to draw inside the content region. It is also used to compute the bottom (or horizontal) scroll bar. Set **wXOrigin** to 0 if you are not using window frame scroll bars.

**wDataH:** Height of entire data area. Used to compute the right scroll bar. Set it to 0 if you are not using window frame scroll bars.

**wDataW:** Width of entire data area. Used to compute the bottom scroll bar. Set it to 0 if you are not using window frame scroll bars.

**wMaxH:** Maximum content height allowed when growing the window. This value is passed to GrowWindow when called by TaskMaster. If set to 0, a default value will be used so the window will take up the height of the desktop. Set **wMaxH** to 0 if your window frame does not have a size box.

**wMaxW:** Maximum content width allowed when growing the window. This value is passed to GrowWindow when called by TaskMaster. If set to 0, a default value will be used so the window will take up the width of the desktop. Set **wMaxW** to 0 if your window frame does not have a size box.

**wScrollVer:** Number of pixels to scroll the content region when the up or down arrows are selected in the right scroll bar. Used only if the scroll bar is part of the frame and TaskMaster is used. Set **wScrollVer** to 0 if you are not using window frame scroll bars.

**wScrollHor:** Number of pixels to scroll the content region when the left or right arrows are selected in the bottom scroll bar. Used only if the scroll bar is part of the frame and TaskMaster is used. Set **wScrollHor** to 0 if you are not using window frame scroll bars.

(continued)
**wPageVer:** Number of pixels to scroll the content region when the up or down page regions are selected in the right scroll bar. Used only if the scroll bar is part of the frame and TaskMaster is used. Set `wPageVer` to 0 for the default value of the content region's current height minus 10.

**wPageHor:** Number of pixels to scroll the content region when the left or right page regions are selected in the bottom scroll bar. Used only if the scroll bar is part of the frame and TaskMaster is used. Set `wPageHor` to 0 for the default value of the content region's current width minus 10.

**wInfoRefCon:** Value passed to draw information bar routine. The value can be anything the application would like, such as a pointer to a string to be printed in the information bar. Set `wInfoRefCon` to 0 if you are not using an information bar.

**wInfoHeight:** Height of the information bar if `jInfo (bit 4)` of `wFrame` is set to 1.

**wFrameDefProc:** Pointer to window's definition procedure; NIL for a standard document window.

**wInfoDefProc:** Pointer to routine that will be called to draw in the information bar. Set it to 0 if you are not using window frame information bar.

**wContDefProc:** Pointer to routine that will be called to draw the window's content region. If you are using window frame scroll bars, this value must be set.

If you are not using window frame scroll bars and want to handle update events yourself, set this value to NIL.

If you are not using window frame scroll bars, but you would like TaskMaster to handle update events, set this value. The routine will be called when the content region needs to be drawn. On entry, the current port will be the window's GrafPort, the visible region will be set to the update area, and the origin set. There are no input or output parameters. Exit the routine via RTL.

**wPosition:** A RECT data structure specifying the window's starting position and size in global coordinates. The RECT becomes the port rectangle of the window's GrafPort; note, however, that the port rectangle is in local coordinates. NewWindow sets the top left corner of the port rectangle to (0,0). For the standard types of windows, this RECT data structure defines the content region of the window.

**wPlane:** Pointer to window's starting plane; that is, to the window's GrafPort behind which this window should appear—0 for bottommost, $FFFFFF0$ for topmost.

**wStorage:** Pointer to memory to use for window's record. If set to NIL, memory for the record will be allocated by the Window Manager. Because window records are not completely defined, the size needed for one is unknown and you must allow at least 325 bytes for a window record. It is usually best to have the record allocated by the Window Manager. The ability to use your own memory for a window record is provided in case you need to put up a window informing the user that there is no more memory.
$210E \textbf{PinRect}

Pins a specified point inside a specified rectangle. If the point is inside the rectangle, the point is returned; otherwise, the point associated with the nearest pixel within the rectangle is returned. (The high-order word of the pinned point is the X coordinate; the low-order word is the Y coordinate.)

More precisely, for a specified rectangle \((\text{left}, \text{top}, \text{right}, \text{bottom})\) and a specified point \((h, v)\), PinRect does the following:
- If \(h < \text{left}\), it returns \text{left}.
- If \(v < \text{top}\), it returns \text{top}.
- If \(h > \text{right}\), it returns \text{right} -1.
- If \(v > \text{bottom}\), it returns \text{bottom} -1.

\textbf{Note:} The 1 is subtracted when the specified point is below or to the right of the specified rectangle so that a pixel drawn at that point will lie within the \text{Rect}.

\textbf{Parameters}

\textbf{Stack before call}

\begin{verbatim}
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>- longspace</td>
</tr>
<tr>
<td>- theXPt</td>
</tr>
<tr>
<td>- theYPt</td>
</tr>
<tr>
<td>- theRectPtr</td>
</tr>
</tbody>
</table>
\end{verbatim}

- \text{Long}—Space for result
- \text{Word}—X coordinate of point to be pinned
- \text{Word}—Y coordinate of point to be pinned
- \text{Long}—POINTER to \text{RECT} data structure defining boundary of point

\text{← SP}

\textbf{Stack after call}

\begin{verbatim}
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>- pinnedPt</td>
</tr>
</tbody>
</table>
\end{verbatim}

- \text{Long}—POINT; high word = X coordinate, low word = Y coordinate

\text{← SP}
Errors  
None

C

extern pascal Long PinRect(theXPt, theYPt, theRectPtr)
Integer theXPt;
Integer theYPt;
RectPointer theRectPtr;

You can also use the following alternate form of the call:
extern pascal Long PinRect(thePoint, theRectPtr)
Point thePoint;
RectPointer theRectPtr;

◆ Note: C Pascal-type functions do not deal properly with data structures returned on the stack. The Long result returned by this call can be passed to any calls requiring a point as a parameter. You cannot use the C dot operator to access the individual Y and X coordinates within the value returned by this call.
RefreshDesktop

Redraws the entire desktop and all the windows. This routine can be useful when the entire screen is clobbered by some application-specific, non-Window Manager operation.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>redrawRect</td>
</tr>
</tbody>
</table>

Long—POINTER to RECT of rectangle to redraw; NIL for entire screen

Stack after call

| previous contents |

Errors

None

C

extern pascal void RefreshDesktop(redrawRect)

Rect *redrawRect;
SelectWindow

Makes a specified window the active window. This routine unhighlights the previously
active window, brings the specified window in front of all other windows, highlights it,
and generates appropriate activate events. Call this routine if you are not using
TaskMaster and there's a mouse-down event in the content region of an inactive
window.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to window's GrafPort

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>

Errors

None

C

extern pascal void SelectWindow(theWindowPtr)

GrafPortPtr theWindowPtr;
SendBehind

Changes the position of a specified window, redrawing any exposed windows.

If `behindWindowPtr` is -2 ($FFFFFFFE), it sends the specified window behind all other windows. If `behindWindowPtr` is -1 ($FFFFFFFD), it puts the specified window in front of all other windows. If the specified window is the active window, the routine unhilights the active window, highlights the new active window, and generates the appropriate activate events.

Parameters

Stack before call

```
previous contents
--behindWindowPtr Long—POINTER to GrafPort or $FFFFFFDF = top or $FFFFFFFE = bottom
-- theWindowPtr Long—POINTER to window's GrafPort
```

Stack after call

```
previous contents ← SP
```

Errors

None

C

```c
extern pascal void SendBehind(behindWindowPtr, theWindowPtr)
GrafPortPtr behindWindowPtr;
GrafPortPtr theWindowPtr;
```
**SetContentDraw**

Sets the pointer to the routine to draw the content region of a specified window.

TaskMaster calls this routine when it gets an update event for that window. See the section "Draw Content Routine" in this chapter for more information about the draw routine.

**Parameters**

**Stack before call**

- **previous contents**
  - **contentProcPtr** — Long—POINTER to routine to draw content region
  - **theWindowPtr** — Long—POINTER to window's GrafPort

**Stack after call**

- **previous contents**
  ← SP

**Errors**

None

**C**

```c
extern pascal void SetContentDraw(contentDrawPtr, theWindowPtr)
VoidProcPtr contentDrawPtr;
GrafPortPtr theWindowPtr;
```
SetContentOrigin

Sets the origin of the window's GrafPort when handling an update event. The values are used by TaskMaster to set the origin of the window's GrafPort and are also used by the Window Manager to compute scroll bars in the window frame. See the section "Origin Movement" in this chapter for an illustration of the origin values.

Setting the origin values generates an update event.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>xOrigin</td>
</tr>
<tr>
<td>yOrigin</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

Word—Content region's horizontal offset into data area

Word—Content region's vertical offset into data area

Long—POINTER to window's GrafPort

Stack after call

| previous contents |

<- SP

Errors

None

C

extern pascal void SetContentOrigin(xOrigin,yOrigin,theWindowPtr)

Word   xOrigin;
Word   yOrigin;

GrafPortPtr   theWindowPtr;

Window Manager routines 25-95
$570E \textbf{SetContentOrigin2} \\

Sets the origin of the window's GrafPort when handling an update event and allows the application to scroll or not scroll the window's content region. If \textit{scrollFlag} is set to 1, this call is the same as SetContentOrigin. If \textit{scrollFlag} is set to 0, the window's origin will be set without the Window Manager scrolling the data in the window. This feature is useful, for example, if a window's data area has to be expanded to the left or above the current origin and you don't want your application to redraw everything in the window.

The \textit{xOrigin} and \textit{yOrigin} values are used by TaskMaster to set the origin of the window's GrafPort and are also used by the Window Manager to compute scroll bars in the window frame. See the section "Origin Movement" in this chapter for an illustration of the origin values.

Setting the origin values generates an update event.

\textbf{Parameters}

\textbf{Stack before call}

\begin{tabular}{|c|}
\hline
\textit{previous contents} \\
\hline
\textit{scrollFlag} \\
\textit{xOrigin} \\
\textit{yOrigin} \\
\hline
\textit{theWindowPtr} \\
\hline
\end{tabular}

- \textbf{Word}—0 to not scroll content, 1 to scroll content
- \textbf{Word}—Content region's horizontal offset into data area
- \textbf{Word}—Content region's vertical offset into data area
- \textbf{Long}—POINTER to window's GrafPort

\leftarrow \textit{SP}

\textbf{Stack after call}

\begin{tabular}{|c|}
\hline
\textit{previous contents} \\
\hline
\leftarrow \textit{SP}
\end{tabular}

\textbf{Errors} \hspace{1cm} None

\textbf{C} \hspace{1cm}

\texttt{extern pascal void SetContentOrigin2(\textit{scrollFlag,,xOrigin,,yOrigin, theWindowPtr})}

\texttt{Word \hspace{0.5cm} \textit{scrollFlag};}

\texttt{Word \hspace{0.5cm} \textit{xOrigin};}

\texttt{Word \hspace{0.5cm} \textit{yOrigin};}

\texttt{GrafPortPtr \hspace{0.5cm} \textit{theWindowPtr};}

25-96 \hspace{1cm} Window Manager routines
**SetDataSize**

Sets the height and width of the data area of a specified window. Setting these values will not change the scroll bars or generate update events.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataWidth</td>
</tr>
<tr>
<td>dataHeight</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

---

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

---

**Errors**

None

**C**

```c
extern pascal void SetDataSize(dataWidth, dataHeight, theWindowPtr)

Word    dataWidth;
Word    dataHeight;
GrafPortPtr    theWindowPtr;
```

---

Window Manager routines 25-97
SetDefProc

Sets the pointer to the routine that defines a window's frame and behavior. See the section "Defining Your Own Windows" in this chapter for an explanation of what a definition procedure does.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wDefProcPtr</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Errors

None

C

extern pascal void SetDefProc(wDefProcPtr, theWindowPtr)

LongProcPtr wDefProcPtr;
GrafPortPtr theWindowPtr;
SetFrameColor

Sets the color of a specified window's frame. Does not redraw the window. Do a HideWindow call before the SetFrameColor call and a ShowWindow call after the SetFrameColor call to redraw the window in its new colors.

The interaction between the `newColorPtr` and `theWindowPtr` parameters is shown in Figure 25-20. See the section "Window Frame Colors and Patterns" in this chapter for a definition of the color table.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>newColorPtr</code></td>
</tr>
<tr>
<td><code>theWindowPtr</code></td>
</tr>
</tbody>
</table>

Long—POINTER to five-word pattern/color table; NIL for default table

Long—POINTER to window's GrafPort; NIL for window default

← SP

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

None

**C**

```
extern pascal void SetFrameColor(newColorPtr, theWindowPtr)
```

WindColorPtr      newColorPtr;
GrafPortPtr       theWindowPtr;

(continued)
The newColorPtr and theWindowPtr parameters

The precise results of the four possible combinations of the `newColorPtr` and `theWindowPtr` parameters are shown in Figure 25-20.

<table>
<thead>
<tr>
<th>SetFrameColor parameters</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>newColorPtr</code> <code>theWindowPtr</code></td>
<td>Changes a specified window to a specified color table</td>
</tr>
<tr>
<td>Pointer Pointer</td>
<td>Makes a specified color table the default table for all future windows</td>
</tr>
<tr>
<td>Pointer NIL</td>
<td>Changes the specified window to the Window Manager’s startup color table</td>
</tr>
<tr>
<td>NIL Pointer</td>
<td>Replaces the default color table for all future windows with the Window Manager’s startup color table</td>
</tr>
</tbody>
</table>

Figure 25-20

SetFrameColor newColorPtr and theWindowPtr values

25-100 Window Manager routines
$160E \textbf{SetInfoDraw} \newline
Sets the pointer to a routine that draws the information bar for a specified window. If the window has an information bar, the standard window definition procedure calls this routine whenever the window's frame needs to be drawn. See the section "Draw Information Bar Routine" in this chapter for more information about the draw routine.

\textbf{Parameters}\newline

\textbf{Stack before call}\newline
\begin{center}
\begin{tabular}{c|c}
\textit{previous contents} & \textit{infoDrawPtr} \\
\textit{theWindowPtr} & \textit{SP} \\
\end{tabular}
\end{center}

\begin{itemize}
\item \textbf{infoDrawPtr} \texttt{Long—POINTER to draw information bar routine}
\item \textbf{theWindowPtr} \texttt{Long—POINTER to window's GrafPort}
\end{itemize}

\textbf{Stack after call}\newline
\begin{center}
\begin{tabular}{c|c}
\textit{previous contents} & \textit{SP} \\
\end{tabular}
\end{center}

\textbf{Errors} \newline
None

\textbf{C}\newline
\begin{verbatim}
extern pascal void SetInfoDraw(infoDrawPtr, theWindowPtr)
VoidProcPtr   infoDrawPtr;
GrafPortPtr   theWindowPtr;
\end{verbatim}
SetInfoRefCon

Sets the value associated with the draw information bar routine for a specified window. That value is reserved for use by the application.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image-url" alt="Stack Diagram" /></td>
</tr>
</tbody>
</table>

- infoRefCon: Long—Value to pass to draw information bar routine
- theWindowPtr: Long—POINTER to window’s GrafPort

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image-url" alt="Stack Diagram" /></td>
</tr>
</tbody>
</table>

Errors

None

C

```c
extern pascal void SetInfoRefCon(infoRefCon, theWindowPtr)
LongWord   infoRefCon;
GrafPortPtr theWindowPtr;
```
$430E  

**SetMaxGrow**

Sets the maximum values to which a specified window's content region can grow.

**Parameters**

**Stack before call**

- `previous contents`
- `maxWidth`
- `maxHeight`
- `theWindowPtr`

  - `Word`—Maximum content width in pixels
  - `Word`—Maximum content height in pixels
  - `Long`—POINTER to window's GrafPort

  `← SP`

**Stack after call**

- `previous contents`

  `← SP`

**Errors**

None

**C**

```c
extern pascal void SetMaxGrow(maxWidth, maxHeight, theWindowPtr);
Word    maxWidth;
Word    maxHeight;
GrafPortPtr    theWindowPtr;
```
SetOriginMask

Specifies the mask used to put the horizontal origin on a grid.

SetOriginMask is useful when you are using a scrollable window in 640 mode with dithered colors. In that mode, pixels must keep the same horizontal position to remain the same color. Scrolling windows can change the color by putting the pixels in the wrong horizontal position. SetOriginMask prevents this problem by providing an originMask that will be ANDed by TaskMaster with any new horizontal origin to force the origin to certain boundaries. The default is $FFFF, single pixel.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Word—Mask used to place horizontal origin on a grid</th>
<th>originMask</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Long—POINTER to window’s GrafPort</td>
<td>theWindowPtr</td>
</tr>
<tr>
<td></td>
<td>SP</td>
<td></td>
</tr>
</tbody>
</table>

Stack after call

| previous contents | SP |

Errors

None

C

extern pascal void SetOriginMask(originMask, theWindowPtr)

Word originMask;

GrafPortPtr theWindowPtr;
$470E \textbf{SetPage}

Sets the number of pixels by which TaskMaster will scroll the content region when the user selects the page regions on window frame scroll bars.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>hPage</td>
</tr>
<tr>
<td>vPage</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

Word—Number of pixels to page horizontally
Word—Number of pixels to page vertically
Long—POINTER to window's GrafPort

| ← SP |

**Stack after call**

| previous contents |
| ← SP |

**Errors**

None

**C**

```c
extern pascal void SetPage(hPage, vPage, theWindowPtr)

Word hPage;
Word vPage;
GrafPortPtr theWindowPtr;
```
### $450E$ SetScroll

Sets the number of pixels by which TaskMaster will scroll the content region when the user selects the arrows on window frame scroll bars.

#### Parameters

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Word—Number of pixels to scroll horizontally</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Word—Number of pixels to scroll vertically</td>
</tr>
<tr>
<td></td>
<td>Long—POINTER to window’s GrafPort</td>
</tr>
<tr>
<td></td>
<td>← SP</td>
</tr>
</tbody>
</table>

**Stack after call**

| previous contents   | ← SP                                         |

#### Errors

None

#### C

```c
extern pascal void SetScroll(hScroll,vScroll, theWindowPtr)

Word   hScroll;
Word   vScroll;
GrafPortPtr   theWindowPtr;
```
$4B0E  SetSysWindow

Marks a specified window as a system window.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- theWindowPtr --</td>
</tr>
<tr>
<td>Long--POINTER to window's GrafPort</td>
</tr>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

| previous contents |
|← SP |

Errors

None

C

extern pascal void SetSysWindow(theWindowPtr)
GrafPortPtr theWindowPtr;
$2D0E \textbf{SetWFrame}

Sets the bit flag that describes a specified window's frame type. The window frame is not redrawn. See the discussion of \textit{wFrameBits} in the section "NewWindow" in this chapter for the definition of the bits of the \textit{wFrame} parameter.

\begin{itemize}
  \item \textbf{Note:} Normally, you won't need to call this routine; instead, you should set up the window frame correctly with the \textit{NewWindow} routine. The \textit{SetWFrame} routine is provided for custom window definition procedures.
\end{itemize}

\textbf{Parameters}

\textbf{Stack before call}

\begin{tabular}{|c|}
  \hline
  \textit{previous contents} \\
  \textit{wFrame} \\
  \textit{theWindowPtr} \\
  \hline
\end{tabular}

\textbf{Word}—Bit flag specifying window's frame type

\textbf{Long}—POINTER to window's GrafPort

\textbf{Stack after call}

\begin{tabular}{|c|}
  \hline
  \textit{previous contents} \\
  \hline
\end{tabular}

\leftarrow \textbf{SP}

\textbf{Errors} \hspace{1cm} \textbf{None}

\textbf{C}

\begin{verbatim}
extern pascal void SetWFrame(wFrame, theWindowPtr)
Word wFrame;
GrafPortPtr theWindowPtr;
\end{verbatim}

\begin{flushright}
25-108 \hspace{1cm} \textbf{Window Manager routines}
\end{flushright}
SetWindowIcons

Sets the icon font for the Window Manager. See the section "Window Manager Icon Font" in this chapter for more information about the font.

If you want to use the call to simply retrieve the handle of the current font, specify a negative value for newFontHandle.

Parameters

Stack before call

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>
| -- longspace --   | Long—Space for result
| -- newFontHandle  | Long—HANDLE to new icon font; negative to not replace font
```

Stack after call

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>
| -- oldFontHandle  | Long—HANDLE to icon font before call
```

Errors

None

C

```c
extern pascal FontHndl SetWindowIcons(newFontHandle)
FontHndl newFontHandle;
```
SetWRefCon

Sets a value that is inside a specified window record and is reserved for the application's use.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-- wRefCon --</strong></td>
</tr>
<tr>
<td><strong>-- theWindowPtr --</strong></td>
</tr>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>← SP</td>
</tr>
</tbody>
</table>

**Errors**

None

**C**

```c
extern pascal void SetWRefCon(wRefCon, theWindowPtr)
Longint    wRefCon;
GrafPortPtr theWindowPtr;
```
$0DOE SetWTitle

Changes the title of a specified window to a specified title and redraws the window. The string pointed to by titlePtr must be a Pascal-type string.

Warning
The string pointed to by titlePtr must not be changed or moved.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>titlePtr</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Errors
None

C

extern pascal void SetWTitle(titlePtr, theWindowPtr)

Pointer titlePtr;
GrafPortPtr theWindowPtr;
SetZoomRect

Sets the rectangle to be used as the content's zoomed or unzoomed size for a specified window. If the window is currently in a zoomed state (that is, bit 2, the \texttt{fZoomed} bit, is set to 1 in the \texttt{wFrame} flag—see the section “NewWindow” in this chapter), \texttt{wZoomSizePtr} should point to a RECT data structure that specifies the window's unzoomed size and position.

If the window is currently in an unzoomed state (that is, the \texttt{fZoomed} bit is set to 0), \texttt{wZoomSizePtr} should point to a RECT data structure that specifies the zoomed size and position of the window. The rectangle will be used as the window's content region (port) the next time the window is zoomed by a call to \texttt{ZoomWindow}.

**Parameters**

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{wZoomSizePtr} -- Long--POINTER to rectangle to be used as content's zoomed size</td>
</tr>
<tr>
<td>\texttt{theWindowPtr} -- Long--POINTER to window's GrafPort</td>
</tr>
<tr>
<td>\texttt{SP}</td>
</tr>
</tbody>
</table>

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{SP}</td>
</tr>
</tbody>
</table>

**Errors**

None

**C**

```c
extern pascal void SetZoomRect(wZoomSizePtr, theWindowPtr)

Rect *wZoomSizePtr;
GrafPortPtr theWindowPtr;
```
$230E  

**ShowHide**

Shows or hides a window. If \textit{showFlag} is \textbf{TRUE}, ShowHide makes the specified window visible if it’s not already visible and has no effect if it is already visible. If \textit{showFlag} is \textbf{FALSE}, ShowHide makes the window invisible if it’s not already invisible and has no effect if it is already invisible.

Unlike HideWindow and ShowWindow, ShowHide never generates activate events or changes the highlighting or front-to-back ordering of windows.

\begin{center}
\textbf{Important}

Use this procedure carefully and only in special circumstances in which you need more control than is allowed by ShowWindow and HideWindow. You could end up with an active window that isn’t highlighted.
\end{center}

### Parameters

#### Stack before call

- \textbf{previous contents}
- \textbf{showFlag} \text{Word—BOOLEAN; TRUE to show, FALSE to hide}
- \textbf{theWindowPtr} \text{Long—POINTER to window's GrafPort}

\begin{center}
\textbf{← SP}
\end{center}

#### Stack after call

- \textbf{previous contents}

\begin{center}
\textbf{← SP}
\end{center}

### Errors

None

\begin{verbatim}
extern pascal void ShowHide(showFlag,theWindowPtr)

Boolean    showFlag;
GrafPortPtr theWindowPtr;
\end{verbatim}

Window Manager routines 25-113
ShowWindow

Makes a specified window visible if it was invisible and then draws the window. It does not change the front-to-back ordering of the windows. If you have previously hidden the frontmost window with HideWindow, HideWindow will have brought the window behind it to the front. If you then do a ShowWindow of the window you hid, it will no longer be frontmost.

If the specified window is already visible, ShowWindow has no effect.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>
| Long—POINTER to window's GrafPort | SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>
|                     | SP

Errors

None

C

extern pascal void ShowWindow(theWindowPtr)

GrafPortPtr theWindowPtr;

25-114 Window Manager routines
SizeWindow

Enlarges or shrinks the port rectangle of the specified window's GrafPort to a specified width and height. If the new width and height are specified as 0, SizeWindow does nothing. The window's position on the screen does not change. When the new window frame is drawn, if the width of a document window changes, the title is recentered in the title bar, or if it no longer fits, it is truncated.

Parameters

Stack before call

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>newWidth</td>
</tr>
<tr>
<td>newHeight</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>
```

Word—New width of window in pixels

Word—New height of window in pixels

Long—POINTER to window's GrafPort

← SP

Stack after call

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>
```

← SP

Errors

None

C

```
extern pascal void SizeWindow(newWidth, newHeight, theWindowPtr)

Word       newWidth;
Word       newHeight;
GrafPortPtr theWindowPtr;
```

Window Manager routines 25-115
**StartDrawing**

Makes a specified window the current port and sets its origin. After the call, any drawing occurs inside the specified window's content area and in the proper coordinate system.

---

**Important**

Do not call StartDrawing between a BeginUpdate and an EndUpdate call. Also, when you have finished drawing, call the QuickDraw II routine SetOrigin(0,0).

StartDrawing can be used for drawing in a window's content region outside of update events.

*Note:* StartDrawing is useful only with standard document windows with frame scroll bars. Otherwise, only a SetPort call is needed to make the correct port current.

**Parameters**

**Stack before call**

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>
```

Long—POINTER to window’s GrafPort

---

**Stack after call**

```
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
</table>
```

---

**Errors**

None

**C**

```c
extern pascal void StartDrawing(theWindowPtr)
GrafPortPtr theWindowPtr;
```
StartInfoDrawing

Allows an application to draw or hit test outside of its information bar definition procedure. If there is no information in the window, the coordinates of the RECT data structure pointed to by infoRectPtr will all be 0. The coordinate system will be local to the window's frame; that is, 0,0 will be the upper left corner of the window, and the current GrafPort will be the Window Manager's.

You can set the clip region after a StartInfoDrawing call and before an EndInfoDrawing call; this is not true from within the information bar definition procedure.

Important
When you finish dealing with the information bar, you must call the EndInfoDrawing routine before you make any other calls to the Window Manager.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>infoRectPtr</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to RECT where information bar's data will be stored
Long—POINTER to window's GrafPort

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

← SP

Errors
None

C

extern pascal void StartInfoDrawing(infoRectPtr,theWindowPtr)

Rect *infoRectPtr;
GrafPortPtr theWindowPtr;
Calls GetNextEvent and looks in the event part of the task record to see if it can handle
the event. If no event is returned by GetNextEvent, 0 is returned in taskCode. For
further description of TaskMaster's activities, see the following section "TaskMaster
Pseudocode."

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
<tr>
<td>taskMask</td>
</tr>
<tr>
<td>taskRecPtr</td>
</tr>
</tbody>
</table>

  | Word—Space for result
  | Word—Mask used by TaskMaster to call GetNextEvent
  | Long—POINTER to Window Manager task record
  | ← SP

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskCode</td>
</tr>
</tbody>
</table>

  | Word—Code indicating action to be taken; 0 = no further task to perform
  | ← SP

Errors

$0E03 taskMaskErr Reserved bits not clear in wmTaskMask field of task record

C

extern pascal Word TaskMaster(taskMask, taskRecPtr)

Word taskMask;
WmTaskRecPtr taskRecPtr;
TaskMaster pseudocode

TaskMaster can be a powerful tool, and can take care of much of the mundane processing of events. Because the routine is so important, this section presents pseudocode describing TaskMaster's operations.

To perform its functions, TaskMaster takes the following steps:

Calls the Desk Manager routine SystemTask.

Calls the Event Manager routine GetNextEvent with a TaskRec and eventMask.

The wmMessage field of TaskRec is duplicated into the wmTaskData field of TaskRec.

If any of the reserved bits in the wmTaskMask field are not 0:

\[
\{ \\
\text{Low word of } \text{wmTaskData} = 0. \\
\text{Returns nullEvt ($0000$).} \\
\text{Error returned: wmTaskMaskErr ($0E03$).} \\
\}
\]

If wmWhat field of TaskRec = nullEvt ($0000$):

\[
\{ \\
\text{Low word of TaskData} = 0. \\
\text{Returns nullEvt ($0000$).} \\
\}
\]

If wmWhat field of TaskRec = updateEvt ($0006$):

\[
\{ \\
\text{If } \text{wmTaskMask} \text{ bit } \text{tmUpdate} (\text{bit 1}) = 0: \\
\{ \\
\text{wmTaskData} = \text{pointer to window to be updated}. \\
\text{Returns updateEvt ($0006$).} \\
\} \\
\text{If window's } \text{wContDefProc} \text{ field = 0:} \\
\{ \\
\text{wmTaskData} = \text{pointer to window to be updated}. \\
\text{Returns updateEvt ($0006$).} \\
\}
\}
\]

Calls the BeginUpdate routine.

The window's draw routine in window's wContDefProc field is called (routine in application).

Calls the EndUpdate routine.

\[
\text{wmTaskData} \text{ low word = updateEvt ($0006$).} \\
\text{Returns nullEvt ($0000$).}
\]

(continued)
If `wmWhat` field of TaskRec = `activateEvt ($0008)`:
{
    If `wmTaskMask` bit `tmCRedraw` (bit 13) = 1:
    {
        If `wframe` bit `fCHTie` (bit 3) = 1:
        {
            Calls the Control Manager routine `DrawControls` to draw controls in proper state.
        }
    }
    `wmTaskData` = pointer to window that was activated or deactivated (check modifier field).
    Returns `activateEvt ($0008)`.
}

If `wmWhat` field of TaskRec = `keyDownEvt ($0003)` OR `autoKeyEvt ($0005)`:
{
    If `wmTaskMask` bit `tmMenuKey` (bit 0) = 0:
    {
        `wmTaskData` = message field as returned by `GetNextEvent`.
        Returns `keyDownEvt ($0003)`.
    }
    Calls the Menu Manager routine `MenuKey` with the given TaskRec for the system menu bar.
    Go to Menu Selection.
    (The remainder of TaskMaster from this point is the same as when the Menu Manager routine `MenuSelect` is called.)
}

If `wmWhat` field of TaskRec does not equal `mouseDownEvt ($0001)`:
{
    Returns `what` field from TaskRec.
}
If \texttt{wmWhat} field of TaskRec = \texttt{mouseDownEvt($0001$)}:
{
    If TaskMask bit \texttt{tmFindW} (bit 2) = 0:
    {
        \texttt{wmTaskData} = message field from GetNextEvent.
        \texttt{wmTaskData} = \texttt{mouseDownEvt ($0001$)}.
    }
    Calls FindWindow.
    If FindWindow returns \texttt{wInMenuBar ($0011$)}:
    {
        If TaskMask \texttt{tmMenuSel} (bit 3) = 0:
        {
            Low word of \texttt{wmTaskData} = 0.
            \texttt{wInMenuBar ($0011$)}.
        }
        \texttt{MenuSelect} is called with TaskRec passed to TaskMaster.
    }
    \textbf{Menu Selection:}
    If low word of \texttt{wmTaskData} = 0, then no selection made:
    {
        If \texttt{wmTaskMask} bit \texttt{tmInactive} (bit 14) = 0:
        {
            Low word of \texttt{wmTaskData} = \texttt{wInMenuBar ($0011$)}.
            \texttt{nullEvt ($0000$)}.
        }
        If high word of \texttt{wmTaskData} = \text{nonzero}:
        {
            Low word of \texttt{wmTaskData} = 0.
            High word of \texttt{wmTaskData} = ID of selected inactive
            \text{menu item}.
            \texttt{wInactMenu ($001C$)}.
        }
    }
    \texttt{wInMenuBar ($0011$)}.
    \texttt{nullEvt ($0000$)}.

    \textit{(continued)}
Else, menu selection made:
{
    If low word of \texttt{wmTaskData} (menu item ID) < 250:
    {
        If \texttt{wmTaskMask} bit \texttt{tmOpenNDA} (bit 4) = 0:
        {
            Low word of \texttt{wmTaskData} = ID of selected menu item.
            High word of \texttt{wmTaskData} = ID of menu from which selection was made.
            Returns \texttt{winDeskItem} ($001A$).
        }
        Calls the Desk Manager routine \texttt{OpenNDA} to open the desk accessory selected.
        Calls Menu Manager routine \texttt{HiliteMenu} to unhighlight the selected menu.
        Low word of \texttt{wmTaskData} = \texttt{winDeskItem} ($001A$).
        Returns \texttt{nullEvt} ($0000$).
    }
    If \texttt{TaskMask} bit \texttt{tmSpecial} (bit 12) = 0:
    {
        Low word of \texttt{wmTaskData} = ID of selected menu item.
        High word of \texttt{wmTaskData} = ID of menu from which selection was made.
        Returns \texttt{winSpecial} ($0019$).
    }
    If top window is an application (nonsystem) window:
    {
        Low word of \texttt{wmTaskData} = ID of selected menu item.
        High word of \texttt{wmTaskData} = ID of menu from which selection was made.
        Returns \texttt{winSpecial} ($0019$).
    }
    If low word of \texttt{wmTaskData} (menu item ID) = 255 (Close item):
    {
        Calls Desk Manager routine \texttt{CloseNDAbyWinPtr} for top window (system window).
        Calls \texttt{HiliteMenu} to unhighlight the selected menu.
        Low word of \texttt{wmTaskData} = \texttt{closedNDA} ($001D$).
        Returns \texttt{nullEvt} ($0000$).
    }
}

25-122 Window Manager routines
If low word of `wmTaskData` (menu item ID) = 250, 251, 252, 253, or 254 (edit item):

```
{ 
  Calls Desk Manager routine SystemEdit with ID of special edit menu item.
  If SystemEdit returns FALSE:
  { 
    Low word of `wmTaskData` = ID of selected menu item.
    High word of `wmTaskData` = ID of menu from which selection was made.
    Returns winSpecial ($0019).
  }
  (Top system window handled the special menu item selection.)
  Calls the Menu Manager routine HiliteMenu to unhighlight the selected menu.
  Low word of TaskData = wCalledSysEdit ($001E).
  Returns nullEvt ($0000).
}
```

(Top system window handled the special menu item selection.)
Calls the Menu Manager routine HiliteMenu to unhighlight the selected menu.

```
Low word of `wmTaskData` (menu item ID) > 255.
Low word of `wmTaskData` = ID of selected menu item.
High word of `wmTaskData` = ID of menu from which selection was made.
Returns winMenuBar ($0011).
}
```

(End menu selection)

If FindWindow returns a negative value:

```
{ 
  If `wmTaskMask` bit tmSysClick (bit 5) = 0:
  { 
    `wmTaskData` = window pointer returned from FindWindow.
    Returns result from FindWindow.
  }
  Calls Desk Manager routine SystemClick with result from FindWindow.
  `wmTaskData` low word = wClickCalled ($0012).
  Returns nullEvt ($0000).
}
```

(Continued)

Window Manager routines 25-123
If FindWindow returns winDrag (0014):
{
    If \text{wmTaskMask} \text{ bit } tmDragW \text{ (bit 6)} = 0:
    \{ 
        \text{wmTaskData} = \text{window pointer returned from FindWindow.}
        \text{Returns winDrag (0014).}
    \}
    \text{If bit 8 in the modifier field of TaskRec (Apple key up) and the}
    \text{window is not active:}
    \{
        \text{Calls SelectWindow to make window active.}
    \}
    \text{Calls DragWindow.}
    \text{wmTaskData} = \text{winDrag (0014).}
    \text{Returns nullEvt (0000).}
}

If FindWindow returns winContent (0013):
{
    If \text{wmTaskMask} \text{ bit } tmContent \text{ (bit 7)} = 0:
    \{ 
        \text{wmTaskData} = \text{window pointer returned from FindWindow.}
        \text{Returns winContent (0013).}
    \}
    \text{If the window is not active:}
    \{
        \text{Calls SelectWindow to make window active.}
    \}
    \text{If wFrame field fQContent \text{ (bit 6)} = 1:}
    \{ 
        \text{wmTaskData} = \text{window pointer returned from FindWindow.}
        \text{Returns winContent (0013).}
    \}
    \text{Low word of \text{wmTaskData} = winContent (0013).}
    \text{Returns nullEvt (0000).}
}
If FindWindow returns wInGoAway ($0016):
{
    If \textit{wmTaskMask} bit \textit{tmClose} (bit 8) = 0:
    {
        \textit{wmTaskData} = window pointer returned from FindWindow.
        \textbf{Returns} wInGoAway ($0016$).
    }
    \textbf{Calls} TrackGoAway.
    \textbf{If} TrackGoAway returns \textbf{TRUE}:
    {
        \textit{wmTaskData} = window pointer returned from FindWindow.
        \textbf{Returns} wInGoAway ($0016$).
    }
    \textbf{Low word of} \textit{wmTaskData} = wInGoAway ($0016$).
    \textbf{Returns} nullEvt ($0000$).
}

If FindWindow returns wInZoom ($0017$):
{
    If \textit{wmTaskMask} bit \textit{tmZoom} (bit 9) = 0:
    {
        \textit{wmTaskData} = window pointer returned from FindWindow.
        \textbf{Returns} wInZoom ($0017$).
    }
    \textbf{Calls} TrackZoom.
    \textbf{If} TrackZoom returns \textbf{TRUE}:
    {
        \textbf{Calls} ZoomWindow.
        \textbf{Low word of} \textit{wmTaskData} = wInZoom ($0017$).
        \textbf{Returns} nullEvt ($0000$).
    }
    \textbf{Low word of} \textit{wmTaskData} = wTrackZoom ($001F$).
    \textbf{Returns} nullEvt ($0000$).
}

If FindWindow returns wInGrow ($0015$):
{
    If \textit{wmTaskMask} bit \textit{tmGrow} (bit 10) = 0:
    {
        \textit{wmTaskData} = window pointer returned from FindWindow.
        \textbf{Returns} wInGrow ($0015$).
    }
    \textbf{Calls} GrowWindow.
    \textbf{Calls} SizeWindow with results from GrowWindow.
    \textbf{Low word of} \textit{wmTaskData} = wInGrow ($0015$).
    \textbf{Returns} nullEvt ($0000$).
} 

\textbf{(continued)}
If FindWindow returns winInfo ($0018):
{
    If wmTaskMask bit tmInfo (bit 15) = 1:
    {
        If window is not active:
        {
            Calls SelectWindow.
            Low word of \texttt{wmTaskData} = \texttt{winInfo ($0018)}.
            Returns nullEvt ($0000).
        }
    }
    \texttt{wmTaskData} = window pointer returned from FindWindow.
    Returns \texttt{winInfo ($0018)}.
}

If FindWindow returns winFrame ($001B):
{
    If wmTaskMask bit tmScroll (bit 11) = 0:
    {
        \texttt{wmTaskData} = window pointer returned from FindWindow.
        Returns \texttt{winFrame ($001B)}.
    }
    If window is not active:
    {
        Calls SelectWindow to make active.
        Low word of \texttt{wmTaskData} = \texttt{wHitFrame ($0020)}.
        Returns nullEvt ($0000).
    }
    If button was on a window frame control:
    {
        Low word of \texttt{wmTaskData} = \texttt{wHitFrame ($0020)}.
        Returns nullEvt ($0000).
    }
    Calls TrackControl with an action procedure within TaskMaster.
    The action procedure in TrackMaster performs scrolling and
    updates.
    Low word of \texttt{wmTaskData} = \texttt{winFrame ($001B)}.
    Returns nullEvt ($0000).
}

Else (something returned from FindWindow other than those handled
above):
{
    \texttt{wmTaskData} = returned value from FindWindow.
    Returns result from FindWindow.
}

25-126 Window Manager routines
$180E  TrackGoAway

Tracks the mouse until the mouse button is released, highlighting the go-away region as long as the mouse location remains inside it and unhighlighting it when the mouse moves outside it.

When there's a mouse-down event in the go-away region of the specified window and the application is not using TaskMaster, the application should call TrackGoAway with $startX$ and $startY$ equal to the point where the mouse button was pressed (in global coordinates, as stored in the $where$ field of the event record).

The exact way a window's go-away region is highlighted depends on its window definition procedure. If the user releases the mouse button while the cursor is inside the go-away region, TrackGoAway unhighlights the go-away region and returns TRUE (following which the application should eventually perform a CloseWindow). If the user releases the mouse button while the cursor is outside the go-away region, TrackGoAway returns FALSE (in which case the application should do nothing).

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$wordspace$</td>
</tr>
<tr>
<td>$startX$</td>
</tr>
<tr>
<td>$startY$</td>
</tr>
<tr>
<td>$theWindowPtr$</td>
</tr>
</tbody>
</table>

Word—Space for result
Word—Starting X coordinate of cursor, in global coordinates
Word—Starting Y coordinate of cursor, in global coordinates
Long—POINTER to window's GrafPort

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$goAway$</td>
</tr>
</tbody>
</table>

Word—BOOLEAN; TRUE if go-away selected when button released, FALSE if not

Errors
None
extern pascal Boolean TrackGoAway(startX,startY,theWindowPtr)
Integer startX;
Integer startY;
GrafPortPtr theWindowPtr;
You can also use the following alternate form of the call:

extern pascal Boolean TrackGoAway(start,theWindowPtr)
Point start;
GrafPortPtr theWindowPtr;
$260E  TrackZoom

Tracks the mouse until the mouse button is released, highlighting the zoom region as long as the mouse location remains inside it and unhighlighting it when the mouse moves outside it.

When there's a mouse-down event in the zoom region of the specified window and the application is not using TaskMaster, the application should call TrackZoom with startX and startY equal to the point where the mouse button was pressed (in global coordinates, as stored in the where field of the event record).

The exact way a window's zoom region is highlighted depends on its window definition procedure. If the mouse button is released inside the zoom region, TrackZoom unhighlights the zoom region and returns TRUE (following which the application should eventually perform a ZoomWindow). If the mouse button is released outside the zoom region, TrackZoom returns FALSE (in which case the application should do nothing).

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordSpace</td>
</tr>
<tr>
<td>startX</td>
</tr>
<tr>
<td>startY</td>
</tr>
<tr>
<td>theWindowPtr</td>
</tr>
</tbody>
</table>

Word—Space for result
Word—Starting X coordinate of cursor, in global coordinates
Word—Starting Y coordinate of cursor, in global coordinates
Long—POINTER to window's GrafPort

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>zoom</td>
</tr>
</tbody>
</table>

Word—BOOLEAN; TRUE if zoom region was selected, FALSE if not

← SP

Window Manager routines  25-129
C

extern pascal Boolean TrackZoom(startX, startY, theWindowPtr)
Integer startX;
Integer startY;
GrafPortPtr theWindowPtr;
You can also use the following alternate form of the call:
extern pascal Boolean TrackZoom(start, theWindowPtr)
Point start;
GrafPortPtr theWindowPtr;
ValidRect

Removes a specified rectangle from the update region of the window whose GrafPort is the current port and tells the Window Manager to cancel any updates accumulated for that rectangle. The rectangle is clipped to the window's content region and is given in local coordinates.

Important

This routine changes the coordinates you give it. Save the coordinates if you need to restore them later.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>goodRectPtr</td>
</tr>
</tbody>
</table>

Long—POINTER to RECT specifying rectangle to be removed

Stack after call

| previous contents |

Errors

None

C

extern pascal void ValidRect(goodRectPtr)

Rect *goodRectPtr;
$3DOE ValidRgn

Removes a specified region from the update region of the window whose GrafPort is the current port and tells the Window Manager to cancel any updates accumulated for that region. The region is clipped to the window's content region and is given in local coordinates.

Important
This routine changes the coordinates you give it. Save the coordinates if you need to restore them later.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
<th>Long—HANDLE to region to be subtracted from update region</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- goodRgnHandle --</td>
<td>← SP</td>
</tr>
</tbody>
</table>

Stack after call

| previous contents | ← SP |

Errors
None

C

```c
extern pascal void ValidRgn(goodRgnHandle)
RgnHandle goodRgnHandle;
```
WindDragRect

Pulls a dotted outline of a specified rectangle around the screen, following the movements of the mouse until the mouse button is released. WindDragRect is a way of accessing the Control Manager DragRect routine. See the section "DragRect" in Chapter 4, "Control Manager," in Volume 1 for more information.

Parameters

Stack before call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long—Space for result</td>
</tr>
<tr>
<td>Long—POINTER to routine; NIL for default</td>
</tr>
<tr>
<td>Long—POINTER to pattern to use for drag outline</td>
</tr>
<tr>
<td>Word—X coordinate of starting point, in global coordinates</td>
</tr>
<tr>
<td>Word—Y coordinate of starting point, in global coordinates</td>
</tr>
<tr>
<td>Long—POINTER to RECT data structure of rectangle to be dragged</td>
</tr>
<tr>
<td>Long—POINTER to RECT of limit rectangle</td>
</tr>
<tr>
<td>Long—POINTER to RECT of slop rectangle</td>
</tr>
<tr>
<td>Word—Bit flag customizing drag rectangle (see Figure 4-25 in Volume 1)</td>
</tr>
</tbody>
</table>

Stack after call

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long—High word = amount X changed; low word = amount Y changed</td>
</tr>
</tbody>
</table>

← SP

Window Manager routines 25-133
Errors

None

C

extern pascal LongWord WindDragRect(actionProcPtr, dragPatternPtr, startX, startY, dragRectPtr, limitRectPtr, slopRectPtr, dragFlag)

VoidProcPtr    actionProcPtr;
Pattern     dragPatternPtr;
Integer     startX;
Integer     startY;
Rect *dragRectPtr;
Rect *limitRectPtr;
Rect *slopRectPtr;
Word     dragFlag;

You can also use the following alternate form of the call:

extern pascal LongWord WindDragRect(actionProcPtr, dragPatternPtr, start, dragRectPtr, limitRectPtr, slopRectPtr, dragFlag)

VoidProcPtr    actionProcPtr;
Pattern     dragPatternPtr;
Point     start;
Rect *dragRectPtr;
Rect *limitRectPtr;
Rect *slopRectPtr;
Word     dragFlag;
**$250E WindNewRes**

Closes the Window Manager’s GrafPort and opens a new GrafPort in the other Super Hi-Res resolution. However, the screen is not redrawn by the Window Manager in the new resolution. You can then call the RefreshDesktop routine when all resolution changes, such as changes to the desktop pattern and window colors, have been completed.

You should call WindNewRes after the screen resolution has been changed.

**Parameters**

The stack is not affected by this call. There are no input or output parameters.

**Errors**

None

**C**

```pascal
extern pascal void WindNewRes()
```
WindowGlobal

Specifies a mask that determines how the Window Manager performs tasks. If windowGlobalMask has bit 15 set to 1, the mask will be ANDed with the global flag. If windowGlobalMask has bit 15 set to 0, the mask will be ORed with the global flag. WindowGlobal also returns the current state of the global flag in windowGlobalFlag.

---

**Important**

At the time of publication, all bits except 15 and 0 are reserved for future use.

---

**Parameters**

---

**Stack before call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordspace</td>
</tr>
<tr>
<td>windowGlobalMask</td>
</tr>
</tbody>
</table>

Word—Space for result
Word—Global mask (see Table 25-11)
← SP

---

**Stack after call**

<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>windowGlobalFlag</td>
</tr>
</tbody>
</table>

Word—Current state of window global flag (see Table 25-11)
← SP

---

**Errors**

None

---

**C**

```c
extern pascal Word WindowGlobal(windowGlobalMask);
Word WindowGlobalMask;
```
Stopping window highlighting

The only valid values for the window global mask at the time of publication are used to change the Window Manager's normal highlighting procedure, as shown in Table 25-10.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000</td>
<td>Flag does not change (used to retrieve current state of flag)</td>
</tr>
<tr>
<td>$0001</td>
<td>Stop the Window Manager from highlighting and unhighlighting windows when NewWindow and CloseWindow calls are made</td>
</tr>
<tr>
<td>$FFFFE</td>
<td>Return the Window Manager to normal highlighting operation</td>
</tr>
</tbody>
</table>

Under normal circumstances, the Window Manager highlights and unhighlights windows appropriately when they become active or inactive. However, you may wish to suppress that highlighting in order to speed up window redrawing. For example, the following pseudocode sequence demonstrates one use of WindowGlobal:

```plaintext
NewWindow(appropriate parameters);   ; Put up a new window, which is
                               ; automatically highlighted.
WindowGlobal($0001);                ; Turn highlighting off.
do Alert();                         ; Put up an alert window, allow the user
                               ; to choose something, and close the
                               ; alert. Although the alert window is
                               ; on top of the document window, the
                               ; document window remains highlighted.
WindowGlobal($FFFFE);               ; Return the Window Manager to normal
                               ; highlighting operations.
```

The value returned in `windowGlobalFlag` indicates the state of the flag after any changes have been made. The values at the time of publication are shown in Table 25-11.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000</td>
<td>Normal highlighting off</td>
</tr>
<tr>
<td>$0001</td>
<td>Normal highlighting on</td>
</tr>
</tbody>
</table>
$270E \textbf{ZoomWindow}

Switches the size and position of a specified window between its current size and position and its maximum size. If the routine is called again before the specified window is moved or resized, the window will be resized and positioned to the size and position before the last \texttt{ZoomWindow} was performed. When a \texttt{SizeWindow} or \texttt{MoveWindow} is performed while a window is zoomed, the last size becomes the new size and position.

\textbf{Parameters}

\texttt{Stack before call}

\begin{verbatim}
<table>
<thead>
<tr>
<th>previous contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- theWindowPtr --</td>
</tr>
</tbody>
</table>
\end{verbatim}

\texttt{Long—POINTER to window’s GrafPort} \leftarrow \texttt{SP}

\texttt{Stack after call}

\begin{verbatim}
| previous contents |
\end{verbatim}

\leftarrow \texttt{SP}

\textbf{Errors}

None

\texttt{C}

\begin{verbatim}
extern pascal void ZoomWindow(theWindowPtr)
GrafPortPtr theWindowPtr;
\end{verbatim}

25-138 \texttt{Window Manager routines}
Window Manager summary

This section briefly summarizes the constants, data structures, and tool set error codes contained in the Window Manager.

---

**Important**

These definitions are provided in the appropriate Interface file.

---

### Table 25-12

Window Manager constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axis parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wNoConstraint</td>
<td>$0000</td>
<td>No constraint on movement</td>
</tr>
<tr>
<td>wHAxisOnly</td>
<td>$0001</td>
<td>Horizontal axis only</td>
</tr>
<tr>
<td>wVAxisOnly</td>
<td>$0002</td>
<td>Vertical axis only</td>
</tr>
<tr>
<td>Desktop commands</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FromDesk</td>
<td>$00</td>
<td>Subtract region from desktop</td>
</tr>
<tr>
<td>ToDesk</td>
<td>$01</td>
<td>Add region to desktop</td>
</tr>
<tr>
<td>GetDesktop</td>
<td>$02</td>
<td>Get handle to desktop region</td>
</tr>
<tr>
<td>SetDesktop</td>
<td>$03</td>
<td>Set handle to desktop region</td>
</tr>
<tr>
<td>GetDeskPat</td>
<td>$04</td>
<td>Address of pattern or drawing routine</td>
</tr>
<tr>
<td>SetDeskPat</td>
<td>$05</td>
<td>Change address of pattern or drawing routine</td>
</tr>
<tr>
<td>GetVisDesktop</td>
<td>$06</td>
<td>Get desktop region minus visible windows</td>
</tr>
<tr>
<td>BackGroundRgn</td>
<td>$07</td>
<td>For drawing directly on desktop</td>
</tr>
<tr>
<td>SendBehind values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>toBottom</td>
<td>−2</td>
<td>Send window to bottom</td>
</tr>
<tr>
<td>topMost</td>
<td>−1</td>
<td>Make window frontmost</td>
</tr>
<tr>
<td>bottomMost</td>
<td>$0000</td>
<td>Make window bottom</td>
</tr>
</tbody>
</table>

(continued)
**Table 25-12** (continued)

Window Manager constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Task mask values</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tmMenuKey</td>
<td>$0001</td>
<td>Handle menu key events</td>
</tr>
<tr>
<td>tmUpdate</td>
<td>$0002</td>
<td>Handle update events</td>
</tr>
<tr>
<td>tmFindW</td>
<td>$0004</td>
<td>FindWindow called</td>
</tr>
<tr>
<td>tmMenuSel</td>
<td>$0008</td>
<td>MenuSelect called</td>
</tr>
<tr>
<td>tmOpenNDA</td>
<td>$0010</td>
<td>OpenNDA called</td>
</tr>
<tr>
<td>tmSysClick</td>
<td>$0020</td>
<td>SystemClick called</td>
</tr>
<tr>
<td>tmDragW</td>
<td>$0040</td>
<td>DragWindow called</td>
</tr>
<tr>
<td>tmContent</td>
<td>$0080</td>
<td>Activate inactive window on click in content region</td>
</tr>
<tr>
<td>tmClose</td>
<td>$0100</td>
<td>TrackGoAway called</td>
</tr>
<tr>
<td>tmZoom</td>
<td>$0200</td>
<td>TrackZoom called</td>
</tr>
<tr>
<td>tmGrow</td>
<td>$0400</td>
<td>GrowWindow called</td>
</tr>
<tr>
<td>tmScroll</td>
<td>$0800</td>
<td>Enable scrolling and activate inactive window on click in scroll bar</td>
</tr>
<tr>
<td>tmSpecial</td>
<td>$1000</td>
<td>Handle events in special menu items</td>
</tr>
<tr>
<td>tmCRedraw</td>
<td>$2000</td>
<td>Redraw controls upon activate event</td>
</tr>
<tr>
<td>tmInactive</td>
<td>$4000</td>
<td>Allows selection of inactive menu items</td>
</tr>
<tr>
<td>tmInfo</td>
<td>$8000</td>
<td>Don't activate inactive window on click in information bar</td>
</tr>
<tr>
<td><strong>TaskMaster codes</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>winDesk</td>
<td>$0010</td>
<td>In Desktop</td>
</tr>
<tr>
<td>winMenuBar</td>
<td>$0011</td>
<td>In system menu bar</td>
</tr>
<tr>
<td>wClickCalled</td>
<td>$0012</td>
<td>System click called</td>
</tr>
<tr>
<td>winContent</td>
<td>$0013</td>
<td>In content region</td>
</tr>
<tr>
<td>winDrag</td>
<td>$0014</td>
<td>In drag region</td>
</tr>
<tr>
<td>winGrow</td>
<td>$0015</td>
<td>In grow region, active window only</td>
</tr>
<tr>
<td>winGoAway</td>
<td>$0016</td>
<td>In go-away region, active window only</td>
</tr>
<tr>
<td>winZoom</td>
<td>$0017</td>
<td>In zoom region, active window only</td>
</tr>
<tr>
<td>winInfo</td>
<td>$0018</td>
<td>In information bar</td>
</tr>
<tr>
<td>winSpecial</td>
<td>$0019</td>
<td>Item ID selected was 250–255</td>
</tr>
<tr>
<td>winDeskItem</td>
<td>$001A</td>
<td>Item ID selected was 1–249</td>
</tr>
<tr>
<td>winFrame</td>
<td>$001B</td>
<td>In Frame, but not on anything else</td>
</tr>
<tr>
<td>wInactMenu</td>
<td>$001C</td>
<td>Inactive menu item selected</td>
</tr>
<tr>
<td>wClosedNDA</td>
<td>$001D</td>
<td>Desk accessory closed</td>
</tr>
<tr>
<td>wCalledSysEdit</td>
<td>$001E</td>
<td>Inactive menu item selected</td>
</tr>
<tr>
<td>winSysWindow</td>
<td>$8000</td>
<td>High-order bit set for system windows</td>
</tr>
</tbody>
</table>

25-140 Chapter 25: Window Manager
Table 25-12 (continued)  
Window Manager constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>varCode values</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wDraw</td>
<td>$00</td>
<td>Draw window frame command</td>
</tr>
<tr>
<td>wHit</td>
<td>$01</td>
<td>Hit test command</td>
</tr>
<tr>
<td>wCalcRgns</td>
<td>$02</td>
<td>Compute regions command</td>
</tr>
<tr>
<td>wNew</td>
<td>$03</td>
<td>Initialization command</td>
</tr>
<tr>
<td>wDispose</td>
<td>$04</td>
<td>Dispose command</td>
</tr>
<tr>
<td><strong>wFrame values</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fHilited</td>
<td>$0001</td>
<td>Window is highlighted</td>
</tr>
<tr>
<td>fZoomed</td>
<td>$0002</td>
<td>Window is zoomed</td>
</tr>
<tr>
<td>fAllocated</td>
<td>$0004</td>
<td>Window record was allocated</td>
</tr>
<tr>
<td>fCtlTie</td>
<td>$0008</td>
<td>State of window's controls tied to state of window</td>
</tr>
<tr>
<td>fInfo</td>
<td>$0010</td>
<td>Window has an information bar</td>
</tr>
<tr>
<td>fVis</td>
<td>$0020</td>
<td>Window is visible</td>
</tr>
<tr>
<td>fQContent</td>
<td>$0040</td>
<td>Select window if mouseDownEvt in inactive window's content</td>
</tr>
<tr>
<td>fMove</td>
<td>$0080</td>
<td>Window can be dragged</td>
</tr>
<tr>
<td>fZoom</td>
<td>$0100</td>
<td>Window has a zoom box</td>
</tr>
<tr>
<td>fFlex</td>
<td>$0200</td>
<td>Data height and width are flexible</td>
</tr>
<tr>
<td>fGrow</td>
<td>$0400</td>
<td>Window has a size box</td>
</tr>
<tr>
<td>fBScroll</td>
<td>$0800</td>
<td>Window has a horizontal scroll bar</td>
</tr>
<tr>
<td>fRScroll</td>
<td>$1000</td>
<td>Window has a vertical scroll bar</td>
</tr>
<tr>
<td>fAlert</td>
<td>$2000</td>
<td>Alert-type window frame</td>
</tr>
<tr>
<td>fClose</td>
<td>$4000</td>
<td>Window has a close box</td>
</tr>
<tr>
<td>fTitle</td>
<td>$8000</td>
<td>Window has a title bar</td>
</tr>
<tr>
<td><strong>Record sizes</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>winSize</td>
<td>$145</td>
<td>Size of window record</td>
</tr>
<tr>
<td>wmTaskRecSize</td>
<td>$16</td>
<td>Size of task record</td>
</tr>
</tbody>
</table>
Table 25-13
Window Manager data structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>WindRec (window record)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wNext</td>
<td>$00</td>
<td>GrafPortPtr</td>
<td>Pointer to next window record</td>
</tr>
<tr>
<td>port</td>
<td>$04</td>
<td>GrafPort</td>
<td>Window's port</td>
</tr>
<tr>
<td>wPadding</td>
<td>$AE</td>
<td>16 bytes</td>
<td>Space for possible future expansion</td>
</tr>
<tr>
<td>wStructRgn</td>
<td>$BE</td>
<td>RgnHandle</td>
<td>Region of frame plus content</td>
</tr>
<tr>
<td>wContRgn</td>
<td>$C2</td>
<td>RgnHandle</td>
<td>Content region</td>
</tr>
<tr>
<td>wUpdateRgn</td>
<td>$C6</td>
<td>RgnHandle</td>
<td>Update region</td>
</tr>
<tr>
<td>wControls</td>
<td>$CA</td>
<td>CtlRecHndl</td>
<td>Window's control list</td>
</tr>
<tr>
<td>wFrameCtrls</td>
<td>$CE</td>
<td>CtlRecHndl</td>
<td>Window frame's control list</td>
</tr>
<tr>
<td>wFrame</td>
<td>$D2</td>
<td>Word</td>
<td>Bit flags</td>
</tr>
<tr>
<td>WindColor (window color table)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>frameColor</td>
<td>$00</td>
<td>Word</td>
<td>Color of window frame</td>
</tr>
<tr>
<td>titleColor</td>
<td>$02</td>
<td>Word</td>
<td>Color of title and bar</td>
</tr>
<tr>
<td>tBarColor</td>
<td>$04</td>
<td>Word</td>
<td>Color and pattern of title bar</td>
</tr>
<tr>
<td>growColor</td>
<td>$06</td>
<td>Word</td>
<td>Color of grow box</td>
</tr>
<tr>
<td>infoColor</td>
<td>$08</td>
<td>Word</td>
<td>Color of information bar</td>
</tr>
<tr>
<td>Paramlist (NewWindow parameter list)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>paramLength</td>
<td>$00</td>
<td>Word</td>
<td>Total number of bytes in parameter table</td>
</tr>
<tr>
<td>wFrameBits</td>
<td>$02</td>
<td>Word</td>
<td>Bit flag that describes window</td>
</tr>
<tr>
<td>wTitle</td>
<td>$04</td>
<td>Pointer</td>
<td>Pointer to window's title</td>
</tr>
<tr>
<td>wRefCon</td>
<td>$08</td>
<td>LongWord</td>
<td>Reserved for application use</td>
</tr>
<tr>
<td>wZoom</td>
<td>$0C</td>
<td>Rect</td>
<td>Size and position of content</td>
</tr>
<tr>
<td>wColor</td>
<td>$14</td>
<td>WindColorPtr</td>
<td>Pointer to window's color table</td>
</tr>
<tr>
<td>wYOrigin</td>
<td>$18</td>
<td>Integer</td>
<td>Content's vertical origin</td>
</tr>
<tr>
<td>wXOrigin</td>
<td>$1A</td>
<td>Integer</td>
<td>Content's horizontal origin</td>
</tr>
<tr>
<td>wDataH</td>
<td>$1C</td>
<td>Word</td>
<td>Height of entire document</td>
</tr>
<tr>
<td>wDataW</td>
<td>$1E</td>
<td>Word</td>
<td>Width of entire document</td>
</tr>
<tr>
<td>wMaxH</td>
<td>$20</td>
<td>Word</td>
<td>Maximum height of content allowed by GrowWindow</td>
</tr>
</tbody>
</table>
Table 25-13 (continued)
Window Manager data structures

<table>
<thead>
<tr>
<th>Name</th>
<th>Offset</th>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paramlist (NewWindow parameter list)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wScrollVer</td>
<td>$24</td>
<td>Word</td>
<td>Number of pixels to scroll vertically when arrow is clicked</td>
</tr>
<tr>
<td>wScrollHor</td>
<td>$26</td>
<td>Word</td>
<td>Number of pixels to scroll horizontally when arrow is clicked</td>
</tr>
<tr>
<td>wPageVer</td>
<td>$28</td>
<td>Word</td>
<td>Number of pixels to scroll vertically for page</td>
</tr>
<tr>
<td>wPageHor</td>
<td>$2A</td>
<td>Word</td>
<td>Number of pixels to scroll horizontally for page</td>
</tr>
<tr>
<td>wInfoRefCon</td>
<td>$2C</td>
<td>LongWord</td>
<td>Value passed to draw information bar routine</td>
</tr>
<tr>
<td>wInfoHeight</td>
<td>$30</td>
<td>Word</td>
<td>Height of information bar</td>
</tr>
<tr>
<td>wFrameDefProc</td>
<td>$32</td>
<td>LongProcPtr</td>
<td>Pointer to standard window definition procedure</td>
</tr>
<tr>
<td>wInfoDefProc</td>
<td>$36</td>
<td>VoidProcPtr</td>
<td>Pointer to routine that draws the interior of the information bar</td>
</tr>
<tr>
<td>wContDefProc</td>
<td>$3A</td>
<td>VoidProcPtr</td>
<td>Pointer to routine that draws the interior of the content region</td>
</tr>
<tr>
<td>wPosition</td>
<td>$3E</td>
<td>Rect</td>
<td>Window's starting position and size</td>
</tr>
<tr>
<td>wPlane</td>
<td>$46</td>
<td>GrafPortPtr</td>
<td>Window's starting plane</td>
</tr>
<tr>
<td>wStorage</td>
<td>$4A</td>
<td>WindRecPtr</td>
<td>Pointer to memory to use for window record</td>
</tr>
<tr>
<td>WmTaskRec (task record)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wmWhat</td>
<td>$00</td>
<td>Word</td>
<td>Unchanged from event record</td>
</tr>
<tr>
<td>wmMessage</td>
<td>$02</td>
<td>LongWord</td>
<td>Unchanged from event record</td>
</tr>
<tr>
<td>wmWhen</td>
<td>$06</td>
<td>LongWord</td>
<td>Unchanged from event record</td>
</tr>
<tr>
<td>wmWhere</td>
<td>$0A</td>
<td>Point</td>
<td>Unchanged from event record</td>
</tr>
<tr>
<td>wmmModifiers</td>
<td>$0E</td>
<td>Word</td>
<td>Unchanged from event record</td>
</tr>
<tr>
<td>wmTaskData</td>
<td>$10</td>
<td>LongWord</td>
<td>TaskMaster return value</td>
</tr>
<tr>
<td>wmTaskMask</td>
<td>$14</td>
<td>LongWord</td>
<td>TaskMaster feature mask</td>
</tr>
</tbody>
</table>

Note: The actual assembly-language equates have a lowercase letter o in front of all the names given in this table.
Table 25-14
Window Manager error codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0E01</td>
<td>paramLenErr</td>
<td>First word of parameter list is the wrong size</td>
</tr>
<tr>
<td>$0E02</td>
<td>allocateErr</td>
<td>Unable to allocate memory for window record</td>
</tr>
<tr>
<td>$0E03</td>
<td>taskMaskErr</td>
<td>Reserved bits not clear in <code>wmTaskMask</code> field of task record</td>
</tr>
</tbody>
</table>

Chapter 25: Window Manager
The Tool Locator system, which is flexible enough to allow you to write your own tool sets for use in your applications, supports both system tools and user tools.

When writing your own tool set, the following must be kept in mind:

- Tool sets get control in full native mode.
- Work space should be dynamically assigned. Tool sets should not use any fixed RAM locations for work space; they should obtain their work space from the Memory Manager. This avoids memory conflicts, such as those caused by fixed usage of screen holes.
- A simple interrupt environment should be supplied. Each function should increment or decrement the busy flag, be reentrant, or disable interrupts during execution. The most common approach is to use the busy flag. See Chapter 19, "Scheduler," for more information.
- Before returning control to the caller, routines must restore the caller's execution environment. This includes the data bank register, the direct-page register, and any soft switches.
- Routines should not assume the presence of any operating system unless the operating system is directly relevant; for example, a routine that reads or writes a file, where other considerations demand that the file type be known anyway.
Structure of the Tool Locator

The Tool Locator requires a few fixed RAM locations and no fixed ROM locations. All functions are accessed through the Tool Locator via their tool set number and function number. The Tool Locator uses the tool set number to find an entry in the tool pointer table (TPT). This table contains pointers to function pointer tables (FPTs). Each tool set has an FPT that contains pointers to the individual routines in the tool. The Tool Locator uses the function number to find the address of the routine being called.

Each tool in ROM has an FPT in ROM. In ROM, there is also a TPT that points to all the FPTs in ROM. One fixed RAM location is used to point to this TPT in ROM. This location is initialized at power up and warm boot by the firmware. In this way, the address of the TPT in ROM does not ever have to be fixed.

The TPT has the form shown in Table A-1.

<table>
<thead>
<tr>
<th>Item</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>4 bytes</td>
<td>Number of tool sets plus 1</td>
</tr>
<tr>
<td>Pointer to TS 1 FPT</td>
<td>4 bytes</td>
<td>Pointer to FPT for tool set number 1</td>
</tr>
<tr>
<td>Pointer to TS 2 FPT</td>
<td>4 bytes</td>
<td>Pointer to FPT for tool set number 2</td>
</tr>
<tr>
<td>Pointer to TS n FPT</td>
<td>4 bytes</td>
<td>Pointer to FPT for tool set number n</td>
</tr>
</tbody>
</table>

The FPT has the form shown in Table A-2.

<table>
<thead>
<tr>
<th>Item</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>4 bytes</td>
<td>Number of routines plus 1</td>
</tr>
<tr>
<td>Address of F1 –1</td>
<td>4 bytes</td>
<td>Pointer to BootInit routine minus 1</td>
</tr>
<tr>
<td>Address of F2 –1</td>
<td>4 bytes</td>
<td>Pointer to StartUp routine minus 1</td>
</tr>
<tr>
<td>Address of F3 –1</td>
<td>4 bytes</td>
<td>Pointer to ShutDown routine minus 1</td>
</tr>
<tr>
<td>Address of F4 –1</td>
<td>4 bytes</td>
<td>Pointer to Version routine minus 1</td>
</tr>
<tr>
<td>Address of F5 –1</td>
<td>4 bytes</td>
<td>Pointer to Reset routine minus 1</td>
</tr>
<tr>
<td>Address of F6 –1</td>
<td>4 bytes</td>
<td>Pointer to Status routine minus 1</td>
</tr>
<tr>
<td>Address of F7 –1</td>
<td>4 bytes</td>
<td>Pointer to reserved routine minus 1</td>
</tr>
<tr>
<td>Address of F8 –1</td>
<td>4 bytes</td>
<td>Pointer to reserved routine minus 1</td>
</tr>
<tr>
<td>Address of F9 –1</td>
<td>4 bytes</td>
<td>Pointer to first nonrequired routine minus 1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Address of Fn - 1</td>
<td>4 bytes</td>
<td>Pointer to last nonrequired routine minus 1</td>
</tr>
</tbody>
</table>
Tool set numbers and function numbers

Each system tool is assigned a permanent tool number. Assignment starts at one and continues with each successive integer.

Each function within a tool set is assigned a permanent function number. For the functions within each tool, assignment starts at 1 and continues with each successive integer. Thus, each function has a unique, permanent identifier of the form (tsNum, funcNum). Both tsNum and funcNum are 8-bit numbers. The tool set numbers assigned to the Apple tools are shown in Table 24-2 in Chapter 24, “Tool Locator.”

For each tool set, certain standard routines must be present. Each tool set must have a boot initialization routine that is executed at boot time either by the ROM startup code or when the tool set is installed in the system. In addition, each tool set has an application StartUp routine, an application ShutDown routine to allow an application to turn each tool on and off, a Version routine that returns information about the version of the tool, a Reset routine to be called when the system is reset, and a Status routine to indicate whether the tool set is active.

All tools must return version information in the form of a word. The high byte of the word indicates the major release number (starting with 1). The low byte of the word indicates the minor release number (starting with 0). The most significant bit of the word indicates whether the code is an official release or a prototype (no distinction is made between alpha, beta, or other prototype releases).

The standard routines are summarized in Table A-3.

Table A-3
Standard tool set routine numbers

<table>
<thead>
<tr>
<th>FuncNum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Boot initialization function for each tool set</td>
</tr>
<tr>
<td>2</td>
<td>Application startup function for each tool set</td>
</tr>
<tr>
<td>3</td>
<td>Application shutdown function for each tool set</td>
</tr>
<tr>
<td>4</td>
<td>Version information</td>
</tr>
<tr>
<td>5</td>
<td>Reset</td>
</tr>
<tr>
<td>6</td>
<td>Status</td>
</tr>
<tr>
<td>7</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>8</td>
<td>Reserved for future use</td>
</tr>
</tbody>
</table>
Obtaining memory

Tool sets are to obtain any memory they need dynamically (using as little fixed memory as possible) through the Memory Manager. To do that, a tool set needs some way to find the location of its data structures. The Tool Locator maintains a table of work area pointers for the individual tools. The work area pointer table (WAPT) is a table of pointers to the work areas of individual tools.

In the WAPT, each tool will have an entry for its own use. Entries are assigned by tool set number (tool 04 has entry 04 and so on). A pointer to the WAPT is kept at a fixed memory location in RAM so that space for the table can be allocated dynamically.

The Tool Locator system permanently reserves some space in bank $E1 for the purposes shown in Table A-4.

Table A-4
Tool Locator permanent RAM space

<table>
<thead>
<tr>
<th>Address</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E105C0</td>
<td>4 bytes</td>
<td>Pointer to the active TPT. The pointer is to the ROM-based TPT if there are no RAM-based tool sets and no RAM-based ROM patches. Otherwise, it will point to a RAM-based TPT.</td>
</tr>
<tr>
<td>$E103C4</td>
<td>4 bytes</td>
<td>Pointer to the active user's TPT. This pointer is 0 initially, indicating that no user tools are present.</td>
</tr>
<tr>
<td>$E103C8</td>
<td>4 bytes</td>
<td>Pointer to the WAPT. The WAPT parallels the TPT. Each WAPT entry is a pointer to a work area assigned to the corresponding tool set. At startup time, each WAPT entry is set to 0, indicating no assigned work area.</td>
</tr>
<tr>
<td>$E103CC</td>
<td>4 bytes</td>
<td>Pointer to the user's Work Area Pointer Table (WAPT).</td>
</tr>
<tr>
<td>$E10000</td>
<td>16 bytes</td>
<td>Entry points to the dispatcher.</td>
</tr>
</tbody>
</table>

This is the only RAM permanently reserved by the Tool Locator system.
Tool Locator system initialization

Each tool set is initialized before use by application programs. Two types of initialization are needed: boot initialization and application initialization. Boot initialization occurs either at system startup time (boot time) or, for tool sets loaded from disk, when the tool is installed. Regardless of the applications to be executed, the system calls the boot initialization function of every tool set. Thus, each tool set must have a boot initialization routine ($funcNum = 1$), even if it does nothing. This function has no input or output parameters.

Application initialization occurs during application execution. The application calls the application startup function ($funcNum = 2$) of each tool set it will use. The application startup function performs the chores needed to start up the tool set so the application can use it. This function may have inputs and outputs, as defined by the individual tool set.

The application shutdown function ($funcNum = 3$) should be executed as soon as the application no longer needs to use the tool. The shutdown releases the resources used by the tool. As a precaution against applications forgetting to execute the shutdown function, the startup function should either execute the shutdown function itself or do something else to ensure a reasonable startup state.

The provision of two initialization times reflects the needs of currently envisioned tools. On the one hand, for example, the Memory Manager requires boot time initialization because it must operate properly even before any application has been loaded. On the other hand, SANE needs to be initialized only if the system executes some application or desk accessory that uses it. Initializing only the tool sets that will be used saves resources, particularly RAM.

Disk and RAM structure of tool sets

System tool sets are load files kept in the TOOLS subdirectory of the SYSTEM directory. Their file type is $BA$; each tool set begins with a function pointer table.

User tool sets may be in any form; it is the responsibility of the application to properly load and install such tool sets.
Installing your tool set

Before you make any calls to a user tool set, you must install it into the system. You do this by calling the Tool Locator routine `SetTSPtr`. `SetTSPtr` takes three inputs on the stack as follows:

| previous contents |  
|-------------------|---
| `userOrSystem`    | Word—$0000 = system tool set, $8000 = user tool set  
| `tsNum`           | Word—Tool set number of the tool set  
| `fPtr`            | Long—POINTER to function pointer table for tool set  

When `SetTSPtr` is called, your tool is installed in the system and its boot initialization function call is executed. The following example illustrates installation of a sample user tool:

```
;-----------------------------------------------
Install START
clc ; Switch to full native mode and
xce ; save initial state
php
rep #S30 ; 16-bit registers
PushWord $8000 ; Signal a user tool
PushWord #$23 ; Put the tool number on the stack
PushLong #CallTable ; Point to call table
_SetTSPtr
plp ; Restore machine state
xce
rts
END
```

A-6 Appendix A: Writing Your Own Tool Set
CallTable START
long (TheEnd-CallTable)/4
long MyBootInit-1
long MyStartUp-1
long MyShutDown-1
long MyVersion-1
long MyReset-1
long NotImp-1
long NotImp-1
long NotImp-1
long FirstFunc-1
long LastFunc-1
TheEND

END

MyBootInit START ; Called when installed
lda #0
clc
rtl
END

MyStartUp START ; User passes me word containing location
RTL1 equ 1
RTL2 equ RTL1+3
ZPToUse equ RTL2+3
ToolNum equ 5

lda ZPToUse,s
pea $8000
phx
pea 0
pha
lda ToolNum,s
and #$00FF
sta ToolNum,s
_SetWAP

; Installing your tool set

Installing your tool set

A-7
Next, the input parameter must be removed from the stack.
Do this by sliding the two return addresses up the stack by two bytes,
such that the most-significant word of the second-level return address
slides right into the spot previously occupied by the input parameter.
This isn't very difficult, because the two addresses occupy the same
number of words.

lda RTL2+1, s  
sta ZPTuse, s  
lda RTL1+2, s  
sta ZPTuse-2  
lda RTL1  
sta RTL1+2

; The bytes at SP+1 and SP+2 (or RTL1 and RTL1+1) are empty and may be pulled
pla  ; as a single word!
lda #0  ; Report that no error occurred
clc
rtl  ; Done

MyShutDown START

MyVersion START

RTL1 equ 1
RTL2 equ RTL1+3
VerNum equ RTL2+3

lda #$90  ; Version 1.0 prototype
sta VerNum, s
lda #0
clc
rtl

END

A-8 Appendix A: Writing Your Own Tool Set
MyReset START

lda #0
clc
rtl
END

NotImp START

txa          ; Tool set number and function number were
xb &=00000001 ; in X on entry
ora $00FF    ; Tool set number was in LSB, move to MSB
sec
rtl
END

FirstFunc START

lda #0
clc
rtl
END

LastFunc START

lda #0
clc
rtl
END

Notes

; The long macro deposits a 4-byte value in memory, low bytes first.
; The PushWord macro pushes a word onto the stack (either from a memory
; location or with a pea instruction if # is used).
; The PushLong macro pushes a long on the stack (either from memory
; or with two pea instructions if # is used).

Disk and RAM structure of tool sets A-9
Function execution environment

When your function is called, the machine is in full native mode and the following three registers are set with specific information to make the function's job easier:

- A register = low-order word of entry in WAPT for tool
- Y register = high-order word of entry in WAPT for tool
- X register = function number and tool set number

When the function is called, the stack looks like this:

<table>
<thead>
<tr>
<th>previous contents</th>
<th>outputSpace</th>
<th>Word or Long — Space for output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>input1</td>
<td>Word or Long — First input</td>
</tr>
<tr>
<td></td>
<td>input2</td>
<td>Word or Long — Second input</td>
</tr>
<tr>
<td></td>
<td>inputLast</td>
<td>Word or Long — Last input</td>
</tr>
<tr>
<td></td>
<td>RTL</td>
<td>3 bytes — RTL address</td>
</tr>
<tr>
<td></td>
<td>RTL</td>
<td></td>
</tr>
</tbody>
</table>
In Appendix B, the tool set error codes are listed and summarized in error number order. If a tool set error has not occurred, the carry flag (c flag) will not be set and the accumulator will contain $0000.

### Table B-1
#### Tool set error codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0001</td>
<td>pdosUnClmdIntErr</td>
<td>Unclaimed interrupt (ProDOS 16)</td>
</tr>
<tr>
<td>$0004</td>
<td>divByZeroErr</td>
<td>Division by 0</td>
</tr>
<tr>
<td>$000A</td>
<td>pdosVCBErr</td>
<td>Volume control block unusable (ProDOS 16)</td>
</tr>
<tr>
<td>$000B</td>
<td>pdosFCBErr</td>
<td>File control block unusable (ProDOS 16)</td>
</tr>
<tr>
<td>$000C</td>
<td>pdosBlkOErr</td>
<td>Block zero allocated illegally (ProDOS 16)</td>
</tr>
<tr>
<td>$000D</td>
<td>pdosIntShdwErr</td>
<td>Interrupt with I/O shadowing off (ProDOS 16)</td>
</tr>
<tr>
<td>$0015</td>
<td>segLoader1Err</td>
<td>Segment Loader error</td>
</tr>
<tr>
<td>$0017</td>
<td>sPackage0Err</td>
<td>Can't load a package</td>
</tr>
<tr>
<td>$0018</td>
<td>package1Err</td>
<td>Can't load a package</td>
</tr>
<tr>
<td>$0019</td>
<td>package2Err</td>
<td>Can't load a package</td>
</tr>
<tr>
<td>$001A</td>
<td>package3Err</td>
<td>Can't load a package</td>
</tr>
<tr>
<td>$001B</td>
<td>package4Err</td>
<td>Can't load a package</td>
</tr>
<tr>
<td>$001C</td>
<td>package5Err</td>
<td>Can't load a package</td>
</tr>
<tr>
<td>$001D</td>
<td>package6Err</td>
<td>Can't load a package</td>
</tr>
<tr>
<td>$001E</td>
<td>package7Err</td>
<td>Can't load a package</td>
</tr>
<tr>
<td>$0020</td>
<td>package8Err</td>
<td>Can't load a package</td>
</tr>
<tr>
<td>$0021</td>
<td>package9Err</td>
<td>Can't load a package</td>
</tr>
</tbody>
</table>

(continued)
<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0022</td>
<td>package10Err</td>
<td>Can't load a package</td>
</tr>
<tr>
<td>$0023</td>
<td>package11Err</td>
<td>Can't load a package</td>
</tr>
<tr>
<td>$0024</td>
<td>package12Err</td>
<td>Can't load a package</td>
</tr>
<tr>
<td>$0025</td>
<td>outOfMemErr</td>
<td>Out of memory</td>
</tr>
<tr>
<td>$0026</td>
<td>segLoader2Err</td>
<td>Segment Loader error</td>
</tr>
<tr>
<td>$0027</td>
<td>fMapTrshdErr</td>
<td>File map destroyed</td>
</tr>
<tr>
<td>$0028</td>
<td>stkOvrFlwErr</td>
<td>Stack overflow</td>
</tr>
<tr>
<td>$0030</td>
<td>psInstFlwErr</td>
<td>Please insert disk (File Manager alert)</td>
</tr>
<tr>
<td>$0032-53</td>
<td></td>
<td>Memory Manager errors</td>
</tr>
<tr>
<td>$0100</td>
<td>stuVolMntErr</td>
<td>Can't mount system startup volume</td>
</tr>
<tr>
<td>$0001</td>
<td>toolNotFoundErr</td>
<td>Specified tool set not found</td>
</tr>
<tr>
<td>$0002</td>
<td>funcNotFoundErr</td>
<td>Specified routine not found</td>
</tr>
<tr>
<td>$0110</td>
<td>toolVersionErr</td>
<td>Specified minimum version not found</td>
</tr>
<tr>
<td>$0111</td>
<td>messNotFoundErr</td>
<td>Specified message not found</td>
</tr>
<tr>
<td>$0201</td>
<td>memErr</td>
<td>Unable to allocate block</td>
</tr>
<tr>
<td>$0202</td>
<td>emptyErr</td>
<td>Illegal operation on an empty handle</td>
</tr>
<tr>
<td>$0203</td>
<td>notEmptyErr</td>
<td>Empty handle expected for this operation</td>
</tr>
<tr>
<td>$0204</td>
<td>lockErr</td>
<td>Illegal operation on a locked or immovable block</td>
</tr>
<tr>
<td>$0205</td>
<td>purgeErr</td>
<td>Attempt to purge an unpurgeable block</td>
</tr>
<tr>
<td>$0206</td>
<td>handleErr</td>
<td>Invalid handle</td>
</tr>
<tr>
<td>$0207</td>
<td>idErr</td>
<td>Invalid user ID</td>
</tr>
<tr>
<td>$0208</td>
<td>attrErr</td>
<td>Illegal operation with specified attributes</td>
</tr>
<tr>
<td>$0301</td>
<td>badInputErr</td>
<td>Bad input parameter</td>
</tr>
<tr>
<td>$0302</td>
<td>noDevParamErr</td>
<td>No device for input parameter</td>
</tr>
<tr>
<td>$0303</td>
<td>taskInstlErr</td>
<td>Specified task already in Heartbeat queue</td>
</tr>
<tr>
<td>$0304</td>
<td>noSigTaskErr</td>
<td>No signature detected in task header</td>
</tr>
<tr>
<td>$0305</td>
<td>queueDmgdErr</td>
<td>Damaged Heartbeat queue detected</td>
</tr>
<tr>
<td>$0306</td>
<td>taskNtFdErr</td>
<td>Specified task not in queue</td>
</tr>
<tr>
<td>$0307</td>
<td>firmTaskErr</td>
<td>Unsccessful firmware task</td>
</tr>
<tr>
<td>$0308</td>
<td>hbQueueBadErr</td>
<td>Damaged HeartBeat queue detected</td>
</tr>
<tr>
<td>$0309</td>
<td>unCnctdDevErr</td>
<td>Dispatch attempted to unconnected device</td>
</tr>
<tr>
<td>$030B</td>
<td>idTagNtAvlErr</td>
<td>No ID tag available</td>
</tr>
<tr>
<td>Code</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>$0401</td>
<td>alreadyInitialized</td>
<td>QuickDraw II already initialized</td>
</tr>
<tr>
<td>$0402</td>
<td>cannotReset</td>
<td>Never used</td>
</tr>
<tr>
<td>$0403</td>
<td>notInitialized</td>
<td>QuickDraw II not initialized</td>
</tr>
<tr>
<td>$0410</td>
<td>screenReserved</td>
<td>Screen reserved</td>
</tr>
<tr>
<td>$0411</td>
<td>badRect</td>
<td>Invalid rectangle specified</td>
</tr>
<tr>
<td>$0420</td>
<td>notEqualChunkiness</td>
<td>Chunkiness not equal</td>
</tr>
<tr>
<td>$0430</td>
<td>rgnAlreadyOpen</td>
<td>Region already open</td>
</tr>
<tr>
<td>$0431</td>
<td>rgnNotOpen</td>
<td>Region not open</td>
</tr>
<tr>
<td>$0432</td>
<td>rgnScanOverflow</td>
<td>Region scan overflow</td>
</tr>
<tr>
<td>$0433</td>
<td>rgnFull</td>
<td>Region full</td>
</tr>
<tr>
<td>$0440</td>
<td>polyAlreadyOpen</td>
<td>Polygon already open</td>
</tr>
<tr>
<td>$0441</td>
<td>polyNotOpen</td>
<td>Polygon not open</td>
</tr>
<tr>
<td>$0442</td>
<td>polyTooBig</td>
<td>Polygon too big</td>
</tr>
<tr>
<td>$0450</td>
<td>badTableNum</td>
<td>Invalid color table number</td>
</tr>
<tr>
<td>$0451</td>
<td>badColorNum</td>
<td>Invalid color number</td>
</tr>
<tr>
<td>$0452</td>
<td>badScanLine</td>
<td>Invalid scan line number</td>
</tr>
<tr>
<td>$04FF</td>
<td>Not implemented</td>
<td></td>
</tr>
<tr>
<td>$0510</td>
<td>daNotFound</td>
<td>Specified DA not available</td>
</tr>
<tr>
<td>$0511</td>
<td>notSysWindow</td>
<td>Window pointer is not a pointer to a window owned by an NDA</td>
</tr>
<tr>
<td>$0601</td>
<td>emDupStrtUpErr</td>
<td>EMStartUp already called</td>
</tr>
<tr>
<td>$0602</td>
<td>emResetErr</td>
<td>Can't reset Event Manager</td>
</tr>
<tr>
<td>$0603</td>
<td>emNotActErr</td>
<td>Event Manager not active</td>
</tr>
<tr>
<td>$0604</td>
<td>emBadEvtCodeErr</td>
<td>Event code is greater than 15</td>
</tr>
<tr>
<td>$0605</td>
<td>emBadBtttnNoErr</td>
<td>Button number specified is not 0 or 1</td>
</tr>
<tr>
<td>$0606</td>
<td>emQSize2LrgErr</td>
<td>Size of event queue is greater than 3639</td>
</tr>
<tr>
<td>$0607</td>
<td>emNoMemQueueErr</td>
<td>Insufficient memory available for queue</td>
</tr>
<tr>
<td>$0681</td>
<td>emBadEvtQErr</td>
<td>Event queue damaged—fatal system error</td>
</tr>
<tr>
<td>$0682</td>
<td>emBadQHndlErr</td>
<td>Queue handle damaged—fatal system error</td>
</tr>
<tr>
<td>$0810</td>
<td>noDOCFndErr</td>
<td>No DOC or RAM found</td>
</tr>
<tr>
<td>$0811</td>
<td>docAddrRngErr</td>
<td>DOC address range error</td>
</tr>
<tr>
<td>$0812</td>
<td>noSAppInitErr</td>
<td>No SoundStartUp call made</td>
</tr>
<tr>
<td>$0813</td>
<td>invalGenNumErr</td>
<td>Invalid generator number</td>
</tr>
<tr>
<td>$0814</td>
<td>synthModeErr</td>
<td>Synthesizer mode error</td>
</tr>
<tr>
<td>$0815</td>
<td>genBusyErr</td>
<td>Generator already in use</td>
</tr>
<tr>
<td>$0817</td>
<td>mstrIRQNotAssignErr</td>
<td>Master IRQ not assigned</td>
</tr>
<tr>
<td>$0818</td>
<td>sndAlreadyStrtErr</td>
<td>Sound Tool Set already started</td>
</tr>
<tr>
<td>$08FF</td>
<td>unclaimedSndIntErr</td>
<td>Unclaimed sound interrupt error (reported through System Failure Manager)</td>
</tr>
</tbody>
</table>

(continued)
Table B-1 (continued)
Tool set error codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0910</td>
<td>cmdIncomple</td>
<td>Command not completed</td>
</tr>
<tr>
<td>$0911</td>
<td>cantSync</td>
<td>Can't synchronize with system</td>
</tr>
<tr>
<td>$0982</td>
<td>adbBusy</td>
<td>ADB busy (command pending)</td>
</tr>
<tr>
<td>$0983</td>
<td>devNotAtAddr</td>
<td>Device not present at address</td>
</tr>
<tr>
<td>$0984</td>
<td>srqListFull</td>
<td>SRQ list full</td>
</tr>
<tr>
<td>$0B01</td>
<td>imBadInptParam</td>
<td>Bad input parameter</td>
</tr>
<tr>
<td>$0B02</td>
<td>imIllegalChar</td>
<td>Illegal character in string</td>
</tr>
<tr>
<td>$0B03</td>
<td>imOverflow</td>
<td>Integer or Longint overflow</td>
</tr>
<tr>
<td>$0B04</td>
<td>imStrOverflow</td>
<td>String overflow</td>
</tr>
<tr>
<td>$0C01</td>
<td>badDevType</td>
<td>Illegal device type</td>
</tr>
<tr>
<td>$0C02</td>
<td>badDevNum</td>
<td>Illegal device number</td>
</tr>
<tr>
<td>$0C03</td>
<td>badMode</td>
<td>Illegal operation</td>
</tr>
<tr>
<td>$0C04</td>
<td>unDefHW</td>
<td>Undefined hardware error</td>
</tr>
<tr>
<td>$0C05</td>
<td>lostDev</td>
<td>Lost device: device no longer on-line</td>
</tr>
<tr>
<td>$0C06</td>
<td>lostFile</td>
<td>File no longer in diskette directory</td>
</tr>
<tr>
<td>$0C07</td>
<td>badTitle</td>
<td>Illegal filename</td>
</tr>
<tr>
<td>$0C08</td>
<td>noRoom</td>
<td>Insufficient space on specified diskette</td>
</tr>
<tr>
<td>$0C09</td>
<td>noDevice</td>
<td>Specified volume not on-line</td>
</tr>
<tr>
<td>$0C0A</td>
<td>noFile</td>
<td>Specified file not in directory of specified volume</td>
</tr>
<tr>
<td>$0C0B</td>
<td>dupFile</td>
<td>Duplicate file: attempt to rewrite a file</td>
</tr>
<tr>
<td>$0C0C</td>
<td>notClosed</td>
<td>Attempt to open file that is already open</td>
</tr>
<tr>
<td>$0C0D</td>
<td>notOpen</td>
<td>Attempt to access a closed file</td>
</tr>
<tr>
<td>$0C0E</td>
<td>badFormat</td>
<td>Error in reading real or integer number</td>
</tr>
<tr>
<td>$0C0F</td>
<td>ringBuffOFlo</td>
<td>Ring buffer overflow: characters arriving faster than the input buffer can accept them</td>
</tr>
<tr>
<td>$0C10</td>
<td>writeProtected</td>
<td>Specified diskette is write-protected</td>
</tr>
<tr>
<td>$0C40</td>
<td>devErr</td>
<td>Device error: device failed to complete a read or write correctly</td>
</tr>
<tr>
<td>$0E01</td>
<td>paramLenErr</td>
<td>First word of parameter list is the wrong size</td>
</tr>
<tr>
<td>$0E02</td>
<td>allocateErr</td>
<td>Unable to allocate window record</td>
</tr>
<tr>
<td>$0E03</td>
<td>taskMaskErr</td>
<td>Reserved bits not clear in \texttt{wmTaskMask} field of \texttt{WmTaskRec}</td>
</tr>
</tbody>
</table>

Window Manager codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0E01</td>
<td>paramLenErr</td>
<td>First word of parameter list is the wrong size</td>
</tr>
<tr>
<td>$0E02</td>
<td>allocateErr</td>
<td>Unable to allocate window record</td>
</tr>
<tr>
<td>$0E03</td>
<td>taskMaskErr</td>
<td>Reserved bits not clear in \texttt{wmTaskMask} field of \texttt{WmTaskRec}</td>
</tr>
</tbody>
</table>

Control Manager codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1001</td>
<td>wmNotStartedUp</td>
<td>Window Manager not initialized</td>
</tr>
<tr>
<td>Code</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>$1301</td>
<td>missingDriver</td>
<td>Specified driver not in the DRIVERS subdirectory of the SYSTEM subdirectory</td>
</tr>
<tr>
<td>$1302</td>
<td>portNotOn</td>
<td>Specified port not selected in the control panel</td>
</tr>
<tr>
<td>$1303</td>
<td>noPrintRecord</td>
<td>No print record specified</td>
</tr>
<tr>
<td>$1304</td>
<td>badLaserPrep</td>
<td>Version of LaserPrep file in LaserWriter is not compatible with this version of Print Manager</td>
</tr>
<tr>
<td>$1305</td>
<td>badLPFile</td>
<td>Version of LaserPrep file in DRIVERS subdirectory of SYSTEM subdirectory is not compatible with this version of Print Manager</td>
</tr>
<tr>
<td>$1306</td>
<td>papConnNotOpen</td>
<td>Connection can't be established with the LaserWriter</td>
</tr>
<tr>
<td>$1307</td>
<td>papReadWriteErr</td>
<td>Read-write error on the LaserWriter</td>
</tr>
<tr>
<td>$1321</td>
<td>startUpAlreadyMade</td>
<td>LLDStartUp call already made</td>
</tr>
<tr>
<td>$1322</td>
<td>invalidCtlVal</td>
<td>Invalid control value specified</td>
</tr>
<tr>
<td>$1401</td>
<td>leDupStrtUpErr</td>
<td>LEStartUp already called</td>
</tr>
<tr>
<td>$1402</td>
<td>leResetError</td>
<td>Can't reset LineEdit</td>
</tr>
<tr>
<td>$1403</td>
<td>leNotActiveErr</td>
<td>LineEdit not active</td>
</tr>
<tr>
<td>$1404</td>
<td>leScrapErr</td>
<td>Desk scrap too big to copy</td>
</tr>
<tr>
<td>$150A</td>
<td>badItemType</td>
<td>Inappropriate item type</td>
</tr>
<tr>
<td>$150B</td>
<td>newItemFailed</td>
<td>Item creation failed</td>
</tr>
<tr>
<td>$150C</td>
<td>itemNotFound</td>
<td>No such item</td>
</tr>
<tr>
<td>$150D</td>
<td>notModalDialog</td>
<td>Frontmost window not a modal dialog window</td>
</tr>
<tr>
<td>$1610</td>
<td>badScrapType</td>
<td>No scrap of this type</td>
</tr>
<tr>
<td>$1B01</td>
<td>fmDupStartUpErr</td>
<td>FMStartUp call already made</td>
</tr>
<tr>
<td>$1B02</td>
<td>fmResetErr</td>
<td>Can't reset the Font Manager</td>
</tr>
<tr>
<td>$1B03</td>
<td>fmNotActiveErr</td>
<td>Font Manager not active</td>
</tr>
<tr>
<td>$1B04</td>
<td>fmFamNotFndErr</td>
<td>Family not found</td>
</tr>
<tr>
<td>$1B05</td>
<td>fmFontNtFndErr</td>
<td>Font not found</td>
</tr>
<tr>
<td>$1B06</td>
<td>fmFontMemErr</td>
<td>Font not in memory</td>
</tr>
<tr>
<td>$1B07</td>
<td>fmSysFontErr</td>
<td>System font cannot be purgeable</td>
</tr>
<tr>
<td>$1B08</td>
<td>fmBadFamNumErr</td>
<td>Illegal family number</td>
</tr>
<tr>
<td>$1B09</td>
<td>fmBadSizeErr</td>
<td>Illegal font size</td>
</tr>
<tr>
<td>$1B0A</td>
<td>fmBadNameErr</td>
<td>Illegal name length</td>
</tr>
<tr>
<td>$1B0B</td>
<td>fmMenuErr</td>
<td>FixFontMenu never called</td>
</tr>
<tr>
<td>$1B0C</td>
<td>fmScaleSizeErr</td>
<td>Scaled size of font exceeds limits</td>
</tr>
</tbody>
</table>
In Appendix C, the interdependencies of the tool sets are listed, and the order in which the tool sets must be started up is given.

**Tool set dependencies**

The Tool Locator (tool set number $01$) does not depend on the presence of any of the other tool sets; rather, all of the other tool sets depend on the Tool Locator. Thus, Table C-1 begins with the Memory Manager (tool set number $02$) and continues in tool set number order.

*Note:* The dependencies given in Table C-1 differ in some minor respects from those given in the individual tool set chapters. Those in Table C-1 are the most current at the time of publication; they were up to date as of October 1, 1987. For any further updates on the dependencies between the tool sets, check Apple II GS Technical Note #12.
<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
<th>Minimum version needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01</td>
<td>Tool Locator</td>
<td>1.0</td>
</tr>
<tr>
<td>$02</td>
<td>Memory Manager</td>
<td>1.0</td>
</tr>
<tr>
<td>$03</td>
<td>Miscellaneous Tool Set</td>
<td>1.0</td>
</tr>
<tr>
<td>$04</td>
<td>QuickDraw II</td>
<td>1.0</td>
</tr>
<tr>
<td>$05</td>
<td>Desk Manager</td>
<td>1.2</td>
</tr>
<tr>
<td>$06</td>
<td>Event Manager</td>
<td>1.2</td>
</tr>
<tr>
<td>$07</td>
<td>Scheduler</td>
<td>1.0</td>
</tr>
<tr>
<td>$08</td>
<td>Sound Tool Set</td>
<td>1.0</td>
</tr>
<tr>
<td>$09</td>
<td>Apple Desktop Bus Tool Set</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Memory Manager (tool set number $02) depends on

Miscellaneous Tool Set (tool set number $03) depends on

QuickDraw II (tool set number $04) depends on

Desk Manager (tool set number $05) depends on

Event Manager (tool set number $06) depends on

Scheduler (tool set number $07) depends on

Sound Tool Set (tool set number $08) depends on

Apple Desktop Bus Tool Set (tool set number $09) depends on
<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
<th>Minimum version needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01</td>
<td>Tool Locator</td>
<td>1.0</td>
</tr>
<tr>
<td>$02</td>
<td>Memory Manager</td>
<td>1.0</td>
</tr>
<tr>
<td>$03</td>
<td>Miscellaneous Tool Set</td>
<td>1.2</td>
</tr>
<tr>
<td>$04</td>
<td>QuickDraw II</td>
<td>1.2</td>
</tr>
<tr>
<td>$06</td>
<td>Event Manager</td>
<td>1.0</td>
</tr>
<tr>
<td>$0E</td>
<td>Window Manager</td>
<td>1.3</td>
</tr>
<tr>
<td>$10</td>
<td>Control Manager</td>
<td>1.3</td>
</tr>
</tbody>
</table>

(continued)
### Table C-1 (continued)
#### Tool set dependencies

<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
<th>Minimum version needed</th>
</tr>
</thead>
</table>

**Print Manager (tool set number $13$) depends on**
- $01$ #01 Tool Locator 1.0
- $02$ #02 Memory Manager 2.0
- $03$ #03 Miscellaneous Tool Set 2.0
- $04$ #04 QuicdDraw II 2.0
- $06$ #06 Event Manager 1.0
- $0E$ #14 Window Manager 1.3
- $0F$ #15 Menu Manager 1.3
- $10$ #16 Control Manager 1.3
- $12$ #18 QuickDraw II Auxiliary 1.0
- $14$ #20 LineEdit Tool Set 1.0
- $15$ #21 Dialog Manager 1.1
- $1B$ #27 Font Manager 1.0
- $1C$ #28 List Manager 1.0

**LineEdit Tool Set (tool set number $14$) depends on**
- $01$ #01 Tool Locator 1.0
- $02$ #02 Memory Manager 1.0
- $03$ #03 Miscellaneous Tool Set 1.0
- $04$ #04 QuicdDraw II 1.1
- $06$ #06 Event Manager 1.0

*Note: If you are going to use the LEToScrap or LEFromScrap routines, the Scrap Manager must be loaded and started up. If you are going to use LETextBox2, you must load and start up the Integer Math Tool Set.*

**Dialog Manager (tool set number $15$) depends on**
- $01$ #01 Tool Locator 1.0
- $02$ #02 Memory Manager 1.0
- $03$ #03 Miscellaneous Tool Set 1.0
- $04$ #04 QuicdDraw II 1.0
- $06$ #06 Event Manager 1.0
- $0E$ #14 Window Manager 1.3
- $0F$ #15 Menu Manager 1.3
- $10$ #16 Control Manager 1.3
- $14$ #20 LineEdit Tool Set 1.0

**Scrap Manager (tool set number $16$) depends on**
- $01$ #01 Tool Locator 1.0
- $02$ #02 Memory Manager 1.0
### Table C-1 (continued)

#### Tool set dependencies

<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
<th>Minimum version needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01</td>
<td>#01 Tool Locator</td>
<td>1.0</td>
</tr>
<tr>
<td>$02</td>
<td>#02 Memory Manager</td>
<td>1.0</td>
</tr>
<tr>
<td>$03</td>
<td>#03 Miscellaneous Tool Set</td>
<td>1.0</td>
</tr>
<tr>
<td>$04</td>
<td>#04 QuickDraw II</td>
<td>1.0</td>
</tr>
<tr>
<td>$06</td>
<td>#06 Event Manager</td>
<td>1.0</td>
</tr>
<tr>
<td>$0E</td>
<td>#14 Window Manager</td>
<td>1.3</td>
</tr>
<tr>
<td>$10</td>
<td>#16 Control Manager</td>
<td>1.3</td>
</tr>
<tr>
<td>$0F</td>
<td>#15 Menu Manager</td>
<td>1.3</td>
</tr>
<tr>
<td>$14</td>
<td>#20 LineEdit Tool Set</td>
<td>1.0</td>
</tr>
<tr>
<td>$15</td>
<td>#21 Dialog Manager</td>
<td>1.1</td>
</tr>
</tbody>
</table>

#### Font Manager (tool set number $1B) depends on

<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
<th>Minimum version needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01</td>
<td>#01 Tool Locator</td>
<td>1.0</td>
</tr>
<tr>
<td>$02</td>
<td>#02 Memory Manager</td>
<td>1.0</td>
</tr>
<tr>
<td>$04</td>
<td>#04 QuickDraw II</td>
<td>1.1</td>
</tr>
</tbody>
</table>

*Note:* In addition to these tool sets, the ChooseFont routine requires

<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
<th>Minimum version needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$03</td>
<td>#03 Miscellaneous Tool Set</td>
<td>1.2</td>
</tr>
<tr>
<td>$0B</td>
<td>#11 Integer Math Tool Set</td>
<td>1.0</td>
</tr>
<tr>
<td>$0E</td>
<td>#14 Window Manager</td>
<td>1.3</td>
</tr>
<tr>
<td>$10</td>
<td>#16 Control Manager</td>
<td>1.3</td>
</tr>
<tr>
<td>$14</td>
<td>#20 LineEdit Tool Set</td>
<td>1.0</td>
</tr>
<tr>
<td>$15</td>
<td>#21 Dialog Manager</td>
<td>1.0</td>
</tr>
</tbody>
</table>

and the FixFontMenu routine requires

<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
<th>Minimum version needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0F</td>
<td>#15 Menu Manager</td>
<td>1.3</td>
</tr>
<tr>
<td>$1C</td>
<td>#28 List Manager</td>
<td>1.0</td>
</tr>
</tbody>
</table>

If you are using the shadowed, outlined, or underlined text styles, QuickDraw II Auxiliary (tool set number $12) must be loaded and started up.

#### List Manager (tool set number $1C) depends on

<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
<th>Minimum version needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01</td>
<td>#01 Tool Locator</td>
<td>1.0</td>
</tr>
<tr>
<td>$02</td>
<td>#02 Memory Manager</td>
<td>1.0</td>
</tr>
<tr>
<td>$03</td>
<td>#03 Miscellaneous Tool Set</td>
<td>1.0</td>
</tr>
<tr>
<td>$04</td>
<td>#04 QuickDraw II</td>
<td>1.0</td>
</tr>
<tr>
<td>$06</td>
<td>#06 Event Manager</td>
<td>1.0</td>
</tr>
<tr>
<td>$0E</td>
<td>#14 Window Manager</td>
<td>1.3</td>
</tr>
<tr>
<td>$10</td>
<td>#16 Control Manager</td>
<td>1.3</td>
</tr>
</tbody>
</table>
Tool set startup order

Because each tool set depends on the presence of other tool sets, certain tool sets must be started up in a prescribed order for others to work. This order is shown in Table C-2, with tool sets lower on the list depending on the presence of all the tool sets higher on the list. Thus, all tool sets from the Tool Locator through the Control Manager must be started up before the Menu Manager.

When you shut the tools down before you quit your application, you must shut them down in the reverse order from that in which they were started up; that is, the last one started up must be shut down first, the next-to-last started up shut down next, and so on.

*Note* The startup order given in Table C-2 differs in some minor respects from that given in Chapter 2. Use the order given in Table C-2. For further updates, refer to Apple II GS Technical Note #12.

Table C-2

<table>
<thead>
<tr>
<th>Tool set number</th>
<th>Tool set name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01</td>
<td>Tool Locator</td>
</tr>
<tr>
<td>$02</td>
<td>Memory Manager</td>
</tr>
<tr>
<td>$03</td>
<td>Miscellaneous Tool Set</td>
</tr>
<tr>
<td>$04</td>
<td>QuickDraw II</td>
</tr>
<tr>
<td>$06</td>
<td>Event Manager</td>
</tr>
<tr>
<td>$08</td>
<td>Window Manager</td>
</tr>
<tr>
<td>$10</td>
<td>Control Manager</td>
</tr>
<tr>
<td>$0F</td>
<td>Menu Manager</td>
</tr>
<tr>
<td>$14</td>
<td>LineEdit Tool Set</td>
</tr>
<tr>
<td>$15</td>
<td>Dialog Manager</td>
</tr>
<tr>
<td>$17</td>
<td>Standard File Operations Tool Set</td>
</tr>
<tr>
<td>$16</td>
<td>Scrap Manager</td>
</tr>
<tr>
<td>$05</td>
<td>Desk Manager</td>
</tr>
<tr>
<td>$1C</td>
<td>List Manager</td>
</tr>
<tr>
<td>$1B</td>
<td>Font Manager</td>
</tr>
<tr>
<td>$13</td>
<td>Print Manager</td>
</tr>
</tbody>
</table>

*Note* If you are using QuickDraw II Auxiliary, it must be started up after QuickDraw II.

You may assume that tool sets other than those listed in Table C-2 do not need to be started up in any particular order; that is, they may be started up or shut down at any time.
Appendix D

List of Routines by Tool Set Number and Routine Number

This appendix lists the tool set routines in tool set number and routine number order. The last two digits of the hexadecimal number comprise the tool set number; the first two digits comprise the routine number. Thus, the number \$0410 means the routine numbered \$04 in the tool set numbered \$10.

The list provided in this appendix can be useful, for example, when you are using a debugger and obtain the hexadecimal number, but not the name, of the routine. You can find the name by using this list and then looking up the name in the appropriate chapter.
<table>
<thead>
<tr>
<th>Number</th>
<th>Routine</th>
<th>Number</th>
<th>Routine</th>
<th>Number</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0101$</td>
<td>TLBootInit</td>
<td>$0103$</td>
<td>MTBootInit</td>
<td>$0103$</td>
<td>MTBootInit</td>
</tr>
<tr>
<td>$0201$</td>
<td>TLStartUp</td>
<td>$0203$</td>
<td>MTStartUp</td>
<td>$0203$</td>
<td>MTStartUp</td>
</tr>
<tr>
<td>$0301$</td>
<td>TLSHutDown</td>
<td>$0303$</td>
<td>MTShutDown</td>
<td>$0303$</td>
<td>MTShutDown</td>
</tr>
<tr>
<td>$0401$</td>
<td>TLVersion</td>
<td>$0403$</td>
<td>MTVersion</td>
<td>$0403$</td>
<td>MTVersion</td>
</tr>
<tr>
<td>$0501$</td>
<td>TLReset</td>
<td>$0503$</td>
<td>MTSip</td>
<td>$0503$</td>
<td>MTSip</td>
</tr>
<tr>
<td>$0601$</td>
<td>TLStatus</td>
<td>$0603$</td>
<td>MTSip</td>
<td>$0603$</td>
<td>MTSip</td>
</tr>
<tr>
<td>$0901$</td>
<td>GetTPtr</td>
<td>$0903$</td>
<td>WriteBRam</td>
<td>$0903$</td>
<td>WriteBRam</td>
</tr>
<tr>
<td>$0A01$</td>
<td>SetTPtr</td>
<td>$0A03$</td>
<td>ReadBRam</td>
<td>$0A03$</td>
<td>ReadBRam</td>
</tr>
<tr>
<td>$0B01$</td>
<td>GetFuncPtr</td>
<td>$0B03$</td>
<td>WriteBParam</td>
<td>$0B03$</td>
<td>WriteBParam</td>
</tr>
<tr>
<td>$0C01$</td>
<td>GetWAP</td>
<td>$0C03$</td>
<td>ReadBParam</td>
<td>$0C03$</td>
<td>ReadBParam</td>
</tr>
<tr>
<td>$0D01$</td>
<td>SetWAP</td>
<td>$0D03$</td>
<td>ReadTimeHex</td>
<td>$0D03$</td>
<td>ReadTimeHex</td>
</tr>
<tr>
<td>$0E01$</td>
<td>LoadTools</td>
<td>$0E03$</td>
<td>WriteTimeHex</td>
<td>$0E03$</td>
<td>WriteTimeHex</td>
</tr>
<tr>
<td>$0F01$</td>
<td>LoadOneTool</td>
<td>$0F03$</td>
<td>ReadAsciiTime</td>
<td>$0F03$</td>
<td>ReadAsciiTime</td>
</tr>
<tr>
<td>$1001$</td>
<td>UnloadOneTool</td>
<td>$1003$</td>
<td>SetVector</td>
<td>$1003$</td>
<td>SetVector</td>
</tr>
<tr>
<td>$1101$</td>
<td>TLMountVolume</td>
<td>$1103$</td>
<td>GetVector</td>
<td>$1103$</td>
<td>GetVector</td>
</tr>
<tr>
<td>$1201$</td>
<td>TLTextMountVolume</td>
<td>$1203$</td>
<td>SetHeartBeat</td>
<td>$1203$</td>
<td>SetHeartBeat</td>
</tr>
<tr>
<td>$1301$</td>
<td>SaveTextState</td>
<td>$1303$</td>
<td>DelHeartBeat</td>
<td>$1303$</td>
<td>DelHeartBeat</td>
</tr>
<tr>
<td>$1401$</td>
<td>RestoreTextState</td>
<td>$1403$</td>
<td>CtrlHeartBeat</td>
<td>$1403$</td>
<td>CtrlHeartBeat</td>
</tr>
<tr>
<td>$1501$</td>
<td>MessageCenter</td>
<td>$1503$</td>
<td>SysFailMgr</td>
<td>$1503$</td>
<td>SysFailMgr</td>
</tr>
</tbody>
</table>

### Miscellaneous Tool Set (Chapter 14)

<table>
<thead>
<tr>
<th>Number</th>
<th>Routine</th>
<th>Number</th>
<th>Routine</th>
<th>Number</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0101$</td>
<td>TLBootInit</td>
<td>$0103$</td>
<td>MTBootInit</td>
<td>$0103$</td>
<td>MTBootInit</td>
</tr>
<tr>
<td>$0201$</td>
<td>TLStartUp</td>
<td>$0203$</td>
<td>MTStartUp</td>
<td>$0203$</td>
<td>MTStartUp</td>
</tr>
<tr>
<td>$0301$</td>
<td>TLSHutDown</td>
<td>$0303$</td>
<td>MTShutDown</td>
<td>$0303$</td>
<td>MTShutDown</td>
</tr>
<tr>
<td>$0401$</td>
<td>TLVersion</td>
<td>$0403$</td>
<td>MTVersion</td>
<td>$0403$</td>
<td>MTVersion</td>
</tr>
<tr>
<td>$0501$</td>
<td>TLReset</td>
<td>$0503$</td>
<td>MTSip</td>
<td>$0503$</td>
<td>MTSip</td>
</tr>
<tr>
<td>$0601$</td>
<td>TLStatus</td>
<td>$0603$</td>
<td>MTSip</td>
<td>$0603$</td>
<td>MTSip</td>
</tr>
<tr>
<td>$0901$</td>
<td>GetTPtr</td>
<td>$0903$</td>
<td>WriteBRam</td>
<td>$0903$</td>
<td>WriteBRam</td>
</tr>
<tr>
<td>$0A01$</td>
<td>SetTPtr</td>
<td>$0A03$</td>
<td>ReadBRam</td>
<td>$0A03$</td>
<td>ReadBRam</td>
</tr>
<tr>
<td>$0B01$</td>
<td>GetFuncPtr</td>
<td>$0B03$</td>
<td>WriteBParam</td>
<td>$0B03$</td>
<td>WriteBParam</td>
</tr>
<tr>
<td>$0C01$</td>
<td>GetWAP</td>
<td>$0C03$</td>
<td>ReadBParam</td>
<td>$0C03$</td>
<td>ReadBParam</td>
</tr>
<tr>
<td>$0D01$</td>
<td>SetWAP</td>
<td>$0D03$</td>
<td>ReadTimeHex</td>
<td>$0D03$</td>
<td>ReadTimeHex</td>
</tr>
<tr>
<td>$0E01$</td>
<td>LoadTools</td>
<td>$0E03$</td>
<td>WriteTimeHex</td>
<td>$0E03$</td>
<td>WriteTimeHex</td>
</tr>
<tr>
<td>$0F01$</td>
<td>LoadOneTool</td>
<td>$0F03$</td>
<td>ReadAsciiTime</td>
<td>$0F03$</td>
<td>ReadAsciiTime</td>
</tr>
<tr>
<td>$1001$</td>
<td>UnloadOneTool</td>
<td>$1003$</td>
<td>SetVector</td>
<td>$1003$</td>
<td>SetVector</td>
</tr>
<tr>
<td>$1101$</td>
<td>TLMountVolume</td>
<td>$1103$</td>
<td>GetVector</td>
<td>$1103$</td>
<td>GetVector</td>
</tr>
<tr>
<td>$1201$</td>
<td>TLTextMountVolume</td>
<td>$1203$</td>
<td>SetHeartBeat</td>
<td>$1203$</td>
<td>SetHeartBeat</td>
</tr>
<tr>
<td>$1301$</td>
<td>SaveTextState</td>
<td>$1303$</td>
<td>DelHeartBeat</td>
<td>$1303$</td>
<td>DelHeartBeat</td>
</tr>
<tr>
<td>$1401$</td>
<td>RestoreTextState</td>
<td>$1403$</td>
<td>CtrlHeartBeat</td>
<td>$1403$</td>
<td>CtrlHeartBeat</td>
</tr>
<tr>
<td>$1501$</td>
<td>MessageCenter</td>
<td>$1503$</td>
<td>SysFailMgr</td>
<td>$1503$</td>
<td>SysFailMgr</td>
</tr>
</tbody>
</table>

### QuickDraw II (Chapter 16)

<table>
<thead>
<tr>
<th>Number</th>
<th>Routine</th>
<th>Number</th>
<th>Routine</th>
<th>Number</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0101$</td>
<td>QDBootInit</td>
<td>$0103$</td>
<td>MTBootInit</td>
<td>$0103$</td>
<td>MTBootInit</td>
</tr>
<tr>
<td>$0201$</td>
<td>QDStartUp</td>
<td>$0203$</td>
<td>MTStartUp</td>
<td>$0203$</td>
<td>MTStartUp</td>
</tr>
<tr>
<td>$0301$</td>
<td>QDSHutDown</td>
<td>$0303$</td>
<td>MTShutDown</td>
<td>$0303$</td>
<td>MTShutDown</td>
</tr>
<tr>
<td>$0401$</td>
<td>QDVersion</td>
<td>$0403$</td>
<td>MTVersion</td>
<td>$0403$</td>
<td>MTVersion</td>
</tr>
<tr>
<td>$0501$</td>
<td>QDReset</td>
<td>$0503$</td>
<td>MTSip</td>
<td>$0503$</td>
<td>MTSip</td>
</tr>
<tr>
<td>$0601$</td>
<td>QDStatus</td>
<td>$0603$</td>
<td>MTSip</td>
<td>$0603$</td>
<td>MTSip</td>
</tr>
<tr>
<td>$0901$</td>
<td>GetAddress</td>
<td>$0903$</td>
<td>WriteBRam</td>
<td>$0903$</td>
<td>WriteBRam</td>
</tr>
<tr>
<td>$0A01$</td>
<td>GrafoN</td>
<td>$0A03$</td>
<td>ReadBRam</td>
<td>$0A03$</td>
<td>ReadBRam</td>
</tr>
<tr>
<td>$0B01$</td>
<td>GrafoF</td>
<td>$0B03$</td>
<td>WriteBParam</td>
<td>$0B03$</td>
<td>WriteBParam</td>
</tr>
<tr>
<td>$0C01$</td>
<td>GetStandardSCB</td>
<td>$0C03$</td>
<td>ReadBParam</td>
<td>$0C03$</td>
<td>ReadBParam</td>
</tr>
<tr>
<td>$0D01$</td>
<td>InitColorTable</td>
<td>$0D03$</td>
<td>ReadTimeHex</td>
<td>$0D03$</td>
<td>ReadTimeHex</td>
</tr>
<tr>
<td>$0E01$</td>
<td>SetColorTable</td>
<td>$0E03$</td>
<td>WriteTimeHex</td>
<td>$0E03$</td>
<td>WriteTimeHex</td>
</tr>
<tr>
<td>$0F01$</td>
<td>GetColorTable</td>
<td>$0F03$</td>
<td>ReadAsciiTime</td>
<td>$0F03$</td>
<td>ReadAsciiTime</td>
</tr>
<tr>
<td>$1001$</td>
<td>SetColorEntry</td>
<td>$1003$</td>
<td>SetVector</td>
<td>$1003$</td>
<td>SetVector</td>
</tr>
<tr>
<td>$1101$</td>
<td>GetColorEntry</td>
<td>$1103$</td>
<td>GetVector</td>
<td>$1103$</td>
<td>GetVector</td>
</tr>
<tr>
<td>$1201$</td>
<td>SetSCB</td>
<td>$1203$</td>
<td>SetHeartBeat</td>
<td>$1203$</td>
<td>SetHeartBeat</td>
</tr>
<tr>
<td>$1301$</td>
<td>GetSCB</td>
<td>$1303$</td>
<td>DelHeartBeat</td>
<td>$1303$</td>
<td>DelHeartBeat</td>
</tr>
<tr>
<td>$1401$</td>
<td>SetAllSCBs</td>
<td>$1403$</td>
<td>CtrlHeartBeat</td>
<td>$1403$</td>
<td>CtrlHeartBeat</td>
</tr>
<tr>
<td>$1501$</td>
<td>ClearScreen</td>
<td>$1503$</td>
<td>SysFailMgr</td>
<td>$1503$</td>
<td>SysFailMgr</td>
</tr>
<tr>
<td>$1601$</td>
<td>SetMasterSCB</td>
<td>$1603$</td>
<td>HomeMouse</td>
<td>$1603$</td>
<td>HomeMouse</td>
</tr>
<tr>
<td>$1701$</td>
<td>GetMasterSCB</td>
<td>$1703$</td>
<td>ClearMouse</td>
<td>$1703$</td>
<td>ClearMouse</td>
</tr>
<tr>
<td>$1801$</td>
<td>OpenPort</td>
<td>$1803$</td>
<td>InitMouse</td>
<td>$1803$</td>
<td>InitMouse</td>
</tr>
<tr>
<td>$1901$</td>
<td>InitPort</td>
<td>$1903$</td>
<td>ClampMouse</td>
<td>$1903$</td>
<td>ClampMouse</td>
</tr>
<tr>
<td>Number</td>
<td>Routine</td>
<td>Number</td>
<td>Routine</td>
<td>Number</td>
<td>Routine</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>--------</td>
<td>---------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>$4104</td>
<td>GetRgnSave</td>
<td>$6904</td>
<td>CopyRgn</td>
<td>$6A04</td>
<td>SetEmptyRgn</td>
</tr>
<tr>
<td>$4204</td>
<td>SetPolySave</td>
<td>$6B04</td>
<td>SetRectRgn</td>
<td>$6C04</td>
<td>RectRgn</td>
</tr>
<tr>
<td>$4304</td>
<td>GetPolySave</td>
<td>$6D04</td>
<td>OpenRgn</td>
<td>$6E04</td>
<td>CloseRgn</td>
</tr>
<tr>
<td>$4404</td>
<td>SetGrafProcs</td>
<td>$6F04</td>
<td>OffsetRgn</td>
<td>$7004</td>
<td>InsetRgn</td>
</tr>
<tr>
<td>$4504</td>
<td>GetGrafProcs</td>
<td>$7104</td>
<td>SetRgn</td>
<td>$7204</td>
<td>UnionRgn</td>
</tr>
<tr>
<td>$4604</td>
<td>SetUserField</td>
<td>$7304</td>
<td>DiffRgn</td>
<td>$7404</td>
<td>XorRgn</td>
</tr>
<tr>
<td>$4704</td>
<td>GetUserField</td>
<td>$7504</td>
<td>PtInRgn</td>
<td>$7604</td>
<td>RectInRgn</td>
</tr>
<tr>
<td>$4804</td>
<td>SetSysField</td>
<td>$7704</td>
<td>EqualRgn</td>
<td>$7804</td>
<td>EmptyRgn</td>
</tr>
<tr>
<td>$4904</td>
<td>GetSysField</td>
<td>$7904</td>
<td>FrameRgn</td>
<td>$8004</td>
<td>AddPt</td>
</tr>
<tr>
<td>$4A04</td>
<td>SetRect</td>
<td>$8104</td>
<td>SubPt</td>
<td>$8204</td>
<td>SetPt</td>
</tr>
<tr>
<td>$4B04</td>
<td>GetRect</td>
<td>$8304</td>
<td>EqualPt</td>
<td>$8404</td>
<td>LocalToGlobal</td>
</tr>
<tr>
<td>$4C04</td>
<td>InsetRect</td>
<td>$8504</td>
<td>GlobalToLocal</td>
<td>$8604</td>
<td>Random</td>
</tr>
<tr>
<td>$4D04</td>
<td>SectRect</td>
<td>$8704</td>
<td>SetRandSeed</td>
<td>$8804</td>
<td>GetPixel</td>
</tr>
<tr>
<td>$4E04</td>
<td>UnionRect</td>
<td>$8904</td>
<td>ScalePt</td>
<td>$8A04</td>
<td>MapPt</td>
</tr>
<tr>
<td>$4F04</td>
<td>PtInRect</td>
<td>$8B04</td>
<td>MapRect</td>
<td>$8C04</td>
<td>MapRgn</td>
</tr>
<tr>
<td>$5004</td>
<td>Pt2Rect</td>
<td>$8D04</td>
<td>SetStdProcs</td>
<td>$8E04</td>
<td>SetCursor</td>
</tr>
<tr>
<td>$5104</td>
<td>EqualRect</td>
<td>$8F04</td>
<td>GetCursorAdr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5204</td>
<td>EmptyRect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5304</td>
<td>NotEmptyRect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5404</td>
<td>PaintRect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5504</td>
<td>EraseRect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5604</td>
<td>InvertRect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5704</td>
<td>FillRect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5804</td>
<td>FrameOval</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5904</td>
<td>PaintOval</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5A04</td>
<td>EraseOval</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5B04</td>
<td>InvertOval</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5C04</td>
<td>FillOval</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5D04</td>
<td>FrameRRect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5E04</td>
<td>PaintRRect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5F04</td>
<td>EraseRRect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$6004</td>
<td>InvertRRect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$6104</td>
<td>FillRRect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$6204</td>
<td>FrameArc</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$6304</td>
<td>PaintArc</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$6404</td>
<td>EraseArc</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$6504</td>
<td>InvertArc</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$6604</td>
<td>FillArc</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$6704</td>
<td>NewRgn</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$6804</td>
<td>DisposeRgn</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(continued)
Table D-1 (continued)
Routines by tool set/routine number

<table>
<thead>
<tr>
<th>Number</th>
<th>Routine</th>
<th>Number</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B704</td>
<td>OpenPicture (Ch. 17)</td>
<td>$B804</td>
<td>PicComment (Ch. 17)</td>
</tr>
<tr>
<td>$B904</td>
<td>ClosePicture (Ch. 17)</td>
<td>$B904</td>
<td>DrawPicture (Ch. 17)</td>
</tr>
<tr>
<td>$B904</td>
<td>KillPicture (Ch. 17)</td>
<td>$B904</td>
<td>FramePoly</td>
</tr>
<tr>
<td>$B904</td>
<td>PaintPoly</td>
<td>$B904</td>
<td>InvertPoly</td>
</tr>
<tr>
<td>$B904</td>
<td>FillPoly</td>
<td>$B904</td>
<td>PaintPoly</td>
</tr>
<tr>
<td>$B904</td>
<td>OpenPoly</td>
<td>$B904</td>
<td>OpenPoly</td>
</tr>
<tr>
<td>$B904</td>
<td>ClosePoly</td>
<td>$B904</td>
<td>OpenPoly</td>
</tr>
<tr>
<td>$B904</td>
<td>KillPoly</td>
<td>$B904</td>
<td>ClosePoly</td>
</tr>
<tr>
<td>$B904</td>
<td>FramePoly</td>
<td>$B904</td>
<td>FramePoly</td>
</tr>
<tr>
<td>$B904</td>
<td>PaintPoly</td>
<td>$B904</td>
<td>PaintPoly</td>
</tr>
<tr>
<td>$B904</td>
<td>OpenPoly</td>
<td>$B904</td>
<td>OpenPoly</td>
</tr>
<tr>
<td>$B904</td>
<td>ClosePoly</td>
<td>$B904</td>
<td>ClosePoly</td>
</tr>
<tr>
<td>$B904</td>
<td>KillPoly</td>
<td>$B904</td>
<td>KillPoly</td>
</tr>
<tr>
<td>$B904</td>
<td>FramePoly</td>
<td>$B904</td>
<td>FramePoly</td>
</tr>
<tr>
<td>$B904</td>
<td>PaintPoly</td>
<td>$B904</td>
<td>PaintPoly</td>
</tr>
<tr>
<td>$B904</td>
<td>OpenPoly</td>
<td>$B904</td>
<td>OpenPoly</td>
</tr>
<tr>
<td>$B904</td>
<td>ClosePoly</td>
<td>$B904</td>
<td>ClosePoly</td>
</tr>
<tr>
<td>$B904</td>
<td>KillPoly</td>
<td>$B904</td>
<td>KillPoly</td>
</tr>
<tr>
<td>$B904</td>
<td>FramePoly</td>
<td>$B904</td>
<td>FramePoly</td>
</tr>
<tr>
<td>$B904</td>
<td>PaintPoly</td>
<td>$B904</td>
<td>PaintPoly</td>
</tr>
</tbody>
</table>

D-4 Appendix D: List of Routines by Tool Set Number and Routine Number
<table>
<thead>
<tr>
<th>Number</th>
<th>Routine</th>
<th>Number</th>
<th>Routine</th>
<th>Number</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0106</td>
<td>EMBootinit</td>
<td>$0108</td>
<td>SoundBootinit</td>
<td>$010A</td>
<td>SANEBootinit</td>
</tr>
<tr>
<td>$0206</td>
<td>EMStartUp</td>
<td>$0208</td>
<td>SoundStartUp</td>
<td>$020A</td>
<td>SANESTartUp</td>
</tr>
<tr>
<td>$0306</td>
<td>EMShutDown</td>
<td>$0308</td>
<td>SoundShutDown</td>
<td>$030A</td>
<td>SANEShutDown</td>
</tr>
<tr>
<td>$0406</td>
<td>EMVersion</td>
<td>$0408</td>
<td>SoundVersion</td>
<td>$040A</td>
<td>SANEVersion</td>
</tr>
<tr>
<td>$0506</td>
<td>EMReset</td>
<td>$0508</td>
<td>SoundReset</td>
<td>$050A</td>
<td>SANEReset</td>
</tr>
<tr>
<td>$0606</td>
<td>EMStatus</td>
<td>$0608</td>
<td>SoundToolStatus</td>
<td>$060A</td>
<td>SANEStatus</td>
</tr>
<tr>
<td>$0906</td>
<td>DoWindows</td>
<td>$0908</td>
<td>WriteRamBlock</td>
<td>$090A</td>
<td>SANEFP816</td>
</tr>
<tr>
<td>$0A06</td>
<td>GetNextEvent</td>
<td>$0A08</td>
<td>ReadRamBlock</td>
<td>$0A0A</td>
<td>SANEDecStr816</td>
</tr>
<tr>
<td>$0B06</td>
<td>EventAvail</td>
<td>$0B08</td>
<td>GetTableAddress</td>
<td>$0B0A</td>
<td>SANEElems816</td>
</tr>
<tr>
<td>$0C06</td>
<td>GetMouse</td>
<td>$0C08</td>
<td>GetSoundVolume</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0D06</td>
<td>Button</td>
<td>$0D08</td>
<td>SetSoundVolume</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0E06</td>
<td>StillDown</td>
<td>$0E08</td>
<td>FFStartSound</td>
<td>$0E0B</td>
<td>IMBootinit</td>
</tr>
<tr>
<td>$0F06</td>
<td>WaitMouseDown</td>
<td>$0F08</td>
<td>FFStopSound</td>
<td>$0F0B</td>
<td>IMStartUp</td>
</tr>
<tr>
<td>$1006</td>
<td>TickCount</td>
<td>$1008</td>
<td>FFSoundStatus</td>
<td>$0F0B</td>
<td>IMShutDown</td>
</tr>
<tr>
<td>$1106</td>
<td>GetDbtTime</td>
<td>$1108</td>
<td>FFGGeneratorStatus</td>
<td>$0F0B</td>
<td>IMVersion</td>
</tr>
<tr>
<td>$1206</td>
<td>GetCaretTime</td>
<td>$1208</td>
<td>SetSoundMIQRV</td>
<td>$0F0B</td>
<td>IMReset</td>
</tr>
<tr>
<td>$1306</td>
<td>SetSwitch</td>
<td>$1308</td>
<td>SetUserSoundRQV</td>
<td>$0F0B</td>
<td>IMStatus</td>
</tr>
<tr>
<td>$1406</td>
<td>PostEvent</td>
<td>$1408</td>
<td>FFSoundDoneStatus</td>
<td>$0F0B</td>
<td>Multiply</td>
</tr>
<tr>
<td>$1506</td>
<td>FlushEvents</td>
<td></td>
<td></td>
<td>$0A0B</td>
<td>SDivide</td>
</tr>
<tr>
<td>$1606</td>
<td>GetOSEvent</td>
<td></td>
<td></td>
<td>$0B0B</td>
<td>UDivide</td>
</tr>
<tr>
<td>$1706</td>
<td>OSEventAvail</td>
<td>$0109</td>
<td>ADBBootInit</td>
<td>$0B0B</td>
<td>LongDivide</td>
</tr>
<tr>
<td>$1806</td>
<td>SetEventMask</td>
<td>$0209</td>
<td>ADBStartUp</td>
<td>$0C0B</td>
<td>LongMul</td>
</tr>
<tr>
<td>$1906</td>
<td>FakeMouse</td>
<td>$0309</td>
<td>ADBShutDown</td>
<td>$0D0B</td>
<td>LongDivide</td>
</tr>
<tr>
<td>$0107</td>
<td>SchBootInit</td>
<td>$0409</td>
<td>ADBVersion</td>
<td>$0E0B</td>
<td>FixRatio</td>
</tr>
<tr>
<td>$0207</td>
<td>SchStartUp</td>
<td>$0509</td>
<td>ADBReset</td>
<td>$0F0B</td>
<td>FixMul</td>
</tr>
<tr>
<td>$0307</td>
<td>SchShutDown</td>
<td>$0609</td>
<td>ADBStatus</td>
<td>$100B</td>
<td>FracMul</td>
</tr>
<tr>
<td>$0407</td>
<td>SchVersion</td>
<td>$0909</td>
<td>SendInfo</td>
<td>$110B</td>
<td>FixDiv</td>
</tr>
<tr>
<td>$0507</td>
<td>SchReset</td>
<td>$0A09</td>
<td>ReadKeyMicroData</td>
<td>$120B</td>
<td>FracDiv</td>
</tr>
<tr>
<td>$0607</td>
<td>SchStatus</td>
<td>$0B09</td>
<td>ReadKeyMicroMem</td>
<td>$130B</td>
<td>FixRound</td>
</tr>
<tr>
<td>$0907</td>
<td>SchAddTask</td>
<td>$0D09</td>
<td>AsyncADBAcReceive</td>
<td>$140B</td>
<td>FracSqrt</td>
</tr>
<tr>
<td>$0A07</td>
<td>SchFlush</td>
<td>$0E09</td>
<td>SyncADBAcReceive</td>
<td>$150B</td>
<td>FracCos</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$0F09</td>
<td>AbsOn</td>
<td>$160B</td>
<td>FracSin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1009</td>
<td>AbsOff</td>
<td>$170B</td>
<td>FixATan2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1109</td>
<td>ReadAbs</td>
<td>$180B</td>
<td>HiWord</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1209</td>
<td>GetAbsScale</td>
<td>$190B</td>
<td>LoWord</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1309</td>
<td>SetAbsScale</td>
<td>$1A0B</td>
<td>Long2Fix</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1409</td>
<td>SRQPoll</td>
<td>$1B0B</td>
<td>Fix2Long</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1509</td>
<td>SRQRemove</td>
<td>$1C0B</td>
<td>Fix2Frac</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1609</td>
<td>ClearSRQTTable</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(continued)
### Table D-1 (continued)
Routines by tool set/routine number

<table>
<thead>
<tr>
<th>Number</th>
<th>Routine</th>
<th>Number</th>
<th>Routine</th>
<th>Number</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Integer Math Tool Set</strong> (Chapter 9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$10D0B</td>
<td>Frac2Fix</td>
<td>$110C</td>
<td>WriteLine</td>
<td>$1F0E</td>
<td>EndUpdate</td>
</tr>
<tr>
<td>$1E0B</td>
<td>Fix2X</td>
<td>$120C</td>
<td>ErrWriteLine</td>
<td>$200E</td>
<td>GetWMgrPort</td>
</tr>
<tr>
<td>$1F0B</td>
<td>Frac2X</td>
<td>$130C</td>
<td>WriteString</td>
<td>$210E</td>
<td>PinRect</td>
</tr>
<tr>
<td>$200B</td>
<td>X2Fix</td>
<td>$140C</td>
<td>ErrWriteString</td>
<td>$220E</td>
<td>HiliteWindow</td>
</tr>
<tr>
<td>$210B</td>
<td>X2Frac</td>
<td>$150C</td>
<td>TextWriteBlock</td>
<td>$230E</td>
<td>ShowHide</td>
</tr>
<tr>
<td>$220B</td>
<td>Int2Hex</td>
<td>$160C</td>
<td>ErrWriteBlock</td>
<td>$240E</td>
<td>BringToFront</td>
</tr>
<tr>
<td>$230B</td>
<td>Long2Hex</td>
<td>$170C</td>
<td>WriteString</td>
<td>$250E</td>
<td>WindNewRes</td>
</tr>
<tr>
<td>$240B</td>
<td>Hex2Int</td>
<td>$180C</td>
<td>ReadChar</td>
<td>$260E</td>
<td>TrackZoom</td>
</tr>
<tr>
<td>$250B</td>
<td>Hex2Long</td>
<td>$190C</td>
<td>TextWriteBlock</td>
<td>$270E</td>
<td>ZoomWindow</td>
</tr>
<tr>
<td>$260B</td>
<td>Int2Dec</td>
<td></td>
<td></td>
<td>$280E</td>
<td>SetWRefCon</td>
</tr>
<tr>
<td>$270B</td>
<td>Long2Dec</td>
<td></td>
<td></td>
<td>$290E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td>$280B</td>
<td>Dec2Int</td>
<td></td>
<td></td>
<td>$2A0E</td>
<td>GetNextWindow</td>
</tr>
<tr>
<td>$290B</td>
<td>Dec2Long</td>
<td></td>
<td></td>
<td>$2B0E</td>
<td>GetWKind</td>
</tr>
<tr>
<td>$2A0B</td>
<td>HexIt</td>
<td></td>
<td></td>
<td>$2C0E</td>
<td>GetWFrame</td>
</tr>
<tr>
<td><strong>Window Manager</strong> (Chapter 25)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$010E</td>
<td>WindBootInit</td>
<td>$020E</td>
<td>WindStartUp</td>
<td>$030E</td>
<td>GetWKind</td>
</tr>
<tr>
<td>$040C</td>
<td>TextVersion</td>
<td>$050E</td>
<td>WindVersion</td>
<td>$060E</td>
<td>GetWFrame</td>
</tr>
<tr>
<td>$050C</td>
<td>TextStatus</td>
<td></td>
<td></td>
<td>$070E</td>
<td>GetWControls</td>
</tr>
<tr>
<td>$060C</td>
<td>TextReset</td>
<td></td>
<td></td>
<td>$080E</td>
<td>SetWRefCon</td>
</tr>
<tr>
<td>$070C</td>
<td>GetInGlobals</td>
<td></td>
<td></td>
<td>$090E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td>SetOutGlobals</td>
<td></td>
<td></td>
<td>$100E</td>
<td>UpdateRgn</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$110E</td>
<td>SetUpdateRgn</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$120E</td>
<td>SetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$130E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$140E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$150E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$160E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$170E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$180E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$190E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$200E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$210E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$220E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$230E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$240E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$250E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$260E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$270E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$280E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$290E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$300E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$310E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$320E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$330E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$340E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$350E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$360E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$370E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$380E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$390E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$400E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$410E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$420E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$430E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$440E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$450E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$460E</td>
<td>GetWRefCon</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$470E</td>
<td>GetWRefCon</td>
</tr>
</tbody>
</table>

D-6 Appendix D: List of Routines by Tool Set Number and Routine Number
<table>
<thead>
<tr>
<th>Number</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$480E</td>
<td>GetContentDraw</td>
</tr>
<tr>
<td>$490E</td>
<td>SetContentView</td>
</tr>
<tr>
<td>$4A0E</td>
<td>GetInfoDraw</td>
</tr>
<tr>
<td>$4B0E</td>
<td>SetSysWindow</td>
</tr>
<tr>
<td>$4D0E</td>
<td>StartDrawing</td>
</tr>
<tr>
<td>$4E0E</td>
<td>SetWindowIcons</td>
</tr>
<tr>
<td>$4F0E</td>
<td>GetRectInfo</td>
</tr>
<tr>
<td>$500E</td>
<td>StartInfoDrawing</td>
</tr>
<tr>
<td>$510E</td>
<td>EndInfoDrawing</td>
</tr>
<tr>
<td>$520E</td>
<td>GetFirstWindow</td>
</tr>
<tr>
<td>$530E</td>
<td>WindDragRect</td>
</tr>
<tr>
<td>$560E</td>
<td>WindowGlobal</td>
</tr>
<tr>
<td>$570E</td>
<td>SetContentOrigin2</td>
</tr>
<tr>
<td>$580E</td>
<td>WindowGlobal</td>
</tr>
<tr>
<td>$590E</td>
<td>SetContent2</td>
</tr>
<tr>
<td>$5A0F</td>
<td>GetSysBar</td>
</tr>
<tr>
<td>$5B0F</td>
<td>FixMenuBar</td>
</tr>
<tr>
<td>$5C0F</td>
<td>CountMitems</td>
</tr>
<tr>
<td>$5D0F</td>
<td>NewMenuBar</td>
</tr>
<tr>
<td>$5E0F</td>
<td>DeleteMenu</td>
</tr>
<tr>
<td>$5F0F</td>
<td>InsertMItem</td>
</tr>
<tr>
<td>$600F</td>
<td>DeleteMItem</td>
</tr>
<tr>
<td>$610F</td>
<td>GetSysBar</td>
</tr>
<tr>
<td>$620F</td>
<td>SetSysBar</td>
</tr>
<tr>
<td>$630F</td>
<td>FixMenuBar</td>
</tr>
<tr>
<td>$640F</td>
<td>CountMitems</td>
</tr>
<tr>
<td>$650F</td>
<td>NewMenuBar</td>
</tr>
</tbody>
</table>

**Window Manager** (Chapter 13)

<table>
<thead>
<tr>
<th>Number</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$160F</td>
<td>GetMHandle</td>
</tr>
<tr>
<td>$170F</td>
<td>SetBarColors</td>
</tr>
<tr>
<td>$180F</td>
<td>GetBarColors</td>
</tr>
<tr>
<td>$190F</td>
<td>SetMTitleStart</td>
</tr>
<tr>
<td>$1A0F</td>
<td>GetMTitleStart</td>
</tr>
<tr>
<td>$1B0F</td>
<td>GetMenuMgrPort</td>
</tr>
<tr>
<td>$1C0F</td>
<td>CalcMenuSize</td>
</tr>
<tr>
<td>$1D0F</td>
<td>SetMTitleWidth</td>
</tr>
<tr>
<td>$1E0F</td>
<td>GetMTitleWidth</td>
</tr>
<tr>
<td>$1F0F</td>
<td>SetMenuFlag</td>
</tr>
<tr>
<td>$200F</td>
<td>GetMenuFlag</td>
</tr>
<tr>
<td>$210F</td>
<td>SetMenuTitle</td>
</tr>
<tr>
<td>$220F</td>
<td>GetMenuTitle</td>
</tr>
<tr>
<td>$230F</td>
<td>MenuGlobal</td>
</tr>
<tr>
<td>$240F</td>
<td>SetMItem</td>
</tr>
<tr>
<td>$250F</td>
<td>GetMItem</td>
</tr>
<tr>
<td>$260F</td>
<td>SetMItemFlag</td>
</tr>
<tr>
<td>$270F</td>
<td>GetMItemFlag</td>
</tr>
<tr>
<td>$280F</td>
<td>SetMItemBlink</td>
</tr>
<tr>
<td>$290F</td>
<td>MenuNewRes</td>
</tr>
<tr>
<td>$2A0F</td>
<td>DrawMenuBar</td>
</tr>
<tr>
<td>$2B0F</td>
<td>MenuSelect</td>
</tr>
<tr>
<td>$2C0F</td>
<td>HillieMenu</td>
</tr>
<tr>
<td>$2D0F</td>
<td>NewMenu</td>
</tr>
<tr>
<td>$2E0F</td>
<td>DisposeMenu</td>
</tr>
<tr>
<td>$2F0F</td>
<td>InitPalette</td>
</tr>
<tr>
<td>$300F</td>
<td>EnableMItem</td>
</tr>
<tr>
<td>$310F</td>
<td>DisableMItem</td>
</tr>
<tr>
<td>$320F</td>
<td>CheckMItem</td>
</tr>
<tr>
<td>$330F</td>
<td>SetMItemMark</td>
</tr>
<tr>
<td>$340F</td>
<td>GetMItemMark</td>
</tr>
<tr>
<td>$350F</td>
<td>SetMItemStyle</td>
</tr>
<tr>
<td>$360F</td>
<td>GetMItemStyle</td>
</tr>
<tr>
<td>$370F</td>
<td>SetMenuID</td>
</tr>
<tr>
<td>$380F</td>
<td>SetMItemID</td>
</tr>
<tr>
<td>$390F</td>
<td>SetMenuBar</td>
</tr>
<tr>
<td>$3A0F</td>
<td>SetMItemName</td>
</tr>
</tbody>
</table>

**Menu Manager** (Chapter 25)

<table>
<thead>
<tr>
<th>Number</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$010F</td>
<td>MenuBootInit</td>
</tr>
<tr>
<td>$020F</td>
<td>MenuStartUp</td>
</tr>
<tr>
<td>$030F</td>
<td>MenuShutDown</td>
</tr>
<tr>
<td>$040F</td>
<td>MenuVersion</td>
</tr>
<tr>
<td>$050F</td>
<td>MenuReset</td>
</tr>
<tr>
<td>$060F</td>
<td>MenuStatus</td>
</tr>
<tr>
<td>$090F</td>
<td>MenuKey</td>
</tr>
<tr>
<td>$0A0F</td>
<td>GetMenuBar</td>
</tr>
<tr>
<td>$0B0F</td>
<td>MenuRefresh</td>
</tr>
<tr>
<td>$0C0F</td>
<td>FlashMenuBar</td>
</tr>
<tr>
<td>$0D0F</td>
<td>InsertMenu</td>
</tr>
<tr>
<td>$0E0F</td>
<td>DeleteMenu</td>
</tr>
<tr>
<td>$0F0F</td>
<td>InsertMItem</td>
</tr>
<tr>
<td>$100F</td>
<td>DeleteMItem</td>
</tr>
<tr>
<td>$110F</td>
<td>GetSysBar</td>
</tr>
<tr>
<td>$120F</td>
<td>SetSysBar</td>
</tr>
<tr>
<td>$130F</td>
<td>FixMenuBar</td>
</tr>
<tr>
<td>$140F</td>
<td>CountMitems</td>
</tr>
<tr>
<td>$150F</td>
<td>NewMenuBar</td>
</tr>
</tbody>
</table>

**Control Manager** (Chapter 4)

<table>
<thead>
<tr>
<th>Number</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0110</td>
<td>CtlBootInit</td>
</tr>
<tr>
<td>$0210</td>
<td>CtlStartUp</td>
</tr>
<tr>
<td>$0310</td>
<td>CtlShutDown</td>
</tr>
<tr>
<td>$0410</td>
<td>CtlVersion</td>
</tr>
<tr>
<td>$0510</td>
<td>CtlReset</td>
</tr>
<tr>
<td>$0610</td>
<td>CtlStatus</td>
</tr>
<tr>
<td>$0910</td>
<td>NewControl</td>
</tr>
<tr>
<td>$0A10</td>
<td>DisposeControl</td>
</tr>
<tr>
<td>$0B10</td>
<td>KillControls</td>
</tr>
<tr>
<td>$0C10</td>
<td>SetCtlTitle</td>
</tr>
<tr>
<td>$0D10</td>
<td>GetCtlTitle</td>
</tr>
<tr>
<td>$0E10</td>
<td>HideControl</td>
</tr>
<tr>
<td>$0F10</td>
<td>ShowControl</td>
</tr>
<tr>
<td>$1010</td>
<td>DrawControls</td>
</tr>
<tr>
<td>$1110</td>
<td>HilliteControl</td>
</tr>
<tr>
<td>$1210</td>
<td>CtlNewRes</td>
</tr>
<tr>
<td>$1310</td>
<td>FindControl</td>
</tr>
<tr>
<td>$1410</td>
<td>TestControl</td>
</tr>
<tr>
<td>$1510</td>
<td>TrackControl</td>
</tr>
<tr>
<td>$1610</td>
<td>MoveControl</td>
</tr>
<tr>
<td>$1710</td>
<td>DragControl</td>
</tr>
<tr>
<td>$1810</td>
<td>SetCtlIcons</td>
</tr>
<tr>
<td>$1910</td>
<td>SetCtlValue</td>
</tr>
<tr>
<td>$1A10</td>
<td>GetCtlValue</td>
</tr>
<tr>
<td>$1B10</td>
<td>SetCtlParams</td>
</tr>
<tr>
<td>$1C10</td>
<td>GetCtlParams</td>
</tr>
<tr>
<td>$1D10</td>
<td>DragRect</td>
</tr>
<tr>
<td>$1E10</td>
<td>GrowSize</td>
</tr>
<tr>
<td>$1F10</td>
<td>GetCtlDPage</td>
</tr>
<tr>
<td>$2010</td>
<td>SetCtlAction</td>
</tr>
<tr>
<td>$2110</td>
<td>GetCtlAction</td>
</tr>
<tr>
<td>$2210</td>
<td>SetCtlRefCon</td>
</tr>
<tr>
<td>$2310</td>
<td>GetCtlRefCon</td>
</tr>
<tr>
<td>$2410</td>
<td>EraseControl</td>
</tr>
<tr>
<td>$2510</td>
<td>DrawOneCtl</td>
</tr>
</tbody>
</table>

(continued)
<table>
<thead>
<tr>
<th>Number</th>
<th>Routine</th>
<th>Number</th>
<th>Routine</th>
<th>Number</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0111</td>
<td>LoaderInitialization</td>
<td>$0113</td>
<td>PMBootInit</td>
<td>$0114</td>
<td>LEBootInit</td>
</tr>
<tr>
<td>$0211</td>
<td>LoaderStartUp</td>
<td>$0213</td>
<td>PMStartUp</td>
<td>$0214</td>
<td>LEStartUp</td>
</tr>
<tr>
<td>$0311</td>
<td>LoaderShutdown</td>
<td>$0313</td>
<td>PMShutdown</td>
<td>$0314</td>
<td>LEShutdown</td>
</tr>
<tr>
<td>$0411</td>
<td>LoaderVersion</td>
<td>$0413</td>
<td>PMVersion</td>
<td>$0414</td>
<td>LEVersion</td>
</tr>
<tr>
<td>$0511</td>
<td>LoaderReset</td>
<td>$0513</td>
<td>PMReset</td>
<td>$0514</td>
<td>LEReset</td>
</tr>
<tr>
<td>$0611</td>
<td>LoaderStatus</td>
<td>$0613</td>
<td>PMStatus</td>
<td>$0614</td>
<td>LEStatus</td>
</tr>
<tr>
<td>$0911</td>
<td>InitialLoad</td>
<td>$0913</td>
<td>PrDefault</td>
<td>$0914</td>
<td>LENew</td>
</tr>
<tr>
<td>$0A11</td>
<td>Restart</td>
<td>$0A13</td>
<td>PrValidate</td>
<td>$0A14</td>
<td>LEDispose</td>
</tr>
<tr>
<td>$0B11</td>
<td>LoadSegNum</td>
<td>$0B13</td>
<td>PrDlgItem</td>
<td>$0B14</td>
<td>LESetText</td>
</tr>
<tr>
<td>$0C11</td>
<td>UnloadSegNum</td>
<td>$0C13</td>
<td>PrJobDialog</td>
<td>$0C14</td>
<td>LEIdle</td>
</tr>
<tr>
<td>$0D11</td>
<td>LoadSegName</td>
<td>$0D13</td>
<td>PrPixelMap</td>
<td>$0D14</td>
<td>LEClick</td>
</tr>
<tr>
<td>$0E11</td>
<td>UnloadSeg</td>
<td>$0E13</td>
<td>PrOpenDoc</td>
<td>$0E14</td>
<td>LEResetSelect</td>
</tr>
<tr>
<td>$0F11</td>
<td>GetLoadSegInfo</td>
<td>$0F13</td>
<td>PrCloseDoc</td>
<td>$0F14</td>
<td>LEActivate</td>
</tr>
<tr>
<td>$1011</td>
<td>GetUserID</td>
<td>$1013</td>
<td>PrOpenPage</td>
<td>$1014</td>
<td>LEDeactivate</td>
</tr>
<tr>
<td>$1111</td>
<td>LGetPathname</td>
<td>$1113</td>
<td>PrClosePage</td>
<td>$1114</td>
<td>LEKey</td>
</tr>
<tr>
<td>$1211</td>
<td>UserShutdown</td>
<td>$1213</td>
<td>PrPicFile</td>
<td>$1214</td>
<td>LECut</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1413</td>
<td>PrError</td>
<td>$1314</td>
<td>LECopy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1513</td>
<td>PrSetError</td>
<td>$1414</td>
<td>LEPaste</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1613</td>
<td>PrChoosePrinter</td>
<td>$1514</td>
<td>LEDelete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$2313</td>
<td>PrDriverVer</td>
<td>$1614</td>
<td>LEReset</td>
</tr>
<tr>
<td></td>
<td>QuickDraw Auxiliary</td>
<td>$2413</td>
<td>PrPortVer</td>
<td>$1714</td>
<td>LEUpdate</td>
</tr>
<tr>
<td></td>
<td>(Chapter 17)</td>
<td></td>
<td></td>
<td>$1814</td>
<td>LEReSetText</td>
</tr>
<tr>
<td>$0112</td>
<td>QDAuxBootInit</td>
<td>$1914</td>
<td>LEForgotScrap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0212</td>
<td>QDAuxStartUp</td>
<td>$1A14</td>
<td>LEToScrap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0312</td>
<td>QDAuxShutdown</td>
<td>$1B14</td>
<td>LEScpareHandle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0412</td>
<td>QDAuxVersion</td>
<td>$1C14</td>
<td>LEGetScrapLen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0512</td>
<td>QDAuxReset</td>
<td>$1D14</td>
<td>LESetScrapLen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0612</td>
<td>QDAuxStatus</td>
<td>$1E14</td>
<td>LESetHiLite</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0912</td>
<td>CopyPixels</td>
<td>$1F14</td>
<td>LESetCaret</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0A12</td>
<td>WaitCursor</td>
<td>$2014</td>
<td>LEReSetButton</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0B12</td>
<td>DrawIcon</td>
<td>$2114</td>
<td>LEReSetJust</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$2214</td>
<td>LEGetTextHand</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$2314</td>
<td>LEGetTextLen</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table D-1 (continued)
Routines by tool set/routine number

Appendix D: List of Routines by Tool Set Number and Routine Number
<table>
<thead>
<tr>
<th>Number</th>
<th>Routine</th>
<th>Number</th>
<th>Routine</th>
<th>Standard File Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0115</td>
<td>DialogBootInit</td>
<td>$2715</td>
<td>SetDItemType</td>
<td></td>
</tr>
<tr>
<td>$0215</td>
<td>DialogStartUp</td>
<td>$2815</td>
<td>GetDItemBox</td>
<td></td>
</tr>
<tr>
<td>$0315</td>
<td>DialogShutDown</td>
<td>$2915</td>
<td>SetDItemBox</td>
<td></td>
</tr>
<tr>
<td>$0415</td>
<td>DialogVersion</td>
<td>$01A1</td>
<td>DialogBootInit</td>
<td></td>
</tr>
<tr>
<td>$0515</td>
<td>DialogReset</td>
<td>$02A1</td>
<td>DialogStartUp</td>
<td></td>
</tr>
<tr>
<td>$0615</td>
<td>DialogStatus</td>
<td>$03A1</td>
<td>DialogShutDown</td>
<td></td>
</tr>
<tr>
<td>$0915</td>
<td>ErrorSound</td>
<td>$04A1</td>
<td>DialogVersion</td>
<td></td>
</tr>
<tr>
<td>$0A15</td>
<td>NewModalDialog</td>
<td>$05A1</td>
<td>DialogReset</td>
<td></td>
</tr>
<tr>
<td>$0B15</td>
<td>NewModelessDialog</td>
<td>$06A1</td>
<td>DialogStatus</td>
<td></td>
</tr>
<tr>
<td>$0C15</td>
<td>CloseDialog</td>
<td>$07A1</td>
<td>ErrorSound</td>
<td></td>
</tr>
<tr>
<td>$0D15</td>
<td>NewDItem</td>
<td>$08A1</td>
<td>NewModalDialog</td>
<td></td>
</tr>
<tr>
<td>$0E15</td>
<td>RemoveDItem</td>
<td>$09A1</td>
<td>NewModelessDialog</td>
<td></td>
</tr>
<tr>
<td>$0F15</td>
<td>ModalDialog</td>
<td>$0A15</td>
<td>CloseDialog</td>
<td></td>
</tr>
<tr>
<td>$1015</td>
<td>IsDialogEvent</td>
<td>$0B15</td>
<td>NewDItem</td>
<td></td>
</tr>
<tr>
<td>$1115</td>
<td>DialogSelect</td>
<td>$0C15</td>
<td>RemoveDItem</td>
<td></td>
</tr>
<tr>
<td>$1215</td>
<td>DlgCut</td>
<td>$0D15</td>
<td>ModalDialog</td>
<td></td>
</tr>
<tr>
<td>$1315</td>
<td>DlgCopy</td>
<td>$1015</td>
<td>IsDialogEvent</td>
<td></td>
</tr>
<tr>
<td>$1415</td>
<td>DlgPaste</td>
<td>$1115</td>
<td>DialogSelect</td>
<td></td>
</tr>
<tr>
<td>$1515</td>
<td>DlgDelete</td>
<td>$1215</td>
<td>DlgCut</td>
<td></td>
</tr>
<tr>
<td>$1615</td>
<td>DrawDialog</td>
<td>$1315</td>
<td>DlgPaste</td>
<td></td>
</tr>
<tr>
<td>$1715</td>
<td>Alert</td>
<td>$1415</td>
<td>DlgDelete</td>
<td></td>
</tr>
<tr>
<td>$1815</td>
<td>StopAlert</td>
<td>$1515</td>
<td>DrawDialog</td>
<td></td>
</tr>
<tr>
<td>$1915</td>
<td>NoteAlert</td>
<td>$1615</td>
<td>Alert</td>
<td></td>
</tr>
<tr>
<td>$1A15</td>
<td>CautionAlert</td>
<td>$1715</td>
<td>StopAlert</td>
<td></td>
</tr>
<tr>
<td>$1B15</td>
<td>ParamText</td>
<td>$1815</td>
<td>NoteAlert</td>
<td></td>
</tr>
<tr>
<td>$1C15</td>
<td>SetDAFont</td>
<td>$1915</td>
<td>CautionAlert</td>
<td></td>
</tr>
<tr>
<td>$1D15</td>
<td>GetControlDItem</td>
<td>$1A15</td>
<td>ParamText</td>
<td></td>
</tr>
<tr>
<td>$1E15</td>
<td>GetDItem</td>
<td>$1B15</td>
<td>SetDAFont</td>
<td></td>
</tr>
<tr>
<td>$1F15</td>
<td>GetText</td>
<td>$1C15</td>
<td>GetControlDItem</td>
<td></td>
</tr>
<tr>
<td>$2015</td>
<td>SetIText</td>
<td>$1D15</td>
<td>GetDItem</td>
<td></td>
</tr>
<tr>
<td>$2115</td>
<td>GetIText</td>
<td>$1E15</td>
<td>SetIText</td>
<td></td>
</tr>
<tr>
<td>$2215</td>
<td>SelectIText</td>
<td>$0F15</td>
<td>GetIText</td>
<td></td>
</tr>
<tr>
<td>$2315</td>
<td>HideDItem</td>
<td>$1015</td>
<td>SelectIText</td>
<td></td>
</tr>
<tr>
<td>$2415</td>
<td>ShowDItem</td>
<td>$1115</td>
<td>HideDItem</td>
<td></td>
</tr>
<tr>
<td>$2515</td>
<td>FindDItem</td>
<td>$1215</td>
<td>ShowDItem</td>
<td></td>
</tr>
<tr>
<td>$2615</td>
<td>UpdateDialog</td>
<td>$1315</td>
<td>FindDItem</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number</th>
<th>Routine</th>
<th>Number</th>
<th>Routine</th>
<th>Note Synthesizer (not documented in this manual)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0116</td>
<td>ScrapBootInit</td>
<td>$0116</td>
<td>ScrapBootInit</td>
<td></td>
</tr>
<tr>
<td>$0216</td>
<td>ScrapStartUp</td>
<td>$0216</td>
<td>ScrapStartUp</td>
<td></td>
</tr>
<tr>
<td>$0316</td>
<td>ScrapShutDown</td>
<td>$0316</td>
<td>ScrapShutDown</td>
<td></td>
</tr>
<tr>
<td>$0416</td>
<td>ScrapVersion</td>
<td>$0416</td>
<td>ScrapVersion</td>
<td></td>
</tr>
<tr>
<td>$0516</td>
<td>ScrapReset</td>
<td>$0516</td>
<td>ScrapReset</td>
<td></td>
</tr>
<tr>
<td>$0616</td>
<td>ScrapStatus</td>
<td>$0616</td>
<td>ScrapStatus</td>
<td></td>
</tr>
<tr>
<td>$0916</td>
<td>UnloadScrap</td>
<td>$0916</td>
<td>UnloadScrap</td>
<td></td>
</tr>
<tr>
<td>$0A16</td>
<td>LoadScrap</td>
<td>$0A16</td>
<td>LoadScrap</td>
<td></td>
</tr>
<tr>
<td>$0B16</td>
<td>ZeroScrap</td>
<td>$0B16</td>
<td>ZeroScrap</td>
<td></td>
</tr>
<tr>
<td>$0C16</td>
<td>PutScrap</td>
<td>$0C16</td>
<td>PutScrap</td>
<td></td>
</tr>
<tr>
<td>$0D16</td>
<td>GetScrap</td>
<td>$0D16</td>
<td>GetScrap</td>
<td></td>
</tr>
<tr>
<td>$0E16</td>
<td>GetScrapHandle</td>
<td>$0E16</td>
<td>GetScrapHandle</td>
<td></td>
</tr>
<tr>
<td>$0F16</td>
<td>GetScrapSize</td>
<td>$0F16</td>
<td>GetScrapSize</td>
<td></td>
</tr>
<tr>
<td>$1016</td>
<td>GetScrapPath</td>
<td>$1016</td>
<td>GetScrapPath</td>
<td></td>
</tr>
<tr>
<td>$1116</td>
<td>SetScrapPath</td>
<td>$1116</td>
<td>SetScrapPath</td>
<td></td>
</tr>
<tr>
<td>$1216</td>
<td>GetScrapCount</td>
<td>$1216</td>
<td>GetScrapCount</td>
<td></td>
</tr>
<tr>
<td>$1316</td>
<td>GetScrapState</td>
<td>$1316</td>
<td>GetScrapState</td>
<td></td>
</tr>
</tbody>
</table>

Appendix D: List of Routines by Tool Set Number and Routine Number
<table>
<thead>
<tr>
<th>Number</th>
<th>Routine</th>
<th>Number</th>
<th>Routine</th>
<th>Number</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$011A$</td>
<td>SeqBootInit</td>
<td>$011B$</td>
<td>FMBootInit</td>
<td>$011C$</td>
<td>ListBootInit</td>
</tr>
<tr>
<td>$021A$</td>
<td>SeqStartUp</td>
<td>$021B$</td>
<td>FMStartUp</td>
<td>$021C$</td>
<td>ListStartUp</td>
</tr>
<tr>
<td>$031A$</td>
<td>SeqShutDown</td>
<td>$031B$</td>
<td>FMShutDown</td>
<td>$031C$</td>
<td>ListShutDown</td>
</tr>
<tr>
<td>$041A$</td>
<td>SeqVersion</td>
<td>$041B$</td>
<td>FMVersion</td>
<td>$041C$</td>
<td>ListVersion</td>
</tr>
<tr>
<td>$051A$</td>
<td>SeqReset</td>
<td>$051B$</td>
<td>FMReset</td>
<td>$051C$</td>
<td>ListReset</td>
</tr>
<tr>
<td>$061A$</td>
<td>SeqStatus</td>
<td>$061B$</td>
<td>FMStatus</td>
<td>$061C$</td>
<td>ListStatus</td>
</tr>
<tr>
<td>$091A$</td>
<td>SetIncr</td>
<td>$091B$</td>
<td>CountFamilies</td>
<td>$091C$</td>
<td>CreateList</td>
</tr>
<tr>
<td>$0A1A$</td>
<td>SeqBootInit</td>
<td>$0A1B$</td>
<td>FindFamily</td>
<td>$0A1C$</td>
<td>SortList</td>
</tr>
<tr>
<td>$0A1A$</td>
<td>ClearIncr</td>
<td>$0B1B$</td>
<td>GetFamInfo</td>
<td>$0B1C$</td>
<td>NextMember</td>
</tr>
<tr>
<td>$0B1A$</td>
<td>GetTimer</td>
<td>$0C1B$</td>
<td>GetFamNum</td>
<td>$0C1C$</td>
<td>DrawMember</td>
</tr>
<tr>
<td>$0C1A$</td>
<td>GetLoc</td>
<td>$0D1B$</td>
<td>AddFamily</td>
<td>$0D1C$</td>
<td>SelectMember</td>
</tr>
<tr>
<td>$0D1A$</td>
<td>AllNotesOff</td>
<td>$0E1B$</td>
<td>InstallFont</td>
<td>$0E1C$</td>
<td>GetListDefProc</td>
</tr>
<tr>
<td>$0E1A$</td>
<td>SetTrkInfo</td>
<td>$0F1B$</td>
<td>SetPurgeStat</td>
<td>$0F1C$</td>
<td>ResetMember</td>
</tr>
<tr>
<td>$0F1A$</td>
<td>StartSeq</td>
<td>$101B$</td>
<td>CountFonts</td>
<td>$101C$</td>
<td>NewList</td>
</tr>
<tr>
<td>$101A$</td>
<td>StepSeq</td>
<td>$111B$</td>
<td>FindFontStats</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$111A$</td>
<td>StopSeq</td>
<td>$121B$</td>
<td>LoadFont</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$121A$</td>
<td>SetInstTable</td>
<td>$131B$</td>
<td>LoadSysFont</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$131A$</td>
<td>StartIns</td>
<td>$141B$</td>
<td>AddFontVar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$141A$</td>
<td>StopIns</td>
<td>$151B$</td>
<td>FixFontMenu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$151A$</td>
<td></td>
<td>$161B$</td>
<td>ChooseFont</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$161A$</td>
<td></td>
<td>$171B$</td>
<td>ItemID2FamNum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$171A$</td>
<td></td>
<td>$181B$</td>
<td>FMSetSysFont</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$181A$</td>
<td></td>
<td>$191B$</td>
<td>FMGetSysFID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$191A$</td>
<td></td>
<td>$1A1B$</td>
<td>FMGetCurFID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$1A1A$</td>
<td></td>
<td>$1B1B$</td>
<td>FamNum2ItemID</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table D-1 (continued)
Routines by tool set/routine number

Note Sequencer (not documented in this manual)
Font Manager (Chapter 8)
List Manager (Chapter 11)
absolute: Said of a load segment or other program code that must be loaded at a specific address in memory and never moved. Compare relocatable.

absolute addressing: An addressing mode in which instruction operands are interpreted as literal addresses.

absolute clamps: Values that establish the minimum and maximum X and Y coordinates for alternative pointing devices.

access, access byte: An attribute of a ProDOS file that controls whether the file may be read from, written to, renamed, or backed up.

accumulator: The register in a computer’s central processor or microprocessor where most computations are performed. Also called A register.

activate: To make active. A control or window may be activated. Compare enable.

activate event: A window event that occurs when a window is made either active or inactive.

active: Able to respond to the user’s mouse or keyboard actions. Controls and windows that are active are displayed differently than inactive items.

ADB: See Apple Desktop Bus.

ADB commands: Commands that are issued to the Apple Desktop Bus. These are not the same as Apple Desktop Bus Tool Set routines; rather, the tool set routines often include an ADB command as a parameter. The Apple Desktop Bus Tool Set then interprets and issues the ADB command.

alert: A warning or report of an error in the form of an alert box, a sound from the computer’s speaker, or both.

alert box: A special type of dialog box that appears on the screen to give a warning or to report an error message during use of an application.

alert sound: A sound generated by a sound procedure that emits a tone or sequence of tones when the user is to be alerted of a condition.

alert stage: One of four stages that correspond to consecutive occurrences of an alert.

alert template: A data structure that contains an alert ID, a RECT determining the alert window’s size and location, information about what should happen at each stage of the alert, and a list of pointers to the item templates.

alert window: The window in which an alert box appears. One of the two predefined window formats. Compare document window.

alternative pointing devices: A device, such as a graphics tablet or trackball, used instead of the mouse.

Apple Desktop Bus (ADB): An input bus, with its own protocol and electrical characteristics, that provides a method of connecting input devices such as keyboards and mice with personal computers.

Apple Desktop Bus Tool Set: The Apple II GS tool set that facilitates an application’s interaction with devices connected to the Apple Desktop Bus.
Apple key: A modifier key on the Apple IIGS keyboard, marked with both an Apple icon and a spinner, the icon used on the equivalent key on some Macintosh keyboards. It performs the same functions as the open Apple key on standard Apple II machines.

AppleTalk network: A local area network developed by Apple Computer, Inc.

Apple II: A family of computers, including the original Apple II, the Apple II Plus, the Apple IIe, the Apple IIc, and the Apple IIGS. Compare standard Apple II.

Apple II Plus: A personal computer in the Apple II family with expansion slots that allow the user to enhance the computer's capabilities with peripheral cards.

Apple IIe: A transportable personal computer in the Apple II family, with a disk drive, serial ports, and 80-column display capability built in.

Apple IIc: A personal computer in the Apple II family, with seven expansion slots and an auxiliary memory slot that allow the user to enhance the computer's capabilities with peripheral memory and video enhancement cards.

Apple IIGS: The most advanced computer in the Apple II family. It features expanded memory, advanced sound and graphics, and the Apple IIGS Toolbox of programming routines.

Apple IIGS Programmer's Workshop: See APW.

Apple IIGS Toolbox: An extensive set of routines that facilitates writing desktop applications and provides easy program access to many Apple IIGS hardware and firmware features. Functions within the toolbox are grouped into tool sets.

application: A stand-alone program that performs a specific function, such as word processing, drawing, or telecommunications. Compare, for example, desk accessory, device driver.

application event: Any of four types of events available for applications to define and respond to as desired.

application prefix: The ProDOS 16 prefix number 1/1. It specifies the directory of the currently running application.

application window: A window in which an application's document appears.

APW (Apple IIGS Programmer's Workshop): A multilanguage development environment for writing Apple IIGS desktop applications.

APW Assembler: The 65816 assembly-language assembler provided with the Apple IIGS Programmer's Workshop.

APW C Compiler: The C-language compiler provided with the Apple IIGS Programmer's Workshop.

APW Shell: The programming environment of the Apple IIGS Programmer's Workshop. It provides facilities for file manipulation and program execution, and supports shell applications.

arbitrary mode: In the List Manager, a selection mode that allows the user to select members in a list without deselecting already-selected members.

arc: A portion of an oval; one of the fundamental shapes drawn by QuickDraw II.

A register: See accumulator.

ascent: In a font, the distance between the base line and the ascent line.

ascent line: A horizontal line that coincides with the tops of the tallest characters in a font. See also base line, descent line.
ASCII: Acronym for *American Standard Code for Information Interchange*, pronounced "ASK-ee." A code in which the numbers from 0 to 127 stand for text characters. ASCII code is used to represent text inside a computer and to transmit text between computers or between a computer and a peripheral device.

assembler: A language translator that converts a program written in assembly language into an equivalent program in machine language.

AsyncADBReceive completion routine: Used in conjunction with the ADB Tool Set routine AsyncADBReceive, the completion routine obtains ADB data from a buffer. Compare SRQ list completion routine.

attributes word: Determines how memory blocks are allocated and maintained. Most of the attributes are defined at allocation time and can't be changed after that; other attributes can be modified after allocation.

auto-key: A keyboard feature and an event type, in which a key being held down continuously is interpreted as a rapid series of identical keystrokes.

auxID: A subfield of the user ID. An application may place any value it wishes into the auxID field.

auxiliary type: A secondary classification of ProDOS files. A file's auxiliary-type field may contain information of use to the applications that read it. Compare file type.

available font: A font that the Font Manager can use because the font is the ROM font, or a font in the FONTS subdirectory, or a font that the application has added with the Font Manager routine AddFontVar.

background: The pixels within a character or other screen object that are not part of the object itself.

background color: The color of background pixels in text; by default it is black.

background pattern: The pattern QuickDraw II uses to erase objects on the screen.

background pixels: in a character image, the pixels that are not part of the character itself; that is, all pixels in the character bounds rectangle that are not foreground pixels.

background procedure: A procedure run by the Print Manager whenever the Print Manager has directed output to the printer and is waiting for the printer to finish.

bank: A 64K (65,536-byte) portion of the Apple II GS internal memory. An individual bank is specified by the value of the 65C816 microprocessor's bank register.

bank $00: The first bank of memory in the Apple II GS. In emulation mode, it is equivalent to main memory in an Apple IIe or Apple IIc computer.

base family: A font family is a base family if it is the ROM font or if a plain-styled example of the family can be found among the fonts in the FONTS subdirectory.

base height: In the LineEdit Tool Set, the distance between the top of the destination rectangle and the base line. This controls where the text is drawn.

base line: A horizontal line that coincides with the bottom of the main body of each character in a font. Character descenders extend below the base line.

BASIC: Acronym for *Beginners All-purpose Symbolic Instruction Code*. BASIC is a high-level programming language designed to be easy to learn.

best-fit font algorithm: The algorithm that the Font Manager routine InstallFont uses to look for a font that matches a given set of specifications.

bit: A contraction of binary digit, the smallest representation of data in a digital computer.
**bit plane**: A method of representing images in computer memory. In a bit plane, consecutive bits in memory specify adjacent pixels in the image; if more than one bit is required to completely specify the state of a pixel, more than one bit plane is used for the image. Compare **chunky pixels**.

**block**: (1) A unit of data storage or transfer, typically 512 bytes. (2) A contiguous region of computer memory of arbitrary size, allocated by the Memory Manager. Also called **memory block**.

**block device**: A device that transfers data to or from a computer in multiples of 1 block (512 bytes) of characters at a time. Disk drives are block devices. Also called **block I/O device**.

**block I/O device**: See **block device**.

**Boolean logic**: A mathematical system in which every expression evaluates to one of two values, usually referred to as TRUE or FALSE.

**Boolean variable**: A variable that can have one of two values, usually referred to as TRUE or FALSE.

**bottom scroll bar**: The scroll bar the user selects to scroll horizontally through the data in a window.

**boundary rectangle**: A rectangle, defined as part of a QuickDraw II **locinfo** record, that encloses the active area of the pixel image and imposes a coordinate system on it. Its top left corner is always aligned on the first pixel in the pixel map.

**boundsRect**: The GrafPort field that defines the port's boundary rectangle.

**buffer**: A holding area of the computer's memory where information can be stored by one program or device and then read, perhaps at a different rate, by another; for example, a print buffer.

**busy flag**: A feature that informs the Scheduler whether a currently needed resource is busy or available.

**button**: (1) A pushbutton-like image in dialog boxes where the user clicks to designate, confirm, or cancel an action. See also **check box**, **radio button**. (2) A button on a mouse or other pointing device. See also **mouse button**.

**byte**: A unit of information consisting of 8 bits. A byte can have any value between 0 and 255, which may represent an instruction, a letter, a number, a punctuation mark, or another character. See also **bit**, **kilobyte**, **megabyte**.

**C**: A high-level programming language. One of the languages available for the Apple II GS Programmer's Workshop.

**cancel**: To stop an operation, such as the setting of page-setup values in a dialog box, without saving any results produced up to that point.

**Cancel**: One of two predefined item ID numbers for dialog box buttons (Cancel=2). Compare **OK**.

**caret**: A symbol that indicates where something should or will be inserted in text. On the screen it designates the insertion point, and is usually a vertical bar (|).

**carry flag**: A status bit in the microprocessor indicating whether an accumulator calculation has resulted in a carry out of the register. Also called **c flag**.

**CDA**: See **classic desk accessory**.

**CDA menu**: The menu on which classic desk accessories are listed; the user selects the menu by pressing Control-Apple-Escape. See also **classic desk accessory**.

**c flag**: See **carry flag**.
character: (1) Any symbol that has a widely understood meaning and thus can convey information. Some characters—such as letters, numbers, and punctuation—can be displayed on the monitor screen and printed on a printer. Most characters are represented in the computer as 1-byte values. (2) In QuickDraw II, a single ASCII character.

character bounds rectangle: The rectangle that determines the extent of the background pixels of a character.

character bounds width: The width of a character’s character bounds rectangle.

character device: A device that transfers data to or from a computer as a stream of individual characters. Keyboards and printers are character devices.

character image: The part of a font strike that represents a character in a font.

character image width: The number of columns in a character image.

character origin: The point on the base line used as a reference location for drawing a character.

character position: An index into LineEdit text, with position 0 corresponding to the first character.

character rectangle: A rectangle that encloses a character image. Its width is equal to the image width of the character; its height is equal to the character height.

character width: The number of pixels the pen position is to be advanced after the character is drawn.

check box: A small box associated with an option in a dialog box. When the user clicks the check box, that may change the option or affect related options. See also radio button.

Choose Printer dialog box: A Print Manager dialog box that lets the user select a printer or port for printing.

chunkiness: The number of bits required to describe the state of a pixel in a pixel image.

chunky pixels: A method of representing images in computer memory. In chunky pixel organization, a number of consecutive bits in memory combine to specify the state of a single pixel in the image. Consecutive groups of bits (the size of the group is equal to the image’s chunkiness) define adjacent pixels in the image. Compare bit plane.

clamp values: The X- and Y-limits, in terms of pixels, on cursor position controlled by mouse movement.

classic desk accessory (CDA): Desk accessories designed to execute in a nondesktop-, nonevent-based environment. Compare new desk accessory.

click: To position the pointer on something, and then to press and quickly release the mouse or alternative pointing device’s button.

clip: To restrict drawing to within a particular boundary; any drawing attempted outside that boundary does not occur.

Clipboard: The holding place for the material the user last cut or copied; a buffer area in memory. Information on the Clipboard can be inserted (pasted) into documents. In memory, the contents of the clipboard are called the desk scrap.

clipping region: The region to which an application limits drawing in a GrafPort.

clock: (1) The timing circuit that controls execution of a microprocessor. Also called system clock. (2) An integrated circuit, often with battery-backup memory, that gives the current date and time.
**close box:** The small white box on the left side of the title bar of an active window. Clicking it closes the window.

**color table:** One table of 16 lookup tables in Apple IIGS memory. The table lists the available color values for a scan line.

**compaction:** The rearrangement of allocated blocks in memory to open up larger contiguous areas of free space.

**compiler:** A program that produces object files (containing machine-language code) from source files written in a high-level language such as C. Compare assembler.

**content region:** The area in a window in which your application presents information to the user.

**control:** An object in a window with which the user, using the mouse, can cause instant action with visible results or change settings to modify a future action.

**control definition procedure:** A procedure used to define the appearance and behavior of a custom control.

**Control Manager:** The Apple IIGS tool set that manages controls, which are objects on the screen that the user can manipulate with the mouse to cause instant action or change settings.

**Control Panel:** A desk accessory that lets the user change certain system parameters, such as speaker volume, display colors, and configuration of slots and ports.

**control record:** A data structure that defines the appearance and behavior of a control.

**coordinate plane:** A two-dimensional grid defined by QuickDraw II. All drawing commands are located in terms of coordinates on the grid.

**coordinates:** X and Y locations on the QuickDraw II coordinate plane. Most QuickDraw II routines accept and return coordinates in the order (Y,X).

**copy:** To duplicate something by selecting it and choosing Copy from the Edit menu. A copy of the selected portion is placed on the Clipboard, without affecting the original selection.

**C string:** An ASCII character string terminated by a null character (ASCII value = 0). Compare Pascal string.

**C-type string:** Same as C string.

**current font:** The font currently being used by QuickDraw II to draw text.

**cursor:** A symbol displayed on the screen, marking where the user's next action will take effect or where the next character typed from the keyboard will appear.

**cursor record:** The data structure that defines the height and width of the cursor, the image of the cursor, the mask controlling the appearance of the cursor, and the hot spot defining where the image of the cursor will be placed by the mouse.

**cut:** To remove something by selecting it and choosing Cut from the Edit menu. The cut portion is placed on the Clipboard.

**data area:** The name for a document as viewed in a window. The data area is the entire document, only a portion of which (the visible region) may be seen in the window at any one time.

**data bank register:** A register in the 65C816 processor that contains the high-order byte of the 24-bit address that references data in memory.

**data structure:** A specifically formatted item of data or a form into which data may be placed.

**DB register:** See data bank register.

**dead character:** A character with a character width of 0.

**default button:** The button in a dialog box whose action will be executed if the user presses the Return key.
**default prefix:** The pathname prefix attached by ProDOS 16 to a partial pathname when no prefix number is supplied by the application. The default prefix is equivalent to prefix number 0/.

**dereference:** To substitute a pointer for a memory handle, or a value for a pointer. When you dereference a memory block's handle, you access the block directly (through its master pointer) rather than indirectly (through its handle).

**descender:** Any part of a character that lies below the base line.

**descent:** In a font, the distance between the base line and the descent line.

**descent line:** A horizontal line that coincides with the bottoms of character descenders (such as the tail on a lowercase "p") that extends farthest below the base line. See also ascent line, base height, font height.

**desk accessory:** A "mini-application" that is available to the user regardless of whether another application is running. The Apple IIGS supports two types of desk accessories: classic desk accessories and new desk accessories.

**desk accessory event:** An event that occurs whenever the user presses Control-Apple-Escape to invoke a classic desk accessory.

**Desk Accessory menu:** The menu whose title is a colored apple symbol.

**Desk Manager:** The Apple IIGS tool set that executes desk accessories and enables applications to support them.

**desk scrap:** A piece of data, maintained by the Scrap Manager, taken from one application and available for insertion into another.

**desktop:** The visual interface between the computer and the user—the menu bar and the gray (or solid-colored) area on the screen. In many applications the user can have a number of documents on the desktop at the same time.

**desktop user interface:** See desktop.

**destination location:** The location (memory buffer or portion of the QuickDraw II coordinate plane) to which data such as text or graphics are copied. See also destination rectangle.

**destination rectangle:** (1) The rectangle (on the QuickDraw II coordinate plane) in which text or graphics are drawn when transferred from somewhere else. Compare source rectangle. (2) In LineEdit, the rectangle that determines where the text will be drawn.

**device:** A piece of hardware used in conjunction with a computer and under the computer's control. Also called peripheral device because such equipment is often physically separate from, but attached to, the computer.

**device driver:** A program that handles the transfer of data to and from a peripheral device, such as a printer or disk drive.

**device driver event:** An event generated by a device driver.

**dial:** An indicator on the screen that displays a quantitative setting or value. Usually found in analog form, such as a fuel gauge or a thermometer. A scroll bar is a standard type of dial.

**dialog:** See dialog box.

**dialog box:** A box on the screen that contains a message requesting more information from the user. See also alert box.

**Dialog Manager:** The Apple IIGS tool set that manipulates dialog boxes and alerts, which appear on the screen when an application needs more information to carry out a command or when the user needs to be notified of an important situation.

**dialog pointer:** A pointer to a dialog's GrafPort; equivalent to the window pointer for the dialog box.

**dialog record:** Information describing a dialog window that is maintained by the Dialog Manager.
dialog template: A record that contains information about a dialog to be created.
dialog window: The window in which a dialog box appears.
digital oscillator chip (DOC): An integrated circuit in the Apple IIGS that contains 32 digital oscillators, each of which can generate a sound from stored digital waveform data.
dim: On the Apple IIGS desktop, to display a control or menu item in gray rather than black, to notify the user that the item is inactive.
direct page: A page (256 bytes) of bank $00 of Apple IIGS memory, any part of which can be addressed with a short (1-byte) address because its high-order address byte is always $00 and its middle address byte is the value of the 65C816 direct register. Co-resident programs or routines can have their own direct pages at different locations. The direct page corresponds to the 6502 processor's zero page. The term direct page is often used informally to refer to any part of the direct-page/stack space. Compare zero page.
direct-page/stack segment: A program segment that is used to initialize the size and contents of an application's stack and direct page.
direct-page/stack space: A single block of memory that contains an application's stack and direct page.
direct register: A hardware register in the 65C816 processor that specifies the start of the direct page.
disable: To make unresponsive to user actions. A dialog box control that is disabled does nothing when selected or manipulated by the user. In appearance, however, it is identical to an enabled control. Compare inactive.
disabled menu: A menu that can be pulled down, but in which items are dimmed and not selectable.
display mode: A specification for the way in which a video display functions, including such parameters as text or graphics display, available colors, and number of pixels. The Apple IIGS has two text display modes (40 column and 80 column), two standard Apple II graphics display modes (320 mode and 640 mode), and two new Super Hi-Res graphics display modes.
display rectangle: A rectangle that determines where an item is displayed within a dialog box.
dispose: To permanently deallocate (a memory block). The Memory Manager disposes of a memory block by removing its master pointer. Any handle to that pointer will then be invalid. Compare purge.
dithering: A technique for alternating the values of adjacent pixels to create the optical effect of intermediate values. Dithering can give the effect of shades of gray on a black-and-white display, or more colors on a color display.
dividing line: A line that divides groups of items in a menu; such a line uses the space of an entire item and requires an item record. Compare underline.
DOC: See digital oscillator chip.
document: A file created by an application.
document window: A window that displays a document. One of the two predefined window formats. Compare alert window.
double-click: To position the pointer where you want an action to take place, and then press and release the mouse button twice in quick succession without moving the mouse.
draft printing: The print method that the LaserWriter uses. QuickDraw II calls are converted directly into command codes the printer understands, which are then immediately used to drive the printer. Compare spool printing.
**drag**: To position the pointer on something, press and hold the mouse button, move the mouse, and release the mouse button. When you release the mouse button, you either confirm a menu selection or move an object to a new location.

**drag region**: A region in a window (usually on the title bar) in which the mouse pointer must be placed before the user can drag the window.

**draw**: In QuickDraw II, to color pixels in a pixel image.

**drawing environment**: The complete description of how and where drawing may take place. Every open window on the Apple II GS screen is associated with a GrafPort record, which specifies the window's drawing environment. Same as graphic port, port.

**drawing mask**: An 8-bit by 8-bit pattern that controls which pixels in the QuickDraw II pen will be modified when the pen draws.

**drawing mode**: One of 16 possible interactions between pixels in QuickDraw II's pen pattern and pixels already on the screen that fall under the pen's path. In modeCopy mode, for example, pixels already on the screen are ignored. In modeXOR mode, on the other hand, bits in pixels on the screen are XOR'd with bits in pixels in the pen; the resulting pixels are drawn on the screen. See also pen mode, text mode.

**drawing pen**: See pen.

**driver**: See device driver.

**dynamic segment**: A load segment capable of being loaded during program execution. Compare static segment.

**edit record**: A complete text-editing environment in the LineEdit Tool Set, which includes the text to be edited, the GrafPort and rectangle in which to display the text, the arrangement of the text within the rectangle, and other editing and display information.

**e flag**: One of three flag bits in the 65C816 processor that programs use to control the processor's operating modes. The setting of the e flag determines whether the processor is in native mode (6502) or emulation mode (65816). See also m flag, x flag.

**empty handle**: A handle pointing to a NIL master pointer.

**emulate**: To operate in a way identical to a different system. For example, the 65C816 microprocessor in the Apple II GS can carry out all the instructions in a program originally written for an Apple II that uses a 6502 microprocessor, thus emulating the 6502.

**emulation mode**: The 8-bit configuration of the 65C816 processor in which it functions like a 6502 processor in all respects except clock speed.

**enable**: To make responsive to user manipulation. A dialog or menu that is enabled can be selected by the user. Enabling does not affect how an item is displayed. Compare activate.

**end-of-file**: See EOF.

**EOF (end-of-file)**: The logical size of a ProDOS 16 file; it is the number of bytes that may be read from or written to the file.

**erasing**: In QuickDraw II, to color an area with the background pattern.

**error**: The state of a computer after it has detected a fault in one or more commands sent to it. Also called error condition.

**error condition**: See error.

**error message**: A message issued by the system or application program when it has encountered an abnormal situation or an error in data.

**event**: A notification to an application of some occurrence (such as an interrupt generated by a keypress) that the application may want to respond to.
event code: A numeric value assigned to each event by the Event Manager. Compare task code.

event-driven program: A program that responds to user inputs in real time by repeatedly testing for events. An event-driven program does nothing until it detects an event such as a click of the mouse button.

Event Manager: The Apple II GS tool set that detects events as they happen and passes the events to the application or appropriate event handler, such as TaskMaster or GetNextEvent.

event mask: A parameter passed to an Event Manager routine to specify which types of events the routine should apply to.

event message: A field in the event record that contains additional information about the event.

event queue: A list of pending events maintained by the Event Manager.

event record: The internal representation of an event, through which your program learns all pertinent information about that event.

event type: The type of event reported to the Event Manager.

execution environment: See operating environment.

execution mode: One of two general states of execution of the 65C816 processor: native mode and 6502 emulation mode.

expansion slot: See slot.

Extended value: An 80-bit signed floating-point value with 64 bits of fraction.

extended task event record: A data structure based on the event record that contains information used and returned by TaskMaster.

FALSE: Zero. The result of a Boolean operation. The opposite of TRUE.

family name: The name identifying a font family. For example, the font family named Helvetica includes 10-point Helvetica, 12-point Helvetica Bold, and 36-point Helvetica Underlined. See also font family.

family number: The number identifying a font family. There is a one-to-one correspondence between family number and family name; that is, any two fonts with the same family number should have the same family name.

FamSpecBits: A bit flag in the Font Manager that restricts the range of font families available to a calling routine.

FamStatBits: A bit flag in the Font Manager that reports on the status of a font family.

file: Any named, ordered collection of information stored on a disk. Application programs and operating systems on disks are examples of files, so also are text or graphics materials created by applications and saved on disks. Text and graphics files are also called documents.
ilename: The string of characters that identifies a particular file within its directory. ProDOS filenames may be up to 15 characters long. Compare pathname.

file type: An attribute of a ProDOS file that characterizes its contents and indicates how the file may be used. On disk, file types are stored as numbers; in a directory listing, they are often displayed as three-character or single-word mnemonic codes. Compare auxiliary type.

filling: In QuickDraw II, using a specified pattern and the drawing mask to fill the interior of a shape.

fill mode: A display option in Super Hi-Res 320 mode. In fill mode, pixels in memory with the value 0 are automatically assigned the color of the previous nonzero pixel on the scan line; the program thus need assign explicit pixel values only to change pixel colors.
**filter procedure**: A procedure that allows the application programmer to control the types of events handled by the Dialog Manager.

**firmware**: Programs stored permanently in ROM; most provide an interface to system hardware. Such programs (for example, the Monitor program) are built into the computer at the factory. They can be executed at any time but cannot be modified or erased. Compare hardware, software.

**fixed**: Not movable in memory once allocated. Program segments that must not be moved are placed in fixed memory blocks. Also called *immovable*. The opposite of movable.

**fixed address**: A memory block that must be at a specified address when allocated.

**fixed bank**: A block of memory that must start in a specified bank.

**Fixed value**: A 32-bit signed value with 16 bits of fraction.

**flag**: A variable whose value (usually 1 or 0, standing for TRUE or FALSE) indicates whether some condition holds or whether some event has occurred. A flag is used to control the program's actions at some later time.

**folder**: The visual representation of a subdirectory. See also subdirectory.

**font**: In typography, a complete set of type in one size and style of character. In computer usage, a collection of letters, numbers, punctuation marks, and other typographical symbols with a consistent appearance; the size and style can be changed readily. See also font scaling.

**font bounds rectangle**: The smallest rectangle that would enclose all the pixels of every character in a font; that is, the rectangle that is the union of all the character bounds rectangles of the characters in the font.

**font family**: All fonts that share the same name but may vary in size or style. For example, all fonts named Helvetica are in the same family, even though that family contains Helvetica, Helvetica Narrow, and Helvetica Bold.

**font height**: The vertical distance from a font's ascent line to its descent line.

**font ID**: A number that specifies a font by family, style, and size.

**font ID record**: A record containing the number that specifies a font by family, style, and size.

**Font Manager**: The Apple II GS tool set that allows applications to use different fonts.

**font rectangle**: The smallest rectangle that would completely enclose all the foreground pixels of the characters of a font if the characters were drawn so that their character origins coincided.

**font scaling**: A process by which the Font Manager creates a font at one size by enlarging or reducing characters in an existing font of another size.

**font size**: The size of a font in points, from 1 to 255. The Font Manager defines the font size as a byte; QuickDraw II and the Apple II GS font record define the font size as a word.

**FontSpecBits**: A bit flag in the Font Manager that restricts the range of fonts available to a calling routine.

**FontStatBits**: A bit flag in the Font Manager that reports on the status of a font.

**font strike**: A 1-bit-per-pixel pixel image consisting of the character images of every defined character in the font, placed sequentially in order of increasing ASCII code.

**font style**: The style in which a font was designed. The Font Manager defines the style as a byte; QuickDraw II and the Apple II GS font record define the font style as a word.
font substitution: An option in the LaserWriter style dialog box in the Print Manager, font substitution tells the system to substitute one font for another if the specified font is not available on the LaserWriter.

foreground color: The color of the foreground pixels in text; by default it is white.

foreground pixels: In a character image, the pixels corresponding to the character itself; that is, the bits set to 1 in the image.

FPT: See function pointer table.

Frac value: A 32-bit signed value with 30 bits of fraction.

fragmentation: A condition in which free (unallocated) portions of memory are scattered due to repeated allocation and deallocation of blocks by the Memory Manager.

frame region: The part of a window that surrounds the window's content region and contains standard window controls.

framing: In QuickDraw II, using the current pen size, pen pattern, drawing mask, and pen mode to draw an outline of a shape.

full native mode: See native mode.

full pathname: The complete name by which a file is specified, starting with the volume directory name. A full pathname always begins with a slash (/) because a volume directory name always begins with a slash. See also pathname.

function pointer table (FPT): A table, maintained by the Tool Locator, that points to all routines in a given tool set.

GCB: See generator control block.

general logic unit: See GLU.

generator: In the swap mode of the DOC, a functional unit formed from a pair of oscillators.

generator control block (GCB): A 16-byte block in the sound routines' work area that controls one generator.

GetNextEvent: The Event Manager call that an application can make on each cycle through its main event loop. Compare TaskMaster.

global coordinates: The coordinate system assigned to a pixel image (such as screen memory) that QuickDraw II draws to. In global coordinates, the boundary rectangle's origin (top left corner) has the value (0,0). Compare local coordinates.

GLU (general logic unit): A class of custom integrated circuits used as interfaces between different parts of the computer.

go-away region: A region in a window frame, corresponding to the close box. Clicking inside this region of the active window makes the window close or disappear.

GrafPort: A data structure (record) that specifies a complete drawing environment, including such elements as a pixel image, boundaries within which to draw, a character font, patterns for drawing and erasing, and other pen characteristics.

graphic port: A specification for how and where QuickDraw II draws. A graphic port is defined by its GrafPort record; an application may have more than one graphic port open at one time, each defined by its own GrafPort. Same as drawing environment.

grow image: A dotted outline of an entire window plus the lines delimiting the title bar, size box, and scroll bar areas. The image can be pulled around to follow the movements of the mouse until the mouse button is released.

grow region: A window region in which dragging changes the size of the window.

handle: See memory handle.
**hardware**: In computer terminology, the machinery that makes up a computer system. Compare firmware, software.

**Heartbeat routines**: Routines that execute at the heartbeat interrupt signal, during the vertical blanking interval (every 1/60 of a second).

**hex**: See hexadecimal.

**hexadecimal, hex**: The representation of numbers in the base-16 system, using the ten digits 0 through 9 and the six letters A through F. Each hexadecimal digit corresponds to a sequence of four binary digits, or bits. Hexadecimal numbers are usually preceded by a dollar sign ($).

**hide**: To make invisible on the screen (but not necessarily to discard).

**highlight**: To make something visually distinct. For example, when a button on a dialog box is selected, it appears as light letters on a dark background, rather than dark on light. An active window or control is highlighted differently than an inactive one.

**horizontal blanking**: The interval between the drawing of each scan line on a video display.

**hot spot**: The interval between the drawing of each scan line on a video display.

**Human Interface Guidelines**: Apple Computer’s set of conventions and suggestions for writing desktop programs. Programs that follow the Human Interface Guidelines: The Apple Desktop Interface present a consistent and friendly interface to users.

**icon**: An image that graphically represents an object, a concept, or a message.

**i flag**: A bit in the 65C816 microprocessor’s Processor Status register that disables interrupts if set to 1.

**image**: A representation of the contents of memory. A code image consists of machine-language instructions or data that may be loaded unchanged into memory. See also pixel image.

**image pointer**: In QuickDraw II, the pointer to the first byte of a pixel image.

**image width**: (1) Part of the QuickDraw II locinfo record that specifies the width of each line of a pixel image; the width must be an even multiple of 8 bytes. (2) For characters, same as character image width.

**immovable**: See fixed.

**inactive**: Said of controls that have no meaning or effect in the current context, such as an Open button when no document has been selected to be opened. These inactive controls are not affected by the user’s mouse action and are dimmed on the screen. Compare disable.

**index register**: A register in a computer processor that holds an index for use in indexed addressing. The 6502 and 65C816 microprocessors used in the Apple II family of computers have two index registers, called the X register and the Y register.

**indicator**: On a dial type of control, the moving part that displays the current setting.

**information bar**: An optional component of a window. If present, the information bar appears just below the title bar. It may contain any application-defined information.

**initialization segment**: A segment in an initial load file that is loaded and executed independently of the rest of the program. It is commonly executed first, to perform any initialization that the program may require.

**input/output**: See I/O.

**insertion point**: The place in a document where something will be added; it is selected by clicking and is normally represented by a blinking vertical bar.

**Integer value**: A 16-bit signed or unsigned value.

**Integer Math String**: An ASCII string with no length indication supplied by the string itself.
**Integer Math Tool Set:** The Apple IIGS tool set that performs simple mathematical functions on integers and other fixed-point numbers and converts numbers to their ASCII string-equivalents.

**interface library:** A set of variable- and data-structure definitions that link a program (such as a C application) with software written in another language (such as the Apple IIGS Toolbox).

**interrupt:** A temporary suspension in the execution of a main program that allows the computer to perform some other task, typically in response to a signal from a peripheral device or other source external to the computer.

**interrupt environment:** The machine state, including register length and contents, that the interrupt handler executes within.

**interrupt mode:** A mode in which interrupts are used to synchronize drawing with vertical blanking.

**inverting:** In QuickDraw II, using the drawing mask to invert the pixels in the interior of a shape.

**I/O (input/output):** A general term that encompasses input/output activity, the devices that accomplish it, and the data involved.

**I/O space:** The portion of the memory map in a standard Apple II (and in banks $E0 and $E1 of an Apple IIGS) with addresses between $C000 and $CFFF. Programs perform I/O by writing to or reading from locations in this I/O space.

**item:** A component of a dialog box, such as a button, a text field, or an icon.

**item descriptor:** In a dialog box, a pointer or a handle to additional information concerning a dialog item.

**item ID:** A unique number that defines an item in a dialog box and allows further reference to it.

**item line:** The line of text that defines a menu item's name and appearance.

**item list:** A list of information about all the items in a dialog box or an alert box.

**item template:** A record that contains information about the items in a dialog box.

**item type:** Identifies the type of dialog item, usually represented by a predefined constant (such as statText) or a series of constants.

**item value:** In a dialog box, additional information concerning a dialog item.

**job dialog box:** A dialog box presented when the user selects Print from the File menu.

**job subrecord:** A field in the print record that contains information about a particular printing job. See also print record.

**journaling mechanism:** A mechanism that can separate the Event Manager from the user and feed the manager events from a file.

**JSL (Jump to Subroutine Long):** A 65816 assembly-language instruction that requires a long (3-byte) address. JSL can be used to transfer execution to code in another memory bank.

**JSR (Jump to Subroutine):** A 6502 and 65816 assembly-language instruction that requires a 2-byte address.

**Jump Table:** (1) A table constructed in memory by the System Loader from all Jump Table segments encountered during a load. The Jump Table contains all references to dynamic segments that may be called during execution of the program. (2) The mechanism the Sound Tool Set uses to find a low-level sound routine.

**Jump Table segment:** A segment in a load file that contains all references to dynamic segments that may be called during execution of that load file. The Jump Table segment is created by the linker. In memory, the loader combines all Jump Table segments it encounters into the Jump Table.
K: See kilobyte.

kerning: The situation that occurs when a character has foreground pixels to the left of the old pen position or to the right of the new pen position or both. When kerning occurs, the character images of adjacent characters may overlap.

keyboard equivalent: The combination of the Apple key and another key, used to invoke a menu item from the keyboard.

key-down event: An event type caused by the user pressing any character key on the keyboard or keypad. The character keys include all keys except Shift, Caps Lock, Control, Option, and Apple, which are called modifier keys. Modifier keys are treated differently and generate no keyboard events of their own.

kilobyte (K): A unit of measurement consisting of 1024 (2^10) bytes. In this usage, kilo (from the Greek, meaning a thousand) stands for 1024. Thus, 64K memory equals 65,536 bytes. See also megabyte.

landscape mode: A printing mode in which text is printed top to bottom (that is, longways) on the paper.

leading: Pronounced "LED-ing." The space between lines of text. It is the number of pixels vertically between the descent line of one character and the ascent line of the character immediately beneath it.

leftward kern: For characters, the distance in pixels from the character origin to the left edge of the character.

length byte: The first byte of a Pascal string. It specifies the length of the string, in bytes.

library, library file: An object file containing program segments, each of which can be used in any number of programs. The linker can search through the library file for segments that have been referenced in the program source file.

library file: See library.

limit rectangle: The rectangle that limits the travel of a region that is being dragged with the mouse.

line: In QuickDraw II, an infinitely thin graphic object that is represented by its ends, which are defined by two points.

LineEdit Tool Set: The Apple IIGS tool set that provides simple text-editing functions. It is used mostly in dialog boxes.

line height: The total amount of vertical space from line to line in a text document. Line height is the sum of ascent, descent, and leading.

list: As defined by the List Manager, a scrollable, vertical arrangement of similar items on the screen; the items are selectable by the user.

list control: A custom control created by the List Manager.

list control record: A data structure that defines the appearance of a list control after the control has been created.

list record: A data structure that defines the initial appearance of a list control.

List Manager: The Apple IIGS Tool set that allows an application to present the user with a list from which to choose (for example, the Font Manager uses the List Manager to arrange lists of fonts).

local coordinates: A coordinate system unique to each GrafPort and independent of the global coordinates of the pixel image that the port is associated with. For example, local coordinates do not change as a window is dragged across the screen; global coordinates do not change as a window's contents are scrolled.

location table: In a font, an array of integers with an entry for each character code.

locinfo: Acronym for location information. The data structure (record) that ties the coordinate plane to an individual pixel image in memory.

Glossary G-15
lock: To prevent a memory block from being moved or purged. A block may be locked or unlocked by a call to the Memory Manager. Compare unlock.

long, long word: On the Apple IIGS, a 32-bit (4-byte) data type.

Longint value: A 32-bit signed or unsigned value.

Macintosh: A family of Apple computers; for example, the Macintosh 512K and the Macintosh Plus. Macintosh computers have high-resolution screens and use mouse devices for choosing commands and for drawing pictures.

macro: A single keystroke or command that a program replaces with several keystrokes or commands. For example, the APW Editor allows you to define macros that execute several editor keystroke commands; the APW Assembler allows you to define macros that execute instructions and directives. Macros are almost like higher-level language instructions, making assembly-language programs easier to write and complex keystrokes easier to execute.

macro library: A file of related macros.

main event loop: The central routine of an event-driven program. During execution, the program continually cycles through the main event loop, branching off to handle events as they occur and then returning to the event loop.

mainlD: A subfield of the user ID. Each running program is assigned a unique mainlD field.

manager: See tool set.

mask: A parameter, typically one or more bytes long, whose individual bits are used to set or block particular features. See, for example, event mask.

master color value: A 2-byte number that specifies the relative intensities of the red, green, and blue signals output by the Apple IIGS video hardware.

master pointer: A fixed location that always contains the address of a specified block, regardless of whether the block is moved. See also memory handle.

master user ID: The value of a user ID disregarding the contents of the auxlD field. If an application allocates various memory blocks and assigns them unique IDs consisting of auxlD values added to its own user ID, then all will share the same master user ID and all can be purged or disposed with a single call.

Mb: See megabyte.

megabyte (Mb): A unit of computer memory or disk drive capacity equaling 1,048,576 bytes.

megahertz (MHz): A unit of measurement of frequency, equal to 1,000,000 hertz (cycles per second).

memory attributes word: A word that determines how a specified memory block is allocated and maintained.

memory block: See block (2).

memory expansion card: A memory card that increases Apple IIGS internal memory capacity beyond 256K, up to 8 megabytes.

memory fragmentation: See fragmentation.

memory handle: A number that identifies a memory block. A handle is a pointer to a pointer; it is the address of a master pointer, which in turn contains the address of the block. Also called simply handle.

Memory Manager: The Apple IIGS Tool set that manages memory use. The Memory Manager keeps track of how much memory is available and allocates memory blocks to hold program segments or data.

menu: A list of choices presented by a program, from which the user can select an action. See also pull-down menu.
menu bar: The horizontal strip at the top of the screen that contains menu titles for the pull-down menus.

menu bar record: A data structure that contains the menu position, color, menu lists, item lists, and other flags the Menu Manager needs to manage menus.

menu definition procedure: A procedure used to define the appearance and behavior of a custom menu.

menu ID: A number in the menu record that identifies an individual menu.

menu item: On a menu, the text of a command or a line dividing groups of choices.

menu line: A line of text plus code characters that defines the appearance of a particular menu title.

Menu Manager: The Apple IIGS Tool Set that maintains the pull-down menus and the items in the menus.

menu record: A data structure that provides information about one of the menus in a menu bar.

m flag: One of three flags in the 65816 microprocessor's Processor Status register that controls execution mode. When the m flag is set to 1, the accumulator is 8 bits wide; otherwise, it is 16 bits wide. See also e flag, x flag.

MHz: See megahertz.

microprocessor: A central processing unit that is contained in a single integrated circuit. The Apple IIGS uses a 65C816 microprocessor.

minimum blink interval: The minimum time between blinks of the caret.

minimum version number: The minimum version of a particular tool set that an application needs to function.

minipalette: In Super Hi-Res 640 mode, a quarter of the color table. Each pixel in 640 mode can have one of four colors specified in a minipalette.

Miscellaneous Tool Set: The Apple IIGS tool set that includes mostly system-level routines that must be available for other tool sets.

missing character: In a font, a character that does not have a defined symbol.

missing symbol: In a font, the symbol substituted for any ASCII value for which the font does not have a defined symbol. In the Apple IIGS system font, the missing symbol is a box containing a question mark.

modal dialog box: A dialog box that puts the machine in a state where the user cannot execute functions outside of the dialog box until the dialog box is closed. Compare modeless dialog box.

mode: A state of a computer or system that determines its behavior. A manner of operating.

modeless dialog box: A dialog box that does not require the user to respond before doing anything else. Unlike a modal dialog box, it is possible to keep working even if the box is still in use. Compare modal dialog box.

modifier keys: The Shift, Caps Lock, Control, Option, and Apple keys. Such keys do not generate a keyboard event by themselves; rather, their states are recorded whenever another event is posted.

monospaced: Said of a font whose character widths are all identical. Compare proportionally spaced.

mouse: A small device the user moves around on a flat surface next to the computer. The mouse controls a pointer on the screen whose movements correspond to those of the mouse. The pointer selects operations, moves data, and draws graphic objects.
**mouse button**: A button on a mouse device with which the user selects objects on the screen.

**mouse clamps**: Values that establish the minimum and maximum X and Y coordinates for the mouse.

**mouse-down**: An action or an event, signifying that the user has pressed the mouse button.

**mouse-up**: An action or an event, signifying that the user has released the mouse button.

**movable**: Able to be moved to different memory locations during program execution (a memory block attribute). The opposite of fixed.

**native mode**: The 16-bit operating configuration of the 65C816 microprocessor.

**NDA**: See new desk accessory.

**new desk accessory (NDA)**: A desk accessory designed to execute in a desktop, event-driven environment. Compare classic desk accessory.

**NIL**: A value of 0. A pointer is NIL if its value is all zeros. A memory handle is NIL if the address it points to is filled with zeros. Handles to purged memory blocks are NIL.

**nonspecial, normal memory**: Memory that has no special restrictions on it. On the Apple IIGS, such memory includes banks $2-$DF and parts of banks $E0$ and $E1$.

**Note Sequencer**: The Apple IIGS tool set that makes it possible to play music asynchronously in programs.

**Note Synthesizer**: An Apple IIGS tool set that facilitates creation and manipulation of musical notes.

**null event**: An event reported when there are no other events to report.

**object module format (OMF)**: The file format used in Apple IIGS object files, library files, and load files. Compare text file format.

**offset**: The number of character positions or memory locations away from a point of reference.

**OK**: One of two predefined item ID numbers for dialog box buttons (OK = 1). Compare Cancel.

**OMF**: See object module format.

**operating environment**: The overall hardware and software setting within which a program runs. Also called execution environment.

**operating system**: A general-purpose program that organizes the actions of the various parts of the computer and its peripheral devices. See also disk operating system.

**origin**: (1) The first memory address of a program or of a portion of one. The first instruction to be executed. (2) The location (0,0) on the QuickDraw II coordinate plane, in either global coordinates or local coordinates. (3) The top left corner of any rectangle (such as a boundary rectangle or a port rectangle) in QuickDraw II. (4) See character origin.

**oscillator**: A device that generates a vibration. In the Apple IIGS digital oscillator chip, an oscillator is an address generator that points to the next data byte in memory that represents part of a particular sound wave.

**oval**: A circle or an ellipse, one of the fundamental classes of objects drawn by QuickDraw II.

**pack**: To compress data into a smaller space to conserve storage space.

**page**: (1) A portion of memory 256 bytes long and beginning at an address that is an even multiple of 256. Memory blocks whose starting addresses are an even multiple of 256 are said to be page aligned. (2) An area of main memory containing text or graphic information being displayed on the screen.

**page-aligned**: Said of a memory block that starts at a memory address that is an even multiple of 256 (a memory block attribute). See also page (1).
**paging region:** in a scroll bar, the area a user clicks to move the view of the data a page at a time.

**painting:** in QuickDraw II, using the current pen pattern, drawing mask, and pen mode to fill the interior of a shape.

**palette:** the full set of colors available for an individual screen pixel.

**parameter:** a value passed to or from a function or other routine.

**parameter RAM:** RAM on the Apple IIGS clock chip. A battery preserves the clock settings and the RAM contents when the power is off. Control Panel settings are kept in battery RAM.

**part code:** a number between 1 and 255 that stands for a particular part of a control. The Control Manager uses part codes to allow different parts of a control to respond in different ways.

**partial pathname:** a pathname that includes the filename of the desired file but excludes the volume directory name (and possibly one or more of the subdirectories in the path). It is the part of a pathname following a prefix; a prefix and a partial pathname together constitute a full pathname. A partial pathname does not begin with a slash because it has no volume directory name.

**Pascal:** a high-level programming language. Named for the philosopher and mathematician Blaise Pascal.

**Pascal string:** an ASCII character string preceded by a single byte whose numerical value is the number of characters in the string. Compare C string.

**Pascal-type string:** same as Pascal string.

**paste:** to place the desk scrap (contents of the Clipboard—whatever was last cut or copied) at the insertion point.

**pathname:** a name that specifies a file. It is a sequence of one or more filenames separated by slashes, tracing the path through subdirectories that a program must follow to locate the file. See also full pathname, partial pathname, prefix.

**pathname prefix:** see prefix.

**pattern:** an 8-by-8 pixel image, used to define a repeating design (such as stripes) or color.

**PB register:** see program bank register.

**PC register:** a register within the 65816 microprocessor that keeps track of the memory address of the next instruction to be executed. PC stands for program counter.

**pen:** the conceptual tool with which QuickDraw II draws shapes and characters. Each GrafPort has its own pen.

**pen location:** the position (on the coordinate plane) at which the next character or line will be drawn.

**pen mode:** one of several Boolean operations that determine how the pen pattern is to affect an existing image. Compare text mode.

**pen pattern:** see pattern.

**pen size:** the size of the rectangle that will be used as the drawing pen.

**peripheral card:** a hardware device placed inside a computer and connected to one of the computer's peripheral expansion slots. Peripheral cards perform a variety of functions, from controlling a disk drive to providing a clock/calendar.

**peripheral device:** see device.

**picture:** a saved sequence of QuickDraw II drawing commands (and, optionally, picture comments) that you can play back later with a single procedure call; also, the image resulting from these commands.
pining: The process of assigning positive overflows to the largest positive representable value and negative overflows to the largest negative representable value.

pixel: A contraction of picture element, the smallest dot you can draw on the screen. Also a location in video memory that corresponds to a point on the graphics screen when the viewing window includes that location. In the Super Hi-Res display on the Apple II, each pixel is represented by either 2 or 4 bits. See also pixel image.

pixel image: A graphics image picture consisting of a rectangular grid of pixels.

plain-styled: Said of a font or character that is not bold, italicized, underlined, or otherwise styled apart from ordinary text.

plane: The front-to-back position of a window on the desktop.

point: (1) A unit of measurement for type; 12 points equal 1 pica, and 6 picas equal 1 inch; thus, 1 point equals 1/72 inch. (2) A relative measure (taken from the type measure) used to distinguish font size on output devices. (3) In QuickDraw II, the Y and X coordinates of a location on the coordinate plane.

pointer: An item of information consisting of the memory address of some other item. For example, the 65C816 stack register contains a pointer to the top of the stack.

pointing device: Any device, such as a mouse, graphics tablet, or light pen, that can be used to specify locations on the computer screen.

polygon: Any sequence of connected lines.

port: (1) A socket on the back panel of the computer where the user can plug in a cable to connect a peripheral device, another computer, or a network. (2) A graphic port (GrafPort).

portrait mode: A printing mode in which text prints from left to right on the paper.

port rectangle: A rectangle that describes the active region of a GrafPort's pixel map—the part that QuickDraw II can draw into. The content region of a window on the desktop corresponds to the window's port rectangle.

portRect: The GrafPort field that defines the port's port rectangle.

post: To place an event in the event queue for later processing.

prefix: A pathname starting with a volume name and ending with a subdirectory name. It is the part of a full pathname that precedes a partial pathname; a prefix and a partial pathname together constitute a full pathname. A prefix always starts with a slash (/) because a volume directory name always starts with a slash.

prestyled: Said of a font that has a certain style or combination of styles built into the font's design.

printer information subrecord: A data structure within the print record that contains the information needed for page composition.

printing loop: The page-by-page cycle that an application goes through when it prints a document.

Print Manager: The Apple II tool set that allows an application to use standard QuickDraw II routines to print text or graphics on a printer.

print record: A record containing all the information needed by the Print Manager to perform a particular printing job.

private scrap: A buffer (and its contents) set up by an application for cutting and pasting, analogous to but apart from the desk scrap.

ProDOS: Acronym for Professional Disk Operating System. A family of disk operating systems developed for the Apple II family of computers. It includes ProDOS 8 and ProDOS 16.
ProDOS 8: A disk operating system developed for standard Apple II computers. It runs on 6502-series microprocessors and on the Apple IIGS when the 65C816 processor is in 6502 emulation mode.

ProDOS 16: A disk operating system developed for 65C816 native-mode operation on the Apple IIGS. It is functionally similar to ProDOS 8 but more powerful.

Program bank register: The 65C816 register whose contents form the high-order byte of all 3-byte code address operands. Also called PB register.

Program counter: See PC register.

Program status register: A register in the 65C816 microprocessor that contains flags reflecting the various aspects of machine state and operation results.

Proportionally spaced: Said of a font whose characters vary in width, so the amount of horizontal space needed for each character is proportional to its width. Compare monospaced.

Pseudo-type: A type that provides some additional information about a parameter of a toolbox routine.

Pull-down menu: A set of choices for actions that appears near the top of the display screen in a desktop application, usually overlaying the present contents of the screen without disrupting them. Dragging through the menu and releasing the mouse button while a command is highlighted chooses that command.

Purge: To temporarily deallocate a memory block. The Memory Manager purges a block by setting its master pointer to NIL (0). All handles to the pointer are still valid, so the block can be reconstructed quickly. Compare dispose.

Purge level: A memory block attribute, indicating that the Memory Manager may purge the block if it needs additional memory space. Purgeable blocks have different purge levels, or priorities for purging; these levels are set by Memory Manager calls.

Queue: A list in which entries are added (pushed) at one end and removed (pulled) at the other end, causing entries to be removed in first-in, first-out (FIFO) order. Compare stack.

QuickDraw II: The Apple IIGS tool set that controls the graphics environment and draws simple objects and text. Other tools call QuickDraw II to draw such things as windows.

QuickDraw II Auxiliary: The Apple IIGS tool set that provides extensions to the capabilities of QuickDraw II.

QuickDraw II Auxiliary Icon Record: A data structure that defines the appearance of an icon.

Quit: To terminate execution in an orderly manner. Apple IIGS applications quit by making a ProDOS 16 QUIT call or the equivalent.

Radio button: A common type of control in dialog boxes. Radio buttons are small circles organized into families; clicking any button on turns off all the others in the family, like the buttons on a car radio. See also check box.

RAM: See random-access memory.

Random-access memory (RAM): Memory in which information can be referred to in an arbitrary or random order. Programs and other data in RAM are lost when the computer is turned off. Technically, the read-only memory (ROM) is also random access, and what's called RAM should correctly be termed read-write memory. Compare read-only memory.

Range mode: In the List Manager, a selection mode that allows the user to select a range of members in a list.
read-only memory (ROM): Memory whose contents can be read, but not changed; used for storing firmware. Information is placed into ROM once, during manufacture; it then remains there permanently, even when the computer’s power is turned off. Compare random-access memory.

real font: A font that exists on disk or was added by an application and marked as real. Compare unreal font.

rectangle: One of the fundamental shapes drawn by QuickDraw II. Rectangles are completely defined by two points—their upper left and lower right corners on the coordinate plane. The upper left corner of any rectangle is its origin.

reentrant: Said of a routine that is able to accept a call while one or more previous calls to it are pending, without invalidating the previous calls. Under certain conditions, the Apple IIGS Scheduler manages execution of routines that are not reentrant.

region: An arbitrary area or set of areas on the QuickDraw II coordinate plane. The outline of a region must be one or more closed loops.

relocatable: Characteristic of a load segment or other program code that includes no references to specific address, and so can be loaded at any memory address. A relocatable segment consists of a code image followed by a relocation dictionary. Compare absolute.

relocation: The act of modifying a program in memory so that its address operands correctly reflect its location and the locations of other segments in memory. Relocation is performed by the system loader when a relocatable segment is first loaded into memory.

repeat delay: The time interval before the first auto-key event is generated.

repeat speed: The time interval between auto-key events, except for the first auto-key event. See also repeat delay.

reserved memory: Memory not managed by the Memory Manager; that is, memory that is marked as busy at startup time.

right scroll bar: The control the user selects to scroll vertically through the data in the window.

ROM: See read-only-memory.

ROM font: The font contained in system ROM.

rounded result: The nearest representable value to the actual value, with ties going to the value with the larger magnitude.

rounded-corner rectangle: One of the fundamental shapes drawn by QuickDraw II. The rounded corners of this type of rectangle are defined by an oval height and oval width.

routine: A part of a program that accomplishes some task subordinate to the overall task of the program.

RTL (Return from Subroutine Long): A 65816 assembly-language instruction.

RTS (Return from Subroutine): A 65816 assembly-language instruction.

SANE (Standard Apple Numeric Environment): The set of methods that provides the basis for floating-point calculations in Apple computers. SANE meets all requirements for extended-precision, floating-point arithmetic as prescribed by IEEE Standard 754 and ensures that all floating-point operations are performed consistently and return the most accurate results possible.

SANE Tool Set: The Apple IIGS tool set that performs high-precision, floating-point calculations, following SANE standards.

scaled font: A font that is created by the Font Manager by calculation from a real font of a different size.

scaling: The process of taking all characters of a real font and making them bigger or smaller to generate a requested font.
scan line: A single horizontal line of pixels on the screen. It corresponds to a single sweep of the electron gun in the video display tube.

scan line control byte (SCB): A byte in memory that controls certain properties, such as available colors and number of pixels, for a scan line on the Apple IIGS. Each scan line has its own SCB.

SCB: See scan line control byte.

Scheduler: The Apple IIGS tool set that manages requests to execute interrupted software that is not reentrant. If, for example, an interrupt handler needs to make system software calls, it must do so through the Scheduler because ProDOS 16 is not reentrant. Applications normally need not use the Scheduler because ProDOS 16 is not in an interrupted state when it processes applications’ system calls.

scrap count: A count that indicates how many times the desk scrap has changed.

Scrap Manager: The Apple IIGS tool set that supports the desk scrap, which allows data to be copied from one application to another or from one place to another within an application.

scroll: To move an image of a document or directory in its window so that a different part of it is visible.

scroll bar: A rectangular bar that may be along the right or bottom of a window. Clicking or dragging in the scroll bar causes the view of the document to change.

selection range: The series of characters where the next editing action will occur.

serial interface: A standard method, such as RS-232, for transmitting data serially (as a sequence of bits).

serial port: The connector for a peripheral device that uses a serial interface.

Shaston: The Apple IIGS system font.

shut down: To remove from memory or otherwise make unavailable, as a tool set that is no longer needed or an application that has quit.

single mode: In the List Manager, a selection mode that allows the user to select only one member of a list at once; that is, when the user drags the mouse, the selection moves from one member to another.

65C816: The microprocessor used in the Apple IIGS. The 65C816 is a CMOS device with a 16-bit data bus and a 24-bit address bus.

6502: The microprocessor used in the Apple II, Apple II Plus, and early models of the Apple IIe. The 6502 is an NMOS device with 8-bit data registers and 16-bit address registers.

65816 assembly language: A low-level programming language written for the 65816 family of microprocessors.

640 mode: An Apple IIGS video display mode, 640 pixels horizontally by 200 pixels vertically.

size box: A small region in the lower right corner of a window that the user can drag to change the size of the window.

slop rectangle: The rectangle that allows the user some margin for error when moving the mouse.

slot: A narrow socket inside the computer where the user can install peripheral cards. Also called expansion slot.

smoothing: A LaserWriter printing option that asks the system to smooth out any bit-mapped fonts with jagged edges.

software: A collective term for programs, the instructions that tell the computer what to do. Software is usually stored on disks. Compare firmware, hardware.

sound GLU (general logic unit): The interface chip between the system hardware and the sound hardware.
**Sound Tool Set:** The Apple IIGS tool set that provides low-level access to the sound hardware.

**source location:** The location (memory buffer or portion of the QuickDraw II coordinate plane) from which data such as text or graphics are copied. See also destination rectangle.

**source rectangle:** The rectangle (on the QuickDraw II coordinate plane) where text or graphics are drawn when transferred from somewhere else. Compare destination rectangle.

**special memory:** On an Apple IIGS, all of banks $00 and $01 and all display memory in banks $E0 and $E1. It is the memory directly accessed by standard Apple II programs running on the Apple IIGS.

**spool printing:** A two-step printing method used to print graphics on the ImageWriter. In the first step, it writes out (spools) a representation of your document's printed image to a disk file or to memory. The second step consists of this information being converted into a bit image and printed. Compare draft printing.

**S register:** See stack register.

**SRQ list:** A special tool mechanism that can be used to poll the Apple Desktop Bus for data from specific devices.

**SRQ list completion routine:** Used in conjunction with the ADB Tool Set routine AsyncADBReceive, this completion routine obtains ADB data from a buffer. The only major difference between this routine and the AsyncADBReceive completion routine is that the SRQ list routine has an extra return address on the stack when it is called. Compare AsyncADBReceive completion routine.

**stack:** A list in which entries are added (pushed) and removed (pulled) at one end only (the top of the stack), causing entries to be removed in last-in, first-out (LIFO) order. The stack usually refers to the particular stack pointed to by the 65C02's stack pointer. Compare queue.

**stack pointer:** See stack register.

**stack register:** A register in the 65816 processor that indicates the next available memory address in the stack. Also called S register.

**stage byte:** Determines the actions taken by an alert. See also alert stage.

**Standard Apple Numerics Environment:** See SANE.

**standard Apple II:** Any computer in the Apple II family except the Apple IIGS. That includes the original Apple II, the Apple II Plus, the Apple IIe, and the Apple IIc. Compare Apple II.

**standard File Operations Tool Set:** The Apple IIGS tool set that creates a standard user interface for opening and closing files.

**standard window controls:** The window controls that allow the user to scroll through the data in the window, change the window's shape, or close the window. They also provide information about the document currently displayed in the window.

**start up:** To get the system, application program, or tool set running.

**static segment:** A program segment that must be loaded when the program is started and cannot be removed from memory until execution terminates. Compare dynamic segment.

**static text:** Text on the screen that cannot be altered by the user.

**string:** A sequence of characters. See also C string, Pascal string.
**string bounds rectangle:** The smallest rectangle that would enclose all the foreground and background pixels of a string if the string were to be drawn.

**structure region:** An entire window: its content region plus its frame region.

**style dialog box:** A dialog box that allows the user to specify formatting information, page size, and printer options.

**styled variation:** An italicized, bold, underlined, or otherwise altered version of a plain-styled character or font.

**style subrecord:** A data structure within the print record that contains information gathered from the user via the style and job dialog boxes.

**subdirectory:** A file that contains information about other files: In a hierarchical file system, files are accessed through the subdirectories that reference them.

**subroutine:** A part of a program that can be executed on request from another point in the program and that, upon completion, returns control to the point of the request.

**Super Hi-Res:** Either of two high-resolution Apple II GS display modes: 320 mode consists of an array of pixels 320 wide by 200 high, with 16 available colors; 640 mode is an array 640 wide by 200 high, with 16 available colors (with restrictions).

**swap pair:** A pair of oscillators that form a functional unit (called a generator) when the digital oscillator chip (DOC) is in swap mode.

**switcher:** A program that rapidly transfers execution among several applications.

**switch event:** An event that indicates the application is being returned to after being switched out of by a switcher-type application.

**synthesizer:** (1) A hardware device capable of creating sound digitally and converting it into an analog waveform that you can hear. (2) By analogy, any sound-making entity, such as the Free-Form Synthesizer in the Sound Tool Set.

**system clock:** See clock (1).

**system disk:** A disk that contains the operating system and other system software needed to run applications.

**system event mask:** A set of flags that controls which event types get posted into the event queue by the Event Manager.

**system failure:** The unintentional termination of program execution due to a severe software error.

**System Failure Manager:** A part of the Miscellaneous Tool Set that processes fatal errors by displaying a message on the screen and halting execution.

**system folder:** The SYSTEM/subdirectory on a ProDOS 16 system disk.

**system font:** The font that QuickDraw II uses as the default current font when a new GrafPort is opened.

**System Loader:** The program that manages the loading and relocation of load segments (programs) into the Apple II GS memory. The System Loader works closely with ProDOS 16 and the Memory Manager.

**system menu bar:** The menu bar that always appears at the top of the screen in desktop applications. It contains all of the commonly used functions such as File, Edit, and so on. Compare window menu bar.

**system software:** The components of a computer system that support application programs by managing system resources such as memory and I/O devices.

**system window:** A window in which a desk accessory is displayed.
**task code:** A numeric value assigned to the result of each event handled by TaskMaster. Compare event code.

**TaskMaster:** A Window Manager routine that handles many typical events for an application. Applications may call TaskMaster instead of GetNextEvent.

**template:** A data structure or set of parameters that defines the characteristics of a desktop feature, such as a window control. The NewWindow parameter list is a template that defines the appearance of a window to be opened by the NewWindow call.

**text block:** A number of ASCII characters in a buffer, with the number specified separately.

**text buffer:** A 1-bit-per-pixel pixel image reserved for the private use of the QuickDraw II text-drawing call.

**text file format (TFF):** A file that consists of ASCII representations of characters. Compare object module format.

**text mode:** One of eight possible interactions between pixels in text being drawn to the screen and pixels on the screen that fall under characters being drawn. Compare pen mode.

**Text Tool Set:** The Apple II GS tool set that provides an interface between Apple II character device drivers and applications running in native mode.

**TFF:** See text file format.

**320 mode:** An Apple II GS video display mode, 320 pixels horizontally by 200 pixels vertically.

**tick count:** The (approximate) number of 60th-second intervals since system startup.

**title bar:** The horizontal bar at the top of a window that shows the name of the window’s contents. The user can move the window by dragging the title bar.

**toolbox:** See Apple II GS Toolbox.

**Tool Locator:** The Apple II GS tool set that dispatches tool calls. The Tool Locator knows and retrieves the appropriate routine when your application makes a tool call.

**tool pointer table (TPT):** A table, maintained by the Tool Locator, that contains pointers to all active tool sets.

**tool set:** A group of related routines (usually in ROM) that perform necessary functions or provide programming convenience. They are available to applications and system software. The Memory Manager, the System Loader, and QuickDraw II are Apple II GS tool sets.

**tool table:** A list of all needed tool sets and their minimum required versions. An application constructs this table in order to load its tools with the LoadTools call.

**TPT:** See tool pointer table.

**transfer mode:** A specification of which Boolean operation QuickDraw II should perform when drawing. See, for example, XOR.

**TRUE:** Not zero. The result of a Boolean operation. The opposite of FALSE.

**typeID:** A subfield of the user ID. The User ID Manager assigns a typeID value based on the type of program (application, tool set, and so on) requesting the memory.

**unclaimed interrupt:** This occurs when the hardware Interrupt Request Line is active, indicating that an interrupt-producing device needs attention, but none of the installed interrupt handlers claims responsibility for the interrupt.

**underline:** (1) A style of text. (2) A method used to separate groups of items in a menu. An underlined item does not use any more space, on the screen or in memory, than the item does without the underline. Compare dividing line.
unhighlight: To restore to normal display. Selected controls, menu items, or other objects may be highlighted (usually displayed in inverse colors) while in use and unhighlighted when not in use.

unload: To remove a load segment from memory. To unload a segment, the System Loader does not actually "unload" anything; it calls the Memory Manager to either purge or dispose of the memory block in which the code segment resides.

unlock: To permit the Memory Manager to move or purge a memory block if needed. Compare lock.

unmovable: See fixed.

unpack: To restore to normal format from a packed format.

unpurgeable: Having a purge level of zero. The Memory Manager is not permitted to purge memory blocks whose purge level is zero.

unreal font: A font that was scaled by the Font Manager from a real font of a different size or added by an application and marked as unreal. Compare real font.

update event: An event posted by the Window Manager when all or part of a window needs to be redrawn.

user ID: An identification number that specifies the owner of every memory block allocated by the Memory Manager. User ID's are assigned by the User ID Manager.

User ID Manager: A part of the Miscellaneous Tool Set that is responsible for assigning user ID's to every block of memory allocated by the Memory Manager.

vector: A location that contains a value used to find the entry point address of a subroutine.

view rectangle: The rectangle within which text in an edit record is visible; that is, the portion of the text in the destination rectangle that the user can see is determined by the view rectangle.

visible region: The part of a window that's actually visible on the screen. The visible region is defined by a GrafPort field manipulated by the Window Manager.

voice: Any one of 16 pairs of oscillators in the Ensoniq sound chip on the Apple II GS.

wedge: A filled arc, one of the fundamental shapes drawn by QuickDraw II.

window: A rectangular area that displays information on a desktop. You view a document through a window. You can open or close a window, move it around on the desktop, and sometimes change its size, scroll through it, and edit its contents. The area inside the window's frame corresponds to the port rectangle of the window's GrafPort.

window definition procedure: A procedure used to define the appearance and behavior of a custom window.

window frame: The outline of the entire window plus certain standard window controls.

Window Manager: The Apple II GS Tool Set that updates and maintains windows.

window menu bar: A menu bar that appears at the top of the active window, below the system menu bar. It can contain document titles, applications, and functions. Compare system menu bar.

window record: The internal representation of a window, where the Window Manager stores all the information it needs for its operations on that window.

word: On the Apple II GS, a 16-bit (2-byte) data type. Compare long, long word.

x flag: One of three flag bits in the 65C816 processor that programs use to control the processor's operating modes. In native mode, the setting of the x flag determines whether the index registers are 8 bits wide or 16 bits wide. See also c flag, m flag.
**XOR:** Exclusive-OR. A Boolean operation in which the result is TRUE if, and only if, the two items being compared are unequal in value.

**X register:** One of the two index registers in the 65816 microprocessor.

**Y register:** One of the two index registers in the 65816 microprocessor.

**zero page:** The first page (256 bytes) of memory in a standard Apple II computer (or in the Apple IIgs when running a standard Apple II program). Because the high-order byte of any address in this part of memory is zero, only a single byte is needed to specify a zero-page address. Compare direct page.

**z flag:** A bit in the 65816 processor's Processor Status register that is set to 1 if the last operation resulted in 0 (zero).

**zoom box:** A small box with a smaller box enclosed in it found on the right side of the title bar of some windows. Clicking the zoom box expands the window to its maximum size; clicking it again returns the window to its original size.

**zoom region:** The window region that corresponds to the zoom box.
C, calling routines from 2-6
cairo 8-4, 8-51
calcCRect 4-25
calcRect 4-86
calling tool set routines
  assembly language 2-5
C 2-6
cancel 6-89
cancel button 6-4, 6-5, 6-18, 6-21, 6-58, 8-14
cancelDefault 6-89
cannotReset 16-278, B-3
cantSync 3-29, B-4
capsLock 7-9, 7-51
caret 7-36, 10-8, 10-9, 10-11, 10-20, 10-34, 10-38, 10-46
  blinking 10-10, 10-26
  carriage return character 10-40, 10-42
  carry flag. See c flag
  Caution alert 6-6

CautionAlert 6-24, 6-35
caution icon 6-35
CDA header section 5-18
CDA menu 5-3, 5-12, 5-21
CDA name 5-16, 5-23
c flag 2-5, 2-7, B-1
Change Address command 3-4
changeFlag 7-9, 7-10, 7-51
change flag byte 10-43
channel-generator type word 21-16
character 16-26
dead 16-45
  missing 16-48-49
character bounds rectangle 16-28, 16-52, 16-69, 16-76, 16-269
character bounds width 16-52
character device driver 1-4, 23-1, 23-3
character echo-flag word 23-29
character image 16-26
character image width 16-44
character origin 16-26-27, 16-45
character position 10-7
character rectangle 16-44
character width 16-26-27, 16-44, 16-45, 16-51, 16-70, 16-77, 16-267, 16-270
CharBounds 16-29, 16-56, 16-58, 16-69
CharWidth 16-29, 16-56, 16-58, 16-70
check box 4-3, 4-10, 4-16
  control record 4-16-18
checkBox 4-86
CheckHandle 12-22
checkItem 6-10, 6-88
CheckMitem 13-6, 13-34
checkProc 4-13, 4-73, 4-85
CheckUpdate 25-37
chExtra 16-30, 16-51, 16-55, 16-58, 16-59, 16-60, 16-112, 16-217
ChooseCDA 5-12
ChooseFont 8-13, 8-14, 8-15, 8-26-27, C-5
Choose Printer dialog box 15-1, 15-4-5
chunky pixel 16-31
clamp
  absolute 14-21
  absolute device 14-37, 14-38
ClampMouse 7-27, 14-30
clamp value 2-2, 7-27, 14-30, 14-31
classic desk accessory 5-1
clear 5-30
clearAction 5-7, 5-30
Clear command 5-43
clearModes 3-20, 3-28
ClearMouse 14-31
ClearScreen 16-71
ClearSRQTable 3-15
Clipboard 20-1, 20-5, 20-17
clipping region 16-14-15, 16-17, 16-72, 16-113, 16-114, 16-187, 16-197, 16-208, 16-218, 17-10
ClipRect 6-7, 16-40, 16-72
clockFormat 14-65
Close 5-14
CloseAllNDA 5-12
close box 25-6, 25-8, 25-49
CloseDialog 6-23, 6-36
CloseNDA 5-6, 5-13
CloseNDAbyWinPtr 5-6, 5-14, 25-122
ClosePicture 17-5, 17-9
ClosePoly 16-40, 16-72
ClosePrint 16-39, 16-73
CloseRgn 16-40, 16-74
CloseWindow 4-9, 4-68, 11-11, 11-16, 25-11, 25-29, 25-38, 25-127
ClrHeartBeat 14-53
cmdincomplete 3-14, 3-17, 3-18, 3-19, 3-25, 3-26, 3-27, 3-29, B-4
Collision Detect handler 3-4-5
color
  background 16-26, 16-213
  foreground 16-26, 16-228
  list 11-10
  window frame 25-18, 25-57, 25-99
  window information bar 25-20
  window size box 25-19
  window title 25-18
  window title bar 25-19
color box 15-8

Index 1-3
colorMItemHilite 13-78, 13-87
color palette. See palette
color printing 15-8, 15-15-18
color table 11-10, 13-43,
   16-31-33, 16-115, 16-116,
   16-158, 16-220, 16-221
alert window 25-17
document window 25-17
standard 16-159
window 25-142
colorTable 15-8, 15-15-18
colorReplace 13-72, 13-87
color table 11-10, 13-43, 16-31-33, 16-115, 16-116, 16-158, 16-220, 16-221
alert window 25-17
document window 25-17
standard 16-159
window 25-142
colorTable 15-8, 15-15-18
colorReplace 13-72, 13-87
content width, maximum 25-87
control 1-3, 4-1
   active 4-7
   custom 4-24
   defining 4-24-40
   highlighted 4-7
   inactive 4-7
   standard 4-3-4
   window 4-8, 25-6-7
control code 23-15
control definition procedure 4-24,
   4-46, 4-73, 11-1, 11-11
Control-F 10-2, 10-29
control flag 4-12, 4-13, 4-72
   check box 4-18
   radio button 4-14
   scroll bar 4-22
   simple button 4-16
   size box 4-20
controlKey 7-9, 7-51
control list 25-68
Control Manager 1-3, 1-5, 4-1-88,
   6-28, 25-9
counts 4-85-86
data structures 4-87-88
error codes 4-88
icon font 4-11
part codes 4-8
shutdown routine 4-43
startup routine 4-42
status routine 4-44
using 4-9-10
version number routine 4-43
Control Panel 3-1, 5-3, 5-22, 6-21, 7-3, 7-36, 7-37, 10-26, 13-76
control record 4-11-24
check box 4-16-18
radio button 4-18-20
scroll bar 4-20-22
simple button 4-14-16
size box 4-23-24
Control-X 10-2, 10-29
Control-Y 10-2, 10-29
counter specification xxvi
counts 4-85-86
counter width, maximum 25-87
count word 14-49
courier xxxi
cpMgr 14-67
copy 5-30
copyAction 5-7, 5-30
copy command 6-41, 20-1, 20-6
copyPixels 17-5, 17-10
copyRgn 16-75
cosine 9-20
CountFamilies 8-28
CountFonts 8-15, 8-29-30
CountMitems 13-13, 13-35
count word 14-49
courier xxxi, 8-4, 8-51
CreateControl 11-10
CreateList 11-5, 11-11, 11-16,
   11-23
crsrUpdHnd 14-68
crWidth 15-12, 15-13
c string 16-26, 16-41
cString 11-25
CStringLength 16-29, 16-58,
   16-76
CStringLength 16-29, 16-58, 16-77
cntAction 4-12, 4-13, 4-15, 4-19,
   4-21, 4-23, 4-59, 4-74, 4-87,
   11-6, 13-17
cntBootlnit 4-41
cntColor 4-12, 4-14, 4-15, 4-16,
   4-17, 4-19, 4-21, 4-23, 4-87,
   11-6, 11-10, 13-17
cntData 4-12, 4-14, 4-15, 4-17,
   4-19, 4-21, 4-23, 4-63, 4-78,
   4-87, 11-6, 13-17
cntFlag 4-12, 4-13, 4-15, 4-17,
   4-19, 4-21, 4-23, 4-87, 11-6,
   13-17
cntHilite 4-12, 4-13, 4-15, 4-17,
   4-19, 4-21, 4-23, 4-87, 11-6,
   13-17
cntList 11-6, 11-26
cntListBar 11-6, 11-26

conventions

boldface xxxi
courier xxxi
italic xxxi
coordinate plane 16-9-14
copies 15-8
copyAction 5-7, 5-30
copy command 6-41, 20-1, 20-6
copyPixels 17-5, 17-10
copyRgn 16-75
cosine 9-20
CountFamilies 8-28
CountFonts 8-15, 8-29-30
CountMitems 13-13, 13-35
count word 14-49
courier xxxi, 8-4, 8-51
CreateControl 11-10
CreateList 11-5, 11-11, 11-16,
   11-23
crsrUpdHnd 14-68
crWidth 15-12, 15-13
c string 16-26, 16-41
cString 11-25
CStringLength 16-29, 16-58,
   16-76
CStringLength 16-29, 16-58, 16-77
cntAction 4-12, 4-13, 4-15, 4-19,
   4-21, 4-23, 4-59, 4-74, 4-87,
   11-6, 13-17
cntBootlnit 4-41
cntColor 4-12, 4-14, 4-15, 4-16,
   4-17, 4-19, 4-21, 4-23, 4-87,
   11-6, 11-10, 13-17
cntData 4-12, 4-14, 4-15, 4-17,
   4-19, 4-21, 4-23, 4-63, 4-78,
   4-87, 11-6, 13-17
cntFlag 4-12, 4-13, 4-15, 4-17,
   4-19, 4-21, 4-23, 4-87, 11-6,
   13-17
cntHilite 4-12, 4-13, 4-15, 4-17,
   4-19, 4-21, 4-23, 4-87, 11-6,
   13-17
cntList 11-6, 11-26
cntListBar 11-6, 11-26

Index
ctlMemDraw 11-6, 11-26
ctlMemHeight 11-6, 11-26
ctlMemSize 11-6, 11-26
CtlNewRes 4-45
ctlNext 4-12, 4-13, 4-15, 4-17, 4-19, 4-21, 4-23, 4-87, 11-6, 13-17
ctlOwner 4-12, 4-13, 4-15, 4-17, 4-19, 4-21, 4-23, 4-87, 11-6, 13-17
ctlProc 4-12, 4-13, 4-15, 4-17, 4-19, 4-21, 4-23, 4-87, 11-6, 13-17
ctlRect 4-12, 4-13, 4-15, 4-17, 4-19, 4-21, 4-23, 4-87, 11-6, 13-17
ctlRecCon 4-12, 4-14, 4-15, 4-17, 4-19, 4-21, 4-23, 4-62, 4-77, 4-87, 11-6, 13-17
CtlReset 4-44
CtlShutDown 4-9, 4-43
CtlStartUp 4-9, 4-42
CtlStatus 4-44
CtlTextDev 23-15-16
ctlValue 4-12, 4-13, 4-15, 4-17, 4-19, 4-21, 4-23, 4-64, 4-79, 4-87, 11-6, 13-17
CtlVersion 4-43
cntlVector 14-68
current font 8-8, 16-26, 16-119, 16-224
current icon font 25-109
cursorAction 5-7, 5-30
cursor record 16-37, 16-117, 16-222
custom control 4-24
custom menu 13-6
customMenu 13-72, 13-87
cut 5-30
cutAction 5-7, 5-30
Cut command 6-42, 20-1, 20-6
danotFound 5-14, 5-20, 5-30, B-3
darkGray320 16-274
darkGreen320 16-274
data bank 25-43
data structures
Apple Desktop Bus Tool Set 3-29
Control Manager 4-87-88
Dialog Manager 6-89-90
Event Manager 7-51
Font Manager 8-51
LineEdit Tool Set 10-47
List Manager 11-25-26
Menu Manager 13-88
Miscellaneous Tool Set 14-69
Print Manager 15-47-48
QuickDraw II 16-276-278
Sound Tool Set 21-37
Standard File Operations Tool Set 22-32
Window Manager 25-142-143
dateFormat 14-65
dblClickTime 14-65
dead character 16-45
deallociation of memory 12-1, 12-23, 12-24
decByteFlag 14-68
Dec2Int 9-8
Dec2Long 9-9
default button 6-5, 6-11, 6-18, 6-21, 6-24, 6-37, 6-51, 6-80
DefaultFilter 6-37
defProcParm 6-90
DeleteID 14-59
DeleteMenu 13-13, 13-36, 13-39
deleteMessage 24-15, 24-26
DeleteMItem 13-13, 13-37
DelHeartBeat 7-25, 14-52
delta flag byte 10-43
dependencies C-1-5
dereference 12-5
descender 16-45
descent 16-26, 16-45, 16-48, 16-62
descent line 16-45
DESK.ACCESS subdirectory 5-3, 5-6
desk accessory 1-3, 5-1
alternate-display-mode 5-24
classic 5-1
new 5-1
desk accessory event 7-4, 7-14
Desk Accessory menu 13-4-5
deskAccEvnt 7-7, 7-50
deskAccFlag 14-68
deskAccMask 7-11, 7-50
DeskBootInit 5-9
DeskManager 1-3, 1-5, 5-1-30
constants 5-30
error codes 5-30
GetNextEvent and 7-40
shutdown routine 5-10
startup routine 5-9
status routine 5-11
version number routine 5-10
DeskReset 5-11
desk scrap 1-3, 10-11, 10-23, 10-45, 20-1, 20-3, 20-5-6
data types 20-3-4
DeskShutDown 5-4, 5-5, 5-6, 5-7, 5-10
DeskStartUp 5-4, 5-5, 5-6, 5-7, 5-9
DeskStatus 5-11
Desktop 25-39-43
desktop 25-1, 25-39, 25-91
desktop pattern 25-43
desktop user interface xxvi
DeskVersion 5-10
destination rectangle 10-4, 10-6-7, 10-36
development environment xxvii
deDevErr 23-16, 23-47, B-4
device driver 3-3, 7-21
installing 7-23-24
removing 7-25
writing 7-21-23
device driver entry point 23-3
device driver event 7-4
devNotAtAddr 3-25, 3-29, B-4
dial 4-4
DialogBootInit 6-27
dialog box 1-3, 6-1, 6-4-6
dialog pointer 6-19
diagram item type 6-10-11
dialog window 6-7
direct page 6-28, 25-43
SANE 18-6, 18-9
direct page space 2-2, 4-42, 7-27, 10-13, 12-13, 13-30, 15-26, 16-64, 18-12, 21-8, 22-16
disable item 6-40, 6-82
disable menu item 6-13-6, 13-38
disable menu 6-13-6, 13-38
disable menu 6-13-6, 13-38
disableMenu 13-72, 13-87
disableMenuItem 6-40, 6-82
disabled menu item 13-6, 13-38
disable item 6-40, 6-82
disable menu item 6-13-6, 13-38
disable menu 6-13-6, 13-38
disableMenu 13-72, 13-87
disableMenuItem 6-40, 6-82
disable menu item 6-13-6, 13-38
disable menu 6-13-6, 13-38
disableMenu 13-72, 13-87
disableSRQ 3-21, 3-28
Disk button 22-24, 22-31
dispCtl 4-25, 4-30, 4-86
display mode word 17-4
display rectangle 6-17, 6-52, 6-81
F

FakeMouse 7-23, 7-34
fAlert 25-85-86, 25-141
eAllocated 25-141
family 16-43
family 4-85
family name 8-3, 8-32, 8-41, 8-42
family number 8-4, 8-31, 8-32, 8-41, 8-42
familyID 8-51
FamNumber 8-6, 8-51
FamNum2ItemID 8-31, 8-36
FamSpecBits 8-11-12
FamStatBits 8-12
Index
GetFontGlobals 16-30, 16-62, 16-121, 16-124, 16-150
GetFontID 8-16, 16-122
GetFontInfo 16-62, 16-123
GetFontLore 16-30, 16-62, 16-118, 16-121, 16-124, 16-150
GetForeColor 16-55, 16-125
GetFrameColor 25-20, 25-57
GetFuncPtr 24-3, 24-7
GetGrafProcs 16-126
GetHandle 13-39
GetHandleSize 12-27
GetInfoDraw 25-58
GetInfoRefCon 25-59
GetGlobals 23-24
getInitTotal 6-15, 6-88
getInitValue 6-15, 6-88
get initView 6-15, 6-88
GetInputDevice 23-25
GetIRQEnable 14-5, 14-23-24
GetIText 6-23, 6-56
GetListDefProc 11-18
GetMasterSCB 16-127
GetMaxGrow 25-60
GetMenuBar 13-44
GetMenuFlag 13-45
GetMenuMgrPort 13-46
GetMenuTitle 13-47
GetMessage 24-15, 24-26
GetMHandle 13-39, 13-48
GetMItem 13-16, 13-49
GetMItemFlag 13-50
GetMItemMark 13-6, 13-51
GetMItemStyle 13-6, 13-52
GetMouse 7-14, 7-38
GetMouseClamp 14-31
GetMTitleStart 13-53
GetMTitleWidth 13-54
GetNewDItem 6-23, 6-57-58
GetNewID 12-11, 12-35, 14-57-58
GetNewModalDialog 6-7, 6-23, 6-59-60, 22-4
GetNextDItem 6-61
GetNextEvent 4-10, 5-3, 6-33, 6-63, 6-66, 7-5, 7-12, 7-33, 7-39-40, 13-11, 25-11, 25-12, 25-118, 25-119
Desk Manager and 7-40
GetNextWindow 25-56, 25-61
GetNumNDAs 5-17
GetOSEvent 7-41
GetOutGlobals 23-26
GetOutputDevice 23-27
GetPage 25-62
GetPen 16-128
GetPenMask 16-129
GetPenMode 16-130
GetPenPat 16-131
GetPenSize 16-132
GetPenState 16-133
GetPicSave 16-134
GetPixel 16-135
GetPolySave 16-136
GetPort 16-39, 16-137, 25-21
GetPortLoc 16-138
GetPortRect 16-139
GetRectInfo 25-63
GetRgnSave 16-140
GetRomFont 16-141-142
GetSCB 16-143
GetScrap 20-5, 20-6, 20-10
GetScrapCount 20-5, 20-11
GetScrapPath 20-13
GetScrapSize 20-14
GetScrapState 20-15
getScroll 25-64
GetSoundVolume 21-21
GetSpaceExtra 16-55, 16-144
GetStandardSCB 16-145
GetStructRgn 25-65
GetSysBar 13-55
GetSysField 16-146
GetSysFont 16-147
GetSysWFlag 25-66
GetTableAddress 21-6, 21-22-23
GetTextFace 16-56, 16-148
GetTextMode 16-55, 16-149
GetTextSize 16-150
GetTick 14-22
GetTSPtr 24-3, 24-8
GetUpdateRgn 25-67
GetUserField 16-151
GetVector 7-23, 14-5, 14-61, 14-63
GetVisDesktop 25-41, 25-139
GetVisHandle 16-152
GetVisRgn 16-153
GetWAP 24-3, 24-9
GetWControls 25-68
GetWFrame 25-69
GetWKind 25-70
GetWMgrCon 25-72
GetWMgrPort 25-71
GetWTitle 25-73
GetZoomRect 25-74
global coordinates 4-50, 16-9, 16-16, 16-39, 16-154, 16-173, 25-29, 25-47
GlobalToLocal 16-39, 16-154
go-away region 25-9, 25-127
good 22-24, 22-32
GrafOff 16-155
GrafOn 16-155
current 16-241
standard 16-187
GrafProcs 16-126, 16-229, 16-277
graphic object 16-21
graphic port 16-9, 16-14
graphics tablet 7-21
gray scales 15-16, 15-17
green320 16-274
green640 16-274
greenMask 16-274
grid values 25-45
growBox 4-86
growColor 25-13, 25-142
grow image 25-75-76
growNorBack 4-24
growOutline 4-24
growProc 4-13, 4-23, 4-73, 4-85
grow region 25-9, 25-11
GrowSize 4-65
GrowWindow 25-11, 25-75-76, 25-125

H
halt mechanism 18-8
halt vector 18-6, 18-8, 18-9
handle xxx, 12-5-6
empty 12-8
index 1-11

HandToHand 12-28
HandToptr 12-29
hardware interrupt enable states 14-23
hAxisOnly 4-53, 4-86
hbQueueBadErr 14-70, B-2
header section 5-3, 5-6
HeartBeat Interrupt Handler 14-22, 14-27, 14-49
HeartBeat Interrupt Task queue 14-5, 14-27, 14-48, 14-50, 14-52, 14-53
Helvetica 8-4, 8-51
Heatx 9-27
HexInt 9-25
Hex2Long 9-26
HideControl 4-10, 4-45, 4-66, 11-11
HideCursor 14-156
HideDItem 6-24, 6-62, 6-81, 6-85
HidePen 14-156, 17-9, 17-14
high-level Event Manager 7-3
highlighted control 4-7
high-order word 9-28
HiliteControl 4-7, 4-67, 6-11, 6-40, 11-11
HiliteMenu 13-11, 13-12, 13-56, 13-66, 25-122, 25-123
HiliteWindow 25-36, 25-79
HiMouseRes 14-65
HWord 9-28
HLock 14-15, 12-30
HLockAll 14-15, 12-31
HomeMouse 14-32
horizontal resolution movement 25-45
horScroll 4-22, 4-72, 4-85
hot spot 16-37
hPic 15-41, 15-48
hPrint 15-41, 15-48
hrtz50or60 14-64
Human Interface Guidelines xxviii, 6-7, 6-8, 6-18, 6-22, 6-25, 10-1, 13-1, 13-7, 13-9, 15-20, 25-79
HUnlock 12-15, 12-32
HUnlockAll 12-15, 12-33

icon 17-3, 17-11
note 6-74
stop 6-86
icon font 4-65, 4-75, 8-16, 25-15, 25-109
iconimage 6-90
iconitem 6-10, 6-88
icon record, QuickDraw II Auxiliary 17-3-4, 17-11
iconRect 6-90
iCopies 15-14
iCurBand 15-41, 15-48
iCurCopy 15-41, 15-48
iCurPage 15-41, 15-48
idErr 12-17, 12-23, 12-31, 12-33, 12-36, 12-39, 12-41, 12-45, 12-47, B-2
iDev 15-11, 15-47
ID number 13-14, 13-16, 13-79
idTagNotAvlErr 14-57, 14-60, 14-70, B-2
iFileVol 15-14, 15-15, 15-48
iFstPage 15-14, 15-21, 15-48
iHRes 15-11, 15-47
iLstPage 15-14, 15-21, 15-48
image pointer 16-13
image width 16-44
ImageWriter
job dialog box 15-8-9
style dialog box 15-6-7
imBadInptParam 9-34, 9-38, 9-39, 9-42, B-4
IMBootInit 9-5
imIllegalChar 9-8, 9-9, 9-25, 9-26, 9-42, B-4
imOverflow 9-8, 9-9, 9-25, 9-26, 9-42, B-4
IMReset 9-7
IMShutDown 9-4, 9-6
IMStartUp 9-4, 9-5
IMStatus 9-7
imStrOverflow 9-29, 9-30, 9-31, 9-33, 9-42, B-4
IMVersion 9-6
inactive control 4-7, 4-8
inactiveCtl 4-7
inactiveHilite 4-67, 4-86
inactive window 25-8, 25-11, 25-24, 25-92
inButton 6-89
incBsyFlag 14-68
inCheckBox 6-89
#include statement 2-6
indicator 4-4, 4-8, 4-83
inDownArrow 6-89
inEditLine 6-89
infinity 9-40
InflateTextBuffer 16-61, 16-157
infoColor 25-20, 25-142
inGrow 6-89
inIconItem 6-89
InitColorTable 16-158-159
inRectl 4-25, 4-29, 4-86
InitCursor 16-160
InitMouseDown 14-32
InitPalette 13-56
InitPort 16-161
InitTextDev 23-28
InitTextDev 23-6
inLongStatText 6-89
inLongStatText2 6-89
inPageDown 6-89
inPageUp 6-89
input 23-46
input parameter 2-6
inRadioButton 6-89
insertion point 6-78, 10-8, 10-10, 10-11
InsertMenu 13-8, 13-13, 13-57
InsertMitem 13-13, 13-58
InsetRect 16-162
InsetRgn 16-163
InstallCDA 5-18
InstallFont 8-14, 8-16, 8-43-44, 10-30
InstallNDA 5-19
inStatText 6-89
integer xxx, 9-1
Integer Math string 9-1
Integer Math string routines 1-8, 9-4
Index

I-12

Integer Math Tool Set 1-4, 1-7-8, 9-1-42
constants 9-42
error codes 9-42
shutdown routine 9-6
startup routine 9-5
status routine 9-7
using 9-4
version number routine 9-6

integer value 9-1
interface file 2-5, 2-7
interface library 2-6
interrupt enable states 1-42
interrupt handler 14-5, 19-1, 19-3, 21-27
one-second 14-27-28
quarter-second 14-28-29
interrupt mode 16-34
interrupt source 14-5
reference numbers 14-26

inThumb 6-89
intrPtMgr 14-67
IntSource 14-5, 14-25-29
Int2Dec 9-29
Int2Hex 9-27, 9-30
inUpArrow 6-89
inUserItem 6-89
invalidGenItem 6-89
invalidGenNumErr 21-13, 21-15, 21-37, B-5

J
jcbButton 7-51
jcevent 7-51
jcGetMouse 7-51
jcTickCount 7-51
job dialog box 15-1, 15-12, 15-20
imageWriter 15-8-9
job subrecord 15-9, 15-14-15, 15-37
journal codes 7-20
journaling mechanism 7-19-20

K
kbdInt 14-24, 14-66
kerning 16-27, 16-29, 16-45-46, 16-58
kernMax 16-47, 16-50
keyboard 3-22
keyboard equivalent 7-13, 13-7, 13-61-62
keyboard interrupts 14-27
keyboard interrupt vector 7-15
keyboard microcontroller 3-17, 3-18
key code 3-22
keyCode 3-20, 3-28
key-down event 6-39, 6-66, 7-3, 7-13
keyDownEvt 7-7, 25-120
keyDownMask 7-11, 7-50
keyPad 7-9, 7-51
KillControls 4-9, 4-68, 11-11, 11-16
KillPicture 17-5, 17-15
KillPoly 16-40, 16-170
kybdBuffer 14-65
kybdDisable 14-26, 14-67
kybdEnable 14-26, 14-67
kybdIntHnd 14-68
kybdLang 14-65
kybdMicHnd 14-68
kybdRepDel 14-65
kybdRepSpd 14-65

L
landscape mode 15-6
langCount 14-65
lang1 14-65
lang2 14-65
lang3 14-65
lang4 14-65
lang5 14-65
lang6 14-65
lang7 14-65
lang8 14-65
language card 12-3
language specification xxvii
lastBlock 21-37
layoutCount 14-65
layout1 14-65
layout2 14-65
layout3 14-65
layout4 14-66
layout5 14-66
layout6 14-66
layout7 14-66

italic type style 16-258, 17-1, 17-3
item character 13-13
itemColor 6-9, 6-18, 6-90
itemDescr 6-9, 6-90
item descriptor 6-12-16
itemDisable 6-11, 6-53, 6-82
itemFlag 6-9, 6-18, 6-90
item ID 6-18, 6-55, 6-61, 13-14
itemID 6-9, 6-90
ItemID2 FamNum 8-36, 8-45
item list 13-17
itemNotFound 6-40, 6-46, 6-50, 6-53, 6-54, 6-56, 6-62, 6-76, 6-81, 6-82, 6-83, 6-84, 6-85, 6-90, B-5
item number 13-22
itemRect 6-9, 6-90
item template 6-8-9, 6-19, 6-58
item type 6-10-11, 6-53
itemType 6-9, 6-90
item value 6-12-16, 6-54
itemValue 6-9, 6-90
iTotBands 15-41, 15-48
iTotCopies 15-41, 15-48
iTotPages 15-41, 15-48
iVRes 15-11, 15-47
INDEX

M
MacGen 2-5
Macintosh font record 16-41
macro file 2-5
mainID 12-10, 14-58
MapPoly 16-174
MapPt 16-175
MapRect 16-176
MapRgn 16-177
marked menu item 13-80
mask drawing 16-18
maskHandle 16-191, 16-277
maskSize 16-274
master color 16-31, 16-274-275
master color value 16-31, 16-35
master pointer 12-5
master scan line control byte. See master SCB
master SCB 2-2, 16-127, 16-160, 16-231
master user ID 12-10-11, 12-14, 12-15, 12-23, 12-35
masterIRQNotAssignErr 21-37
math routine 9-4
MaxBlock 12-26, 12-34, 15-20, 15-22, 15-30
maxFbrExtent 16-59-60
maxFixed 9-42
maxFontHeight 16-59-60
maxInt 9-42
maxItemType 6-88
maxLongInt 9-42
maxULong 9-42
maxWidth 16-59-60, 16-61, 16-206, 16-276
mChooseMsg 13-22, 13-24, 13-87
MCOPY assembler directive 2-5
mCustom 13-87
mDisabled 13-87
mDownMask 7-11, 7-50
mDrwOnItem 13-22, 13-27, 13-87
mDrwOnMsg 13-22, 13-23, 13-87
mDrwOnTitle 13-22, 13-28, 13-87
mechanical specification xxvi
member record 11-6
  disabled 11-6
  drawing 11-17
  selected 11-6, 11-12, 11-20, 11-21, 11-22
  sorting 11-25
memBit 8-9, 8-10, 8-50
memDisabled 11-6, 11-25
memErr 12-19, 12-36, 12-41, 12-42, 12-43, 12-47, B-2
memFlag 11-6, 11-20, 11-21, 11-26
memOnlyBit 8-11, 8-50
memory
  allocating private 12-11
  allocation 12-35
  attributes 12-12, 12-37
  compaction 12-5-7, 12-15, 12-22
  deallocation 12-23, 12-25
  expansion 12-3
  fragmentation 12-6-8
  limits 12-3
  locking 12-30, 12-31
  purging 12-39, 12-40, 12-44, 12-45
  reallocation 12-41, 12-42
  unlocking 12-32, 12-33
memory attributes word 12-12-13, 12-37
memory block 12-1, 12-5
  attributes 12-12-13, 12-37
  locking 12-30, 12-31
  purging 12-8-10, 12-13, 12-39, 12-40, 12-44, 12-45
  unlocking 12-32, 12-33
memory handle 12-5
Memory Manager 1-3, 1-9, 2-1, 2-2, 12-1-47
  constants 12-47
  error codes 12-47
  shutdown routine 12-18
  startup routine 12-17
  status routine 12-20
  version number routine 12-19
memory space 12-1
memPir 11-6, 11-26
memSelect 11-6, 11-20, 11-21
memSelected 11-6, 11-25
menu 13-1
  pull-down 1-3, 13-1
menu bar 13-4-5. See also system menu bar
  window 13-5
menu bar color 13-18, 13-42-43, 13-69-70
menu bar record 13-17-18
MenuBoxInit 13-29
menu color 13-17
menu definition procedure 13-21-26
menuFlag 13-19, 13-20, 13-21, 13-23, 13-88
mSelected 13-87
msgPtrVctr 14-68
mSizeMsg 13-22, 13-25, 13-87
mstrIRQNotAssignErr  B-3
MTBootlnit 14-6
MTReset 14-8
MTShutDown 14-4, 14-7
MTStartUp 2-1, 14-4, 14-6
MTStatus 14-8
MTVersion 14-7
Multiply 9-37
Munger 14-5, 14-45-47
mUpMask 7-11, 7-50
mvEscape 24-26
mvReturn 24-26
mXor 13-87
m
NaN 9-40, 9-41
native mode 1-2, 1-4
NDA header section 5-19
newBarColor 13-70
NewControl 4-9, 4-24, 4-70-73, 11-11, 25-68
new desk accessory 5-1
action codes 5-7
total number installed 5-17
NewDItem 6-23, 6-68-69
NewHandle 12-15, 12-35-37, 15-19
newInvertColor 13-70
newItemFailed 6-57, 6-69, 6-90, B-5
NewList 11-19
NewMenu 13-8, 13-16, 13-21, 13-67
NewMenuBar 13-68
NewModalDialog 6-7, 6-23, 6-70-71
NewModelessDialog 6-7, 6-23, 6-72-73
newOutColor 13-70
NewRgn 16-40, 16-74, 16-113, 16-153, 16-181
newValue 4-25, 4-37, 4-86
NewWindow parameter list 25-142
newYork 8-4, 8-51
NextMember 11-20
nextWavePtr 21-17, 21-37
noConstraint 4-53, 4-86
noDevice 23-15, 23-47, B-4
noDevParamErr 14-32, 14-70, B-2
noDisplay 22-22, 22-32
noDOCFindErr 21-8, 21-24, 21-28, 21-37, B-3
noEcho 23-46
noFile 23-15, 23-47, B-4
noHilite 4-67, 4-86
nonreentrant code 19-1
nonspecial memory 12-3
newlnvertColor 13-70
newItemFailed 6-57, 6-69, 6-90, B-5
new normal memory 12-3
noRoom 23-15, 23-47, B-4
noSAppInitErr 21-15, 21-37, B-3
noSelect 22-22, 22-32
noSigTaskErr 14-48, 14-70, B-2
notBaseBit 8-12, 8-50
notBC 16-19, 16-20, 16-235, 16-275
notClosed 23-15, 23-47, B-4
notCopy 16-19, 16-20, 16-235, 16-275
notDiskBit 8-9, 8-50
note alert 6-6
NoteAlert 6-24, 6-74
noteIcon 6-74
notEmptyErr 12-41, 12-42, 12-47, B-2
NotEmptyRect 16-182
notEqualChunkiness 16-190, 16-197, 16-278, B-3
Note Sequencer 1-4, 21-1
noteSynth 21-37
Note Synthesizer 1-4, 21-1
noteSynthMode 21-36
notForeBIC 16-30, 16-260, 16-275
notForeCOPY 16-30, 16-260, 16-275
notForeOR 16-30, 16-260, 16-275
notForeXOR 16-30, 16-260, 16-275
notFoundBit 8-12, 8-50
notInitialized 16-278, B-3
notModalDialog 6-65, 6-67, 6-90, B-5
notOpen 23-16, 23-47, B-4
notOR 16-19, 16-20, 16-235, 16-275
notSysWindow 5-14, 5-30, B-3
notXOR 16-19, 16-20, 16-235, 16-275
notUnderItem 13-78, 13-87
notRect 7-7, 7-50, 25-119
number of copies 15-8
numeric spacing 16-226
numOfltems 13-19, 13-20, 13-88

O
object module format xxvii
ObscureCursor 16-182
offsetToFMT 16-43, 16-276
offset point 4-50
OffsetPoly 16-183
OffsetRect 16-184
OffsetRgn 16-185
offset/width table 16-44, 16-50-51
ok 6-89
OK button 6-4, 6-5, 6-18, 6-58
okDefault 6-89
oneSecHnd 14-68
oneSecInt 14-24, 14-66
one-second interrupt handler 14-27-28
Open File dialog box 22-3, 22-4, 22-21, 22-25-26
OpenNDA 5-6, 5-20, 25-122
OpenPicture 15-38, 17-5, 17-9, 17-14
OpenPoly 16-40, 16-186
OpenPort 16-39, 16-161, 16-187, 25-42
OpenRgn 16-40, 16-74, 16-187
operating system xxviii
optKey 7-9, 7-51
Option-Left Arrow 10-1, 10-29
Option-Right Arrow 10-1, 10-29
orange320 16-275
origin mask 25-104
oscillator register 21-5
oscillator table 21-22, 21-23
oscillator-to-generator translation table 21-5
 oscTable 21-36
 oSecDisable 14-24, 14-67
 oSecEnable 14-24, 14-67
 OSEventAvail 7-42
 oVector 14-68
 outlineMask 16-276
 outline type style 16-258, 17-1, 17-3
 outOfMemErr 14-56, 14-70, B-2
 output 23-46
 output parameter 2-6
 output sample rate 21-17
 oval 16-23, 16-89, 16-95, 16-102, 16-105, 16-189

P
package1Err 14-55, 14-70, B-1
package2Err 14-55, 14-70, B-1
package3Err 14-55, 14-70, B-1
package4Err 14-55, 14-70, B-1
package5Err 14-55, 14-70, B-1
package6Err 14-55, 14-70, B-1
package7Err 14-55, 14-70, B-1
package8Err 14-55, 14-70, B-1
package9Err 14-55, 14-70, B-1
package10Err 14-56, 14-70, B-2
package11Err 14-56, 14-70, B-2
package12Err 14-56, 14-70, B-2
PackBytes 14-5, 14-39-41, 14-42
page 4-5
page-aligned memory block 2-2, 12-12, 12-13
pageDown 4-86
page range 15-8, 15-38
page rectangle 15-11
page region 25-62, 25-105
page setup 15-1
pageUp 4-86
paging region 4-5, 4-8
PaintArc 16-188
painting 16-20
PaintOval 16-189
PaintPixels 16-190-191
PaintPixels parameter block 16-191, 16-277
PaintPoly 16-192
PaintRect 16-193
PaintRgn 16-194
PaintRect 16-195
palette standard in 320 mode 16-35
standard in 640 mode 16-36
papConnNotOpen 15-36, 15-42, 15-49, B-5
paper source 15-8
paperType 15-12, 15-13, 15-48
papReadWriteErr 15-36, 15-42, 15-49, B-5
parameter Battery RAM reference numbers 14-12
definition xxix
DragRect 4-50-53
GetAddr reference numbers 14-20
input 2-6
length xxx
NewControl 4-71-73
output 2-6
passing 2-6
pseudo-type xxx
paramLenErr 25-83, 25-144, B-4
paramLength 25-84, 25-142
param1 6-90
param2 6-90
ParamText 6-24, 6-75
part code 4-8, 4-57, 4-67, 4-81, 4-82, 4-83, 4-84, 6-67
Pascal device driver 23-3
Pascal string 5-3, 16-26
pascalType 23-46
paste 5-30
pasteAction 5-7, 5-30
Paste command 6-44, 20-1, 20-6
patSize 16-274
pattern 16-265
pdoSblk0Err 14-55, 14-70, B-1
pdoSFCBErr 14-55, 14-70, B-1
pdoSIntShdErr 14-55, 14-70, B-1
pdoSUnClmdIntErr 14-55, 14-70, B-1
pdoSVCBErr 14-55, 14-70, B-1
pen displacement 16-29, 16-56, 16-70, 16-77, 16-267, 16-270
pen level 16-156, 16-265
pen location 16-17, 16-18, 16-22, 16-26, 16-40, 16-80, 16-81, 16-82, 16-83, 16-128, 16-178, 16-180, 16-196
pen mask 16-129, 16-233
pen mode 16-19-20, 16-22, 16-130, 16-234-235, 16-275
PenNormal 16-196
pen pattern 16-18-19, 16-131, 16-236, 16-252
pen size 16-18, 16-22, 16-132, 16-237
pen state 16-133, 16-196, 16-238
pen state record 16-238
PenState record 16-278
pFileNam e 15-14, 15-15, 15-48
PicComment 17-5, 17-15
picItem 6-10, 6-12, 6-88
picSave 16-134, 16-239, 16-277, 17-14
pinning 9-3
PinRect 25-89-90
pixel 16-10-11, 16-135, 16-197, 16-200, 16-201
background 16-28, 16-30
chunky 16-31
foreground 16-27, 16-30
region 16-190
pixel image 2-2, 16-9, 16-12-14, 16-48, 17-10
place-holding character 5-8
plane 25-8
PMBootlnit 15-25
PMReset 15-28
PMSHutdown 15-19, 15-27
PMStartUp 15-19, 15-26
PMStatus 15-28
PMVersion 15-27
pnStateSize 16-274

Index I-17
point xxx, 8-4, 16-11-12, 16-21-22, 16-40, 16-68, 16-85, 16-135, 16-154, 16-175, 16-200, 16-201, 16-207, 16-268, 25-89

POINTER xxx
pointing device 7-21-25, 7-27, 7-34, 14-5
polling, Apple Desktop Bus 3-3
polygon 16-24-25, 16-40, 16-72, 16-90, 16-96, 16-103, 16-166, 16-170, 16-174, 16-183, 16-186, 16-192
polyNotOpen 16-72, 16-278, B-3
polySave 16-136, 16-240, 16-277
polyTooBig 16-278, B-3
plAddLine 14-64
plBaud 14-64
plBuffer 14-64
plDCDHndShk 14-64
plDelimiter 14-64
plDSRHndShk 14-64
plDtStpBits 14-64
plEcho 14-64
plLineLength 14-64
plParity 14-64
plPrintModem 14-64
plXnfhndShk 14-64
port. See GrafPort
port driver 15-24, 15-43
portrait mode 15-6
portRect field 16-14, 16-15
portSCB 16-13, 16-277
portSize 16-274
posCtl 4-25, 4-86
PosMouse 14-33
PostEvent 7-14, 7-43-44
pPrPort 15-41, 15-48
PPToPort 16-197-198
prAbort 15-20, 15-22, 15-34, 15-44, 15-47
PrChoosePrinter 15-20, 15-22, 15-29
PrClosePage 15-20, 15-21, 15-31, 15-38
PrDefault 15-19, 15-22, 15-46
PrDriverVer 15-33
PrError 15-22, 15-30, 15-34
prestylized fonts 8-5-6, 16-43
prInfo 15-10, 15-11, 15-47
Print 15-1
printer driver 15-20, 15-21, 15-22, 15-44
printer effects choice 15-6
printer error code 15-34, 15-44, 15-49
printer information subrecord 15-11
printer names dialog box 15-5
printer paper 15-5
printer status record 15-41
printing color 15-15-18
draft 15-14-15, 15-23, 15-37
spool 15-14-15, 15-23, 15-37, 15-40
printing loop 15-20, 15-34
Print Manager 1-4, 1-11, 15-1-50
constants 15-47
data structures 15-47-48
error codes 15-49
shutdown routine 15-27
startup routine 15-26
status routine 15-28
using 15-19-22
version number routine 15-27
private memory 12-11, 12-14
private scrap 20-5-6
prJob 15-10, 15-47
ProDOS 5-4
ProDOS 16 xxviii, 5-7, 5-18, 5-19, 12-1, 12-14
proDOSvcr 14-68
proportionally spaced font 16-226
proportional scroll bars 25-10
PrPixelMap 15-42
PrPortVer 15-43
PrSetError 15-21, 15-22, 15-22, 15-44
prStil 15-10, 15-12, 15-47
PrStdDialog 15-20, 15-21, 15-22, 15-45, 15-46
psuedorandom numbers 16-202
pseudo-type xxx
prVersion 15-10, 15-47
pInstDiskErr 14-56, 14-70, B-2
psMask 16-238, 16-278
psMode 16-238, 16-278
psPat 16-238, 16-278
psSize 16-238, 16-278
PrInRect 16-200
PrInRgn 16-201
prtToDestLocInfo 16-191, 16-277
prtToDestPoint 16-191, 16-277
PrToHand 12-38
prtToPixmap 16-13, 16-277
prtToSourceLocInfo 16-191, 16-277
prtToSourceRect 16-191, 16-277
Pt2Rect 16-199
p2AddLine 14-64
p2Baud 14-64
p2Buffer 14-64
p2DCDHndShk 14-64
p2Delimiter 14-64
p2DSRHndShk 14-64
p2DtStpBits 14-64
p2Echo 14-64
p2LineLength 14-64
p2Parity 14-64
p2PrintModem 14-64
p2XnfhndShk 14-64
pull-down menu 1-3, 13-1
purgeable block 12-39, 12-40
PurgeAll 12-11, 12-39
purgeBit 8-9, 8-10, 8-50
purge block 12-8
purgeErr 12-39, 12-40, 12-47, B-2
PurgeHandle 12-40
purge level 12-9, 12-15, 12-44, 12-45, 24-25
purple32 16-274
PutScrap 20-5, 20-6, 20-11, 20-16, 20-18

Q
QDAuxBootInit 17-6
QDAuxReset 17-8
QDAuxShutDown 17-5, 17-7
QDAuxStartUp 17-5, 17-6
QDAuxStatus 17-8
QDAuxVersion 17-7
QDBoolInit 16-63
QDReset 16-67
QDSHutDown 16-39, 16-66
QDStartUp 2-2, 16-39, 16-59, 16-61, 16-64-65, 16-275
QDStatus 16-67
QDVersion 16-66
qSecDisable 14-26, 14-67
qSecEnable 14-26, 14-67
qSecIntHnd 14-68
quarter-second interrupt handler 14-28-29
quartSecInt 14-24, 14-66
queueDmgdErr 14-48, 14-52, 14-70, B-2
QuickDraw II 1-3, 1-11-14, 2-2, 16-1-278
constants 16-274-276
data structures 16-276-278
error codes 16-278
shutdown routine 16-66
startup routine 16-64-65
status routine 16-67
using 16-39-40
version number routine 16-66
QuickDraw II Auxiliary 1-4, 1-14, 2-4, 8-5, 8-14, 10-44, 13-83, 16-258, 17-1-16, C-6
shutdwn routine 17-7
startup routine 17-6
status routine 17-8
using 17-5
version number routine 17-7

R
radio button 4-4, 4-10, 4-18
control record 4-18-20
radioButton 4-86
radioItem 6-10, 6-88
radioProc 4-13, 4-19, 4-73, 4-85
radNor 4-20, 4-88
radReserved 4-20, 4-88
radTitle 4-20, 4-88
ramBased 23-46
RAM-based device driver 23-1, 23-4
RAM-based tool set 24-3
RAM tool set 1-1, 2-3
Random 16-202, 16-246
range mode 11-12
rcADBAaddr 3-29
rcLayoutOrLang 3-29
rcRepeatDelay 3-29
rdMaxRam 14-65
rdMinRam 14-65
ReadAbs 3-16
readADBError 3-17, 3-28
ReadASCIITime 14-4, 14-16
readAvailCharSet 3-17, 3-28
readAvailLayout 3-17, 3-28
ReadBParam 14-4, 14-13
ReadBRam 14-4, 14-10
ReadChar 23-7, 23-29, 23-30, 23-40
readConfig 3-17, 3-28
ReadConfigRec 3-29
ReadKeyMicroData 3-17
ReadKeyMicroMemory 3-18
ReadLine 23-30-31
readMicroMem 3-28
readModes 3-17, 3-28
ReadMouse 14-34
read next 21-23, 21-33
readNext 21-23, 21-36
Read RAM 21-23, 21-31
readRAM 21-23, 21-36
ReadRamBlock 21-24
Read Register 21-23, 21-29
readRegister 21-23, 21-36
ReadTimeHex 14-4, 14-14
readVersionNum 3-17, 3-28
real font 8-7
reallocation of memory 12-43, 12-44
ReAllocHandle 12-15, 12-41
readOnlyBit 8-11, 8-50
reCt 4-25, 4-39, 4-40
recSize 4-25, 4-86
RECT xxx, 16-22
RectlnRgn 16-203
RectRgn 16-204
red320 16-275
red460 16-274
redMask 16-274
redraw routine 13-64-65
reduction 15-12, 15-13, 15-48
reentrant code 18-10, 19-1, 19-3
RefreshDesktop 25-91, 25-135
register 2-7
RemoveDItem 6-24, 6-76
repeat delay 7-3
repeat speed 7-3
reply record 22-24, 22-31, 22-32
reserved memory 12-3
resetADB 3-21, 3-28
ResetAlertStage 6-76
resetKbd 3-20, 3-28
ResetMember 11-21
resetSys 3-20, 3-28
resolution. See screen resolution
RestAll 5-21
RestoreBufDims 16-58, 16-61, 16-205
RestoreHandle 12-15, 12-42

Index 1-19
scrollProc 4-13, 4-21, 4-73, 4-85
ScrollRect 16-39, 16-208
scrollThumb 6-15, 6-88
SDivide 9-38
SEctRect 16-209
SEctRgn 16-210
segLoader1Err 14-55, 14-70, B-1
segLoader2Err 14-56, 14-70, B-2
select mode 11-12
selection range 6-78, 10-4, 10-7-8, 10-9, 10-11
SelectlText 6-77-78
SelectMember 11-22
selectOnlyOne 11-25
SetIText 6-24
SendBehind 25-93
SendInfo 3-4, 3-5, 3-6, 3-19-22, 7-24, 7-25
Serial Communications Controller (SCC) interrupt flag 14-21
ServeMouse 14-34, 14-35
SetAbsClamp 7-27, 14-5, 14-37
SetAbsScale 3-15, 3-23
SetAllSCBs 16-211
SetArcRot 16-212
SetBackColor 16-55, 16-213
SetBackPat 16-214
SetBarColors 13-41, 13-67, 13-69-70
SetBufDims 16-58-59, 16-215-216
SetCharExtra 16-51, 16-55, 16-217
SetClip 6-7, 16-40, 16-218
SetClipHandle 16-219
SetColorEntry 16-220
SetColorTable 16-221
setConfig 3-20, 3-28
setConfigRec 3-29
setContentDraw 25-94
setContentOrigin 25-95
setContentOrigin2 25-96
SetCtlAction 4-74, 11-11
SetCtlIcons 4-11, 4-75
SetCtlParams 4-6, 4-76, 11-11
SetCtlRefCon 4-77
SetCtlTitle 4-78, 11-11
SetCtlValue 4-10, 4-79, 4-84, 11-11
SetCursor 16-222
SetDAFont 6-24, 6-79
SetDataPtr 5-23-25
SetDataSize 25-97
SetDefButton 6-80
SetDefProc 25-98
SetDesktop 25-41, 25-139
SetDItemBox 6-81
SetDItemType 6-82
SetDItemValue 6-12, 6-83
SetEmptyRgn 16-223
SetErrGlobals 23-5, 23-32
SetErrorDevice 23-6, 23-33
SetEventMask 7-45
SetFont 8-16, 16-224
SetFontFlags 16-56, 16-225-226
SetFontID 8-16, 16-227
SetForeColor 16-55, 16-228
SetFrameColor 25-20, 25-99-100
SetGrafProcs 16-229
SetHandleSize 12-42
SetHeartBeat 7-23, 14-48-51
SetInfoDraw 25-101
SetInfoRefCon 25-102
SetInGlobals 23-5, 23-34
SetInputDevice 23-5, 23-35-36
SetIntUse 16-230
SetText 6-23, 6-84
SetMasterSCB 16-231
SetMaxGrow 25-103
SetMenuBar 13-71
SetMenuFlag 13-72
SetMenuID 13-16, 13-73
SetMenuTitle 13-13, 13-74
SetMItem 13-13, 13-75
SetMItemBlink 13-13, 13-76
SetMItemFlag 13-15, 13-77-78
SetMItemID 13-16, 13-79
SetMItemMark 13-6, 13-80
SetMItemName 13-81
SetMItemStyle 13-6
setModes 3-20, 3-28
SetMouse 14-34, 14-36
SetMTileStart 13-82
SetMTileWidth 13-83
SetOriginMask 25-104
SetOutGlobals 23-5, 23-37
SetOutputDevice 23-6, 23-38
SetPage 25-105
setParams 4-25, 4-38, 4-86
SetPenMask 16-233
SetPenMode 16-40, 16-234-235, 16-259
SetPenPat 16-40, 16-236
SetPenSize 16-40, 16-237
SetPenState 16-238
SetPicSave 16-239
SetPolySave 16-240
SetPort 16-39, 16-241, 25-116
SetPortLoc 16-242
SetPortRect 16-243
SetPortSize 16-244
SetPt 16-245
SetPurge 12-15, 12-44
SetPurgeAll 12-15, 12-45
SetPurgeStat 8-15, 8-49
SetRandSeed 16-202, 16-246
SetRect 16-247
SetRectRgn 16-248
SetRgnSave 16-249
SetRecPath 20-17
SetScroll 25-106
SetSolidBackPat 16-251
SetSolidPenPat 16-252
SetSoundMIRQV 21-25
SetSoundVolume 21-26
SetSpaceExtra 16-51, 16-55, 16-253
SetStdProcs 16-254
SetUserField 16-262
SetVector 7-23, 14-5, 14-26, 14-53, 14-61-62, 14-63
SetVisHandle 16-263

Index 1-21
Standard Apple Numeric Environment. See SANE
standard color palette
320 mode 16-35
640 mode 16-36
standard color table 16-159
standard control type value 4-73
Standard File Operations Tool
Set 1-4, 1-15, 22-1-32
constants 22-32
data structures 22-32
shutdown routine 22-17
startup routine 22-16
status routine 21-19
using 22-13-14
version number routine 21-18
standard filter procedure 6-25
standard GrafPort 16-187
standardMenu 13-72, 13-87
standard menu edit 5-27
standard pen state 16-196
standard SCB 16-145
standard window control 25-6-7
StartDrawing 10-11, 25-31, 25-116
StartInfoDrawing 25-47, 25-117
starting up a tool set 2-4
startupAlreadyMade 15-49, B-5
startup order, tool set C-6
startup routines
Apple Desktop Bus Tool Set
3-10
Control Manager 4-44
Desk Manager 5-11
Dialog Manager 6-30
Event Manager 7-30
Font Manager 8-22
Integer Math Tool Set 9-7
LineEdit Tool Set 10-15
List Manager 11-15
Memory Manager 12-20
Menu Manager 13-32
Miscellaneous Tool Set 14-8
Print Manager 15-28
QuickDraw II 16-67
QuickDraw II Auxiliary 17-8
SANE Tool Set 18-14
Scheduler 19-6
Scrap Manager 20-9
Sound Tool Set 21-10
Standard File Operations Tool
Set 21-19
Text Tool Set 23-14
Tool Locator 24-6
Window Manager 25-34
TextToolDev 16-146, 16-255, 16-277
system configuration 14-11
SYSTEM directory 2-3, 5-3, 5-6, 8-1, 8-15, 15-23, 15-24
system direct page 5-4
System Edit 5-6, 5-26, 25-123
SystemClick 5-6, 5-27, 25-123
system configuration 14-11
SYSTEM directory 2-3, 5-3, 5-6, 8-1, 8-15, 15-23, 15-24
system direct page 5-4
System Edit 5-6, 5-27, 25-123
System Failure Manager 14-54
system failure message 14-5, 14-54
system font 6-79, 8-8, 8-39, 8-40, 8-48, 13-78, 16-147, 16-256
System Loader 12-1, 12-14, 12-44, 12-45
Index 1-23
system speaker 14-5
System Task 5-6, 5-8, 5-29, 25-119
system tool A-1
system volume 21-26
system window 5-26, 25-8, 25-66, 25-70, 25-107
sysTool 24-26

text editing 6-11
text face 16-148, 16-257
text face flag 16-258
text file format xxvii
text mode 16-30, 16-149, 16-250-260, 16-275
TextReadBlock 23-40
TextReset 23-13
textScrash 20-4, 20-19
text screen 24-16, 24-17, 24-23
TextShutDown 23-9, 23-11
TextSetup 23-5, 23-9, 23-11
TextStatus 23-14
text style 16-276
menu item 13-52
Text Tool Set 1-4, 1-16, 23-1-47
constants 23-46
error codes 23-47
shutdown routine 23-11
startup routine 23-11
status routine 23-14
using 23-9
version number routine 23-12
TextVersion 23-12
TextWidth 16-29, 16-58, 16-270
TextWriteBlock 23-41
thumb 4-8, 25-9
thumb box 4-5
thumbCtl 4-25, 4-32-34, 4-86
tickCnt 14-66
tickCount 7-14, 7-48
tick counter 14-22
time
ASCII 14-16
hexadecimal 14-14, 14-15
times 8-4, 8-51
title, window 25-73, 25-111, 25-115
title bar 25-6, 25-8, 25-49
title character 13-13
titleColor 25-18, 25-142
titleName 13-19, 13-20, 13-21, 13-88
titleParm 6-90
titleWidth 13-19, 13-20, 13-21, 13-88
TLBootInit 24-4
TLMountVolume 24-3, 24-21-22
TLReset 24-6
TLShutDown 24-3, 24-5
TLStartUp 2-1, 24-3, 24-4
TILStatus 24-6
TLTextMountVolume 24-3, 24-23-24
TLVersion 24-5
tmClose 25-14, 25-125, 25-140
tmContent 25-14, 25-124, 25-140
tmRedraw 25-14, 25-120, 25-140
tmDrags 25-14, 25-124, 25-140
tmFindW 25-14, 25-121, 25-140
tmGrow 25-14, 25-125, 25-140
tmInactive 13-60, 25-14, 25-121, 25-140
tmInfo 25-14, 25-126, 25-140
tmMenuKey 25-14, 25-119, 25-140
tmMenuSel 25-14, 25-121, 25-140
tmOpenNDA 25-14, 25-122, 25-140
tmScroll 25-14, 25-126, 25-140
tmSpecial 25-14, 25-122, 25-140
tmSysClick 25-14, 25-123, 25-140
tmUpdate 25-14, 25-119, 25-140
tmZoom 25-14, 25-125, 25-140
tobottom 25-139
ToDesk 25-40, 25-139
toolErr 2-6
toolToolBar 14-67
toolLoc1 14-67
toolLoc2 14-67
Tool Locator 1-2, 1-3, 1-16, 2-1, 2-3, 24-1-26, A-2
constants 24-26
error codes 24-26
shutdown routine 24-5
startup routine 24-4
status routine 24-6
using 24-3
version number routine 24-5
toolNotFoundErr 24-7, 24-8, 24-9, 24-10, 24-11, 24-19, 24-20, 24-25, 24-26, B-2
tool pointer table (TPT) 24-19, A-2
tool set 1-1-19
tools 2-5, B-1-5
installing A-6-9
loading 2-3-4
minimum version number
24-10, 24-11
RAM-based 24-3
shutting down 2-4
WriteBRam 14-4, 14-9
WriteChar 23-42
WriteCString 23-7, 23-43
WriteLine 23-44
writeMicroMem 3-20, 3-28
write next 21-23, 21-34
writeNext 21-23, 21-36
writeProtected 23-16, 23-47, B-4
Write RAM 21-23, 21-32
writeRAM 21-23, 21-36
WriteRamBlock 21-28
Write Register 21-23, 21-30
writeRegister 21-23, 21-36
WriteString 23-45
WriteTimeHex 14-4, 14-15
wScrollHor 25-84, 25-87, 25-143
wScrollVer 25-84, 25-87, 25-143
wStorage 25-84, 25-88, 25-143
wStructRgn 25-16, 25-142
wTitle 25-84, 25-85, 25-142
wTrackZoom 25-125
wUpdateRgn 25-16, 25-142
wVAxisOnly 25-139
wXOrigin 25-84, 25-87, 25-142
wYOrigin 25-84, 25-87, 25-142
wZoom 25-84, 25-87

X

\texttt{xDivide} 3-24, 3-29
\texttt{xMultiply} 3-24, 3-29
\texttt{xOffset} 3-24, 3-29
\texttt{xorMItemHilite} 13-78, 13-87
\texttt{XorRgn} 16-273
\texttt{xorTitleHilite} 13-72
\texttt{X2Fix} 9-40
\texttt{X2Frac} 9-41

Y

\texttt{yDivide} 3-24, 3-29
\texttt{yMultiply} 3-24, 3-29
\texttt{yOffset} 3-24, 3-29

Z

\texttt{ZeroScrap} 20-5, 20-6, 20-11, 20-16, 20-18
\texttt{zeroSize} 16-226, 16-274
\texttt{zoom box} 25-6, 25-10, 25-49
\texttt{zoomed window size} 25-74, 25-112
\texttt{zoom region} 25-9, 25-11, 25-49, 25-129-130
\texttt{ZoomWindow} 25-11, 25-12, 25-138

Index
THE APPLE PUBLISHING SYSTEM

This Apple manual was written, edited, and composed on a desktop publishing system using the Apple Macintosh® computers and Microsoft® Word. Proof and final pages were created on the Apple LaserWriter® Plus. POSTSCRIPT®, the LaserWriter page-description language, was developed by Adobe Systems Incorporated. Some of the illustrations were created using Adobe Illustrator™.

Text type is ITC Garamond® (a downloadable font distributed by Adobe Systems). Display type is ITC Avant Garde Gothic®. Bullets are ITC Zapf Dingbats®. Program listings are set in Apple Courier, a monospaced font.