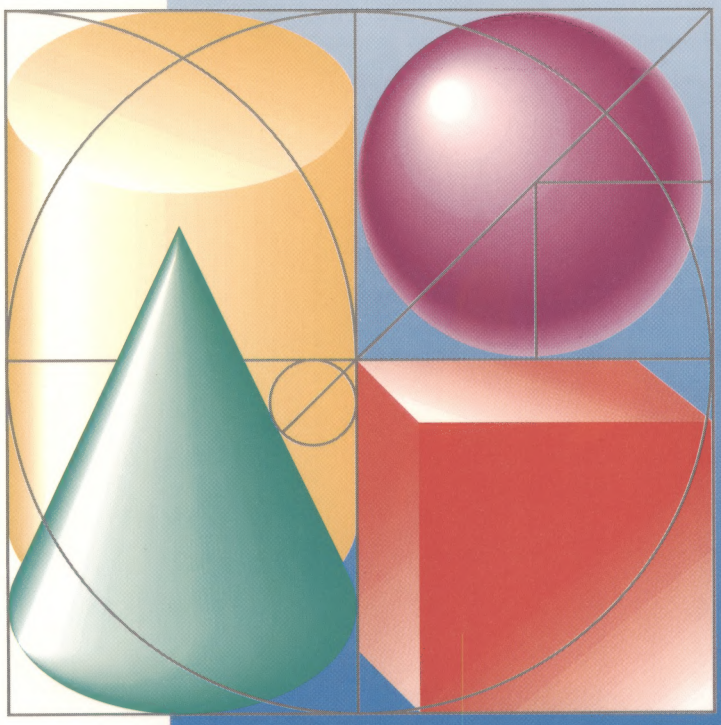
Apple IIGS® C Toolbox Quick Reference

APDA #A0019LL/A



Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010
TLX 171-576

To reorder products, please call:
Apple Programmers and Developers Association
1-800-282-APDA

Apple®

Apple® IIGS® Toolbox Quick Reference

C Language

APPLE COMPUTER, INC.

20525 Mariani Avenue
Cupertino, CA 95014
(408) 996-1010

Simultaneously published in the United States and Canada.

Contents

Preface **About this book**

THE APPLE IIGS QUICK REFERENCES ARE SUCCINCT GUIDES to the tool calls, error codes, and data structures provided by the Apple IIGS Toolbox. They are not intended to be comprehensive explanations. Rather, they are intended for accomplished programmers who may need a quick way to look up a piece of information, such as the correct spelling of a parameter or the order of fields in a data structure.

For more detailed descriptions of all the tool calls, see the *Apple IIGS Toolbox Reference,* Volumes 1 and 2. For a complete description of ProDOS 16 and the System Loader, see the *Apple IIGS ProDOS 16 Reference*.

The quick reference uses a format that is designed to conserve space while remaining readable. The presentation of the information is therefore somewhat unusual.

The tool calls appear in the alphabetical order of the tool sets. Within a tool set, the calls are in the numerical order of the tool call numbers.

A tool call entry consists of seven main elements:

- tool call number
- tool call name
- return type
- descriptive text
- error codes
- input parameters

The tool call number and tool call name are straightforward. The tool call number value is in hexadecimal format. The return type is the storage type of the value returned by the call (if any). If no value is returned, this is indicated by the word "Void." The input types followed by the input parameters are listed to the right of the descriptive text. The somewhat unconventional placement allows a large amount of information to be presented in a small space.

Below the descriptive text, error codes (if any) are listed. If no error code is returned by a call, this element is omitted entirely.

One tool section, ProDOS 16, contains an additional element: the parameter block. Parameter block pointers are all identical, in that they are all longword values; however, they point to many different kinds of parameter blocks. Accordingly, the ProDOS format is slightly different:

- OS call number
- OS call name
- return type
- *pBlockPtr*
- parameter block type
- descriptive text
- parameter block
- error codes

The pBlockPtr element is always literally the same and is always called pBlockPtr. The parameter block type is a descriptive name of the data structure pointed to by pBlockPtr. The actual fields of that data structure are detailed in the parameter block element.

The error codes are listed in numerical order. (Below the hexadecimal number is the name of the error, followed by a very brief description of the error.)

The data structures in listed in alphabetical order of structure name. (Below the name of the structure are the field names, followed by the field types.)

The ancillary calls are listed in alphabetical order. To the right of the call is a very brief description of the call. The ancillary calls should not be made by an application. They are provided for completeness, but in very abbreviated form because no programmer should need to use them.

Two indexes of tool calls conclude the quick reference. The first is arranged by tool set, with the calls in alphabetical order below each tool set title. The second is a straightforward alphabetical listing of all the tool calls.

$021D
ACEStartUp
(Void)

Starts up the Audio Compression and Expansion Tool Set.

Parameters:
Word *dPageAddr*

$031D
ACEShutDown
(Void)

Shuts down the ACE Tool Set. If your application has started up the ACE Tool Set, the application must make this call before it quits.

$041D
ACEVersion
(Word)

Returns the version number of the ACE Tool Set.

$061D
ACEStatus
(Word)

Indicates whether the ACE Tool Set is active.

$071D
ACEInfo
(Long)

Returns a Long value containing the information specified by the *infoItemCode* parameter. At present the only valid input is 0, which causes ACEInfo to return the size, in bytes, of the ACE Direct Page in bank $00.

Parameters:
Word *infoItemCode*

$091D
ACECompress
(Void)

Stores a compressed copy of the specified audio data. *src* is a handle to the buffer containing the source data. The source data is actually located zero or more bytes past the location pointed to by *src*. This offset is the value of *srcOffset*. The buffer is (*nBlks* * 512) bytes long. The destination buffer is similarly designated. *dest* is a handle to the buffer, while *destOffset* is the offset to the actual location of the data. The destination buffer must be at least (*nBlks* * 512) * (5 – *method*) DIV 8) bytes long. If *method* equals 1, the data will be compressed from 8 bits to 4. If it equals 2, then the data will be compressed from 8 bits to 3.

Parameters:
Handle *src*
Long *srcOffset*
Handle *dest*
Long *destOffset*
Word *nBlks*
Word *method*

$0A1D
ACEExpand
(Void)

Stores a decompressed copy of the specified audio data. *src* is a handle to the buffer containing the source data. The source data is actually located zero or more bytes past the location pointed to by *src*. This offset is the value of *srcOffset*. The buffer is ((*nBlks* * 512) * (5 – *method*) DIV 8) bytes long. The destination buffer is similarly designated. *dest* is a handle to the buffer, while *destOffset* is the offset to the actual location of the data. The destination buffer must be at least (*nBlks* * 512) bytes long. *method* code 1 specifies expansion from 4 bits to 8; *method* code 2 specifies expansion from 3 bits to 8.

Parameters:
Handle *src*
Long *srcOffset*
Handle *dest*
Long *destOffset*
Word *nBlks*
Word *method*

$0B1D
ACECompBegin
(Void)

Prepares ACE for data compression. ACE preserves certain state information during any compression process to make it possible to compress large blocks of data by breaking them into subblocks. ACECompBegin should always be called before compressing the first subblock (or only block) of a particular sequence.

$0C1D
ACEExpBegin
(Void)

Prepares ACE for data expansion. ACE preserves certain state information during any expansion process to make it possible to expand large blocks of data by breaking them into subblocks. ACEExpBegin should always be called before expanding the first subblock (or only block) of a particular sequence.

$0209
ADBStartUp
(Void)

Starts up the ADB Tool Set.

$0309
ADBShutDown
(Void)

Shuts down the ADB Tool Set. If your application has started up the ADB Tool Set, the application must make this call before it quits.

$0409
ADBVersion
(Word)

Returns the version number of the ADB Tool Set.

$0609
ADBStatus
(Boolean)

Indicates whether the ADB Tool Set is active.

$0909
SendInfo
(Void)

Sends data to the microcontroller or to an ADB device.
Errors: $0910

Parameters:
Word *dataLength*
Pointer *dataPtr*
Word *AdbCommand*

$0A09
ReadKeyMicroData
(Void)

Receives data from the microcontroller.
Errors: $0910

Parameters:
Word *dataLength*
Pointer *dataPtr*
Word *adbCommand*

$0B09
ReadKeyMicroMem
(Void)

Reads a data byte from keyboard microcontroller ROM or RAM.
Errors: $0910

Parameters:
Pointer *dataOutPtr*
Pointer *dataInPtr*
Word *adbCommand*

$0D09
AsyncADBReceive
(Void)

Receives data from an ADB device. The ADB command byte sent assumes that the command type is Talk, which tells the addressed device to send data to the host.
Errors: $0910 $0982

Parameters:
Pointer *compPtr*
Word *adbCommand*

$0E09
SyncADBReceive
(Void)

Receives data from an ADB device. This routine is very similar to the AsyncADBReceive routine. However, SyncADBReceive needs the *inputWord* parameter, which specifies the ADB command Talk and the address and register.
Errors: $0910 $0982

Parameters:
Word *inputWord*
Pointer *compPtr*
Word *adbCommand*

$1109
ReadAbs
(Word)

Reads flags to determine whether automatic polling of an absolute device is enabled. This call must not be made by an application.

$1209
SetAbsScale
(Void)

Sets up scaling for absolute devices. Allows a generic scaling utility to support virtually any graphics tablet.

Parameters:
ScaleRecPtr *dataOutPtr*

$1309
GetAbsScale
(Void)

Reads absolute device scaling values as set by the SetAbsScale rotuine.

Parameters:
ScaleRecPtr *dataInPtr*

$1409
SRQPoll
(Void)

Adds a device to the SRQ list (if the device exists) so that an application can be notified when this device has data. Whenever an SRQ is generated, the system automatically polls any device in the SRQ list to see if it has data ready. If data is available, the routine jumps to the specified completion routine with the data and notifies the application.
Important: The completion vector is called with 8-bit m and x flags and must return via an RTL instruction with the carry clear.
Errors: $0910 $0983 $0984

Parameters:
Pointer *compPtr*
Word *adbRegAddr*

$1509
SRQRemove
(Void)

Removes from the SRQ list a device previously installed by the SRQPoll routine. The *adbRegAddr* must be the same as that used in the SRQPoll routine to install the device.
Errors: $0910 $0982

Parameters:
Word *adbRegAddr*

$1609
ClearSRQTable
(Void)

Clears the SRQ list of all entries.

$0210
CtlStartUp
(Void)

Starts up the Control Manager. Your application must make this call before it makes any other Control Manager calls.
Errors: $1001

Parameters:
Word *userID*
Word *dPageAddr*

$0310
CtlShutDown
(Void)

Shuts down the Control Manager. If your application has started up the Control Manager, the application must make this call before it quits.

$0410
CtlVersion
(Word)

Returns the version number of the Control Manager.

$0610
CtlStatus
(Boolean)

Indicates whether the Control Manager is active.

$0910
NewControl
(CtlRecHndl)

Creates a control, adds it to the beginning of the specified window's control list, and returns a handle to the the control. The returned handle will be nil if the control cannot be allocated. NewControl does not draw the control.

Parameters:
GrafPortPtr *theWindowPtr*
RectPtr *boundsRectPtr*
Pointer *titlePtr*
Word *flag*
Word *value*
Word *param1*
Word *param2*
LongProcPtr *defProcPtr*
Longint *refCon*
Pointer *colorTablePtr*

$0A10
DisposeControl
(Void)

Deletes a specified control from its window's control list and releases the memory occupied by the control record and any data structures associated with the control.

Parameters:
CtlRecHndl *theControlHandle*

$0B10
KillControls
(Void)

Disposes of all controls associated with a specified window by calling the DisposeControl routine for each control in the window's control list.

Parameters:
GrafPortPtr *theWindowPtr*

$0C10
SetCtlTitle
(Void)

Sets a specified control's *ctlData* field to a specified title and redraws the control.

Parameters:
Pointer *titlePtr*
CtlRecHndl *theControlHandle*

$0D10
GetCtlTitle
(Pointer)

Returns the value in a specified control's *ctlData* field. For controls with titles, the value is the pointer to the control's Pascal-type title string. For scroll bars, the *ctlData* field contains the view and data sizes. For custom controls, the *ctlData* field is defined by the control.

Parameters:
CtlRecHndl *theControlHandle*

$0E10
HideControl
(Void)

Makes a specified control invisible by filling the region the control occupies with the background pattern of the window's GrafPort. The routine also adds the control's enclosing rectangle to the window's update region, so that anything else previously obscured by the control reappears on the screen.

Parameters:
CtlRecHndl *theControlHandle*

$0F10
ShowControl
(Void)

Makes a specified control visible. The control is drawn in its window, but may be completely or partially obscured by overlapping windows or other objects. If the control is marked as visible as specified by bit 7 (ctlInvis) in the *ctlFlag*, ShowControl has no effect.

Parameters:
CtlRecHndl *theControlHandle*

$1010
DrawControls
(Void)

Draws all controls currently visible in a specified window. The controls are drawn in reverse order of creation; thus, in case of overlap, the controls created first appear frontmost in the window.

Parameters:
GrafPortPtr *theWindowPtr*

$1110
HiliteControl
(Void)

Changes the way a specified control is highlighted. HiliteControl calls the control's definition routine to redraw the control with its new highlighting.

Parameters:
Word *hiliteState*
CtlRecHndl *theControlHandle*

$1210
CtlNewRes
(Void)

Reinitializes resolution and mode. Call CtlNewRes after you have changed the video mode.

$1310
FindControl
(Word)

Tells in which of a specified window's controls, if any, the cursor was in when the user pressed the mouse button. The return value is the part code for the part of the control the mouse was in.

Parameters:
CtlRecHndlPtr *foundCtlPtr*
Integer *pointX*
Integer *pointY*
GrafPortPtr *theWindowPtr*

$1410
TestControl
(Word)

Tests which part of a control contains a specified point. Normally used only by FindControl and TrackControl.

Parameters:
Integer *pointX*
Integer *pointY*
CtlRecHndl *theControlHandle*

$1510
TrackControl
(Word)

Follows the mouse movements of the mouse and responds appropriately until the mouse button is released; the exact response depends on the type of control and the part of the control in which the mouse button was pressed. If highlighting is appropriate, TrackControl performs the highlighting and then removes it before returning. When the mouse button is released, TrackControl returns with the part code if the mouse is in the same part of the control that it was originally in; otherwise, TrackControl returns 0 (in which case the application should do nothing).

Parameters:
Integer *startX*
Integer *startY*
LongProcPtr *actionProcPtr*
CtlRecHndl *theControlHndl*

$1610
MoveControl
(Void)

Moves a specified control to a new location within its window. The upper-left corner of the control's enclosing rectangle is moved to the horizontal and vertical coordinates *newX* and *newY* (given in the local coordinates of the control's window); the bottom-right corner is adjusted accordingly to keep the size of the rectangle the same as before. If the control is currently visible, it's hidden and then redrawn at its new location.

Parameters:
Integer *newX*
Integer *newY*
CtlRecHndl *theControlHandle*

$1710
DragControl
(Void)

Pulls a dotted outline of the control around the screen, following the movements of the mouse until the button is released. When the mouse button is released, DragControl calls the MoveControl routine to move the control to the appropriate location.

Parameters:
Integer *startX*
Integer *startY*
RectPtr *limitRectPtr*
RectPtr *slopRectPtr*
Word *dragFlag*
CtlRecHndl *theControlHandle*

$1810
SetCtlIcons
(FontHndl)

Replaces the current icon font with a specified new font and returns the handle of the old font, or just returns the handle of the old font.

Parameters:
FontHndl *newFontHandle*

$1910
SetCtlValue
(Void)

Sets a specified control's *ctlValue* field to a specified value and redraws the control to reflect the new setting. For check boxes and radio buttons, a value of 1 fills the control with the appropriate mark and 0 clears it. For scroll bars, SetCtlValue redraws the thumb when appropriate. If the specified value is out of range, the value is pinned to the nearest endpoint of the range, as specified by the control.

Parameters:
Word *curValue*
CtlRecHndl *theControlHandle*

$1A10
GetCtlValue
(Word)

Returns a specified control's current *ctlValue* field.

Parameters:
CtlRecHndl *theControlHandle*

$1B10
SetCtlParams
(Void)

Sets new parameters to the control's definition procedure, which will set the values and redraw the control if necessary.

Parameters:
Word *param2*
Word *param1*
CtlRecHndl *theControlHandle*

$1C10
GetCtlParams
(Long)

Returns a specified control's additional parameter settings. Scroll bars use *param1* as the scroll bar's view and *param2* as the data size. Simple buttons, check boxes, radio buttons, and grow boxes do not use param1 or param2.

Parameters:
CtlRecHndl *theControlHandle*

$1D10
DragRect
(Long)

Pulls a dotted outline of a specified rectangle around the screen, following the mouse movements until the mouse button is released.

Parameters:
VoidProcPtr *actionProcPtr*
Pattern *dragPatternPtr*
Integer *startX*
Integer *startY*
RectPtr *dragRectPtr*
RectPtr *limitRectPtr*
RectPtr *slopRectPtr*
Word *dragFlag*

$1E10
GrowSize
(Long)

Returns the height and width of the size box control, using the Control Manager's current icon font. You can use this value, for example, to help you compute the size of the scroll bar.

$1F10
GetCtlDPage
(Word)

Returns the value of the Control Manager's direct page. This call is normally made only by the Dialog Manager.

$2010 **SetCtlAction** (Void)	Sets a specified control's *ctlAction* field to a pointer to a custom control action.	***Parameters:*** LongProcPtr *newActionPtr* CtlRecHndl *theControlHandle*
$2110 **GetCtlAction** (LongProcPtr)	Returns the current value of a specified control's *ctlAction* field.	***Parameters:*** CtlRecHndl *theControlHandle*
$2210 **SetCtlRefCon** (Void)	Sets a specified control's *ctlRefCon* field to a new value. The *ctlRefCon* field is reserved for the application's use and is not changed (except by this call) by the Control Manager.	***Parameters:*** Longint *newRefCon* CtlRecHndl *theControlHandle*
$2310 **GetCtlRefCon** (Long)	Returns the current value of a specified control's *ctlRefCon* field.	***Parameters:*** CtlRecHndl *theControlHandle*
$2410 **EraseControl** (Void)	Makes a specified control invisible by filling the region the control occupies with the background pattern of the window's GrafPort. Unlike the HideControl routine, EraseControl does not add the control's enclosing rectangle to the window's update region.	***Parameters:*** CtlRecHndl *theControlHandle*
$2510 **DrawOneCtl** (Void)	Draws a specified control.	***Parameters:*** CtlRecHndl *theControlHandle*

$0205
DeskStartUp
(Void)

Starts up the Desk Manager for use by an application. An application must make this call if it wishes to support new desk accessories. Your application must make this call before it makes any other Desk Manager calls. In addition, all of the tools required by NDAs must be started up before this call is made. It is also important that applications not make this call unless they completely support NDAs.

$0305
DeskShutDown
(Void)

Shuts down the Desk Manager. If your application has started up the Desk Manager, the application must make this call before it quits. In addition, this call must be made before any of the required tools are shut down.

$0405
DeskVersion
(Word)

Returns the version number of the Desk Manager.

$0605
DeskStatus
(Boolean)

Indicates whether the Desk Manager is active.

$0E05
InstallNDA
(Void)

Installs a specified new desk accessory in the system. This routine is normally called only by the operating system when the machine is booted.

Parameters:
Handle *idHandle*

$0F05
InstallCDA
(Void)

Installs a specified CDA in the system. This routine is normally called only by the operating system when the machine is booted.

Parameters:
Handle *idHandle*

$1305
SetDAStrPtr
(Void)

Changes the names of the built-in CDAs. This routine can be used to localize the built-in desk accessories.

Parameters:
Handle *altDispHandle*
Pointer *stringTablePtr*

$1405
GetDAStrPtr
(Pointer)

Returns the pointer to the table of strings containing the built-in CDA names.

$1505
OpenNDA
(Word)

Opens a specified NDA. The *idNum* passed is the same ID returned by the Menu Manager and set up by the FixAppleMenu call. If your application is using the Window Manager routine TaskMaster, the application doesn't need to make the OpenNDA call.
Errors: $0510

Parameters:
Word *idNum*

$1605
CloseNDA
(Void)

Closes a specified NDA. You normally won't use the CloseNDA routine in an application.

Parameters:
Word *refNum*

$1705
SystemClick
(Void)

Handles mouse-down events in the size, drag, zoom, and close boxes. This routine should be called when the application detects a mouse-down event in a system window. If an application is using TaskMaster, it never needs to make this call; TaskMaster does the work for it.

Parameters:
EventRecordPtr *eventRecPtr*
GrafPortPtr *theWindowPtr*
Word *findWndwResult*

$1805
SystemEdit
(Boolean)

Passes standard menu edits to system windows. The valid edit types are 1: undo; 2: cut; 3: copy; 4: paste; 5: clear.

Parameters:
Word *editType*

$1905
SystemTask
(Void)

Causes each open desk accessory to perform the periodic action defined for it, if any such action was defined and if the proper time period has elapsed since the action was last performed. The routine should be called periodically by an application to support desk accessories that perform periodic actions.

$1B05
GetNumNDAs
(unsigned Integer)

Returns the number of NDAs installed in the system.

$1C05
CloseNDAByWinPtr
(Void)

Closes the NDA whose window pointer is equal to the one that is passed. This call is handy when the system is trying to close a desk accessory because the user chose Close from the File menu. When the user chooses Close, your application should use the Window Manager FrontWindow routine to determine which window is to be closed. If the front window is not an application window, the application can pass the pointer to the CloseNDAbyWinPtr routine.
Errors: $0510 $0511

Parameters:
GrafPortPtr *theWindowPtr*

$1D05
CloseAllNDAs
(Void)

Closes all open NDAs.

$1E05
FixAppleMenu
(Void)

Adds the names of the NDAs to a specified menu. This call is used to add the names of the currently installed NDAs to a menu (usually the Apple menu). The first NDA appended to the menu is given an ID of 1, the second NDA an ID of 2, and so on.

Parameters:
Word *startingID*

$0215
DialogStartUp
(Void)

Starts up the Dialog Manager. Your application must make this call before it makes any other Dialog Manager calls.

Parameters:
Word *userID*

$0315
DialogShutDown
(Void)

Shuts down the Dialog Manager and frees any memory allocated by the Dialog Manager. If your application has started up the Dialog Manager, the application must make this call before it quits.

$0415
DialogVersion
(Word)

Returns the version number of the Dialog Manager.

$0615
DialogStatus
(Boolean)

Indicates whether the Dialog Manager is active.

$0915
ErrorSound
(Void)

Establishes the sound procedure for alerts. If you don't call ErrorSound or if you pass NIL for *soundProcPtr*, the Dialog Manager uses the standard sound procedure.

Parameters:
VoidProcPtr *soundProcPtr*

$0A15
NewModalDialog
(GrafPortPtr)

Creates a specified modal dialog and returns a pointer to the GrafPort of the new dialog.

Parameters:
RectPtr *dBoundsRectPtr*
Boolean *dVisibleFlag*
Long *dRefCon*

$0B15
NewModelessDialog
(GrafPortPtr)

Creates a specified modeless dialog and returns a pointer to the GrafPort of the new dialog.

Parameters:
RectPtr *dBoundsRectPtr*
Pointer *dTitlePtr*
GrafPortPtr *dBehindPtr*
Word *dFlag*
Long *dRefCon*
RectPtr *dFullSizePtr*

$0C15
CloseDialog
(Void)

Removes a specified dialog window from the screen and deletes it from the window list.

Parameters:
GrafPortPtr *theDialogPtr*

$0D15
NewDItem
(Void)

Adds a new item to a specified dialog's item list.
Errors: **$**150A $150B

Parameters:
GrafPortPtr *theDialogPtr*
Word *itemID*
RectPtr *itemRectPtr*
Word *itemType*
Pointer *itemDescr*
Word *itemValue*
Word *itemFlag*
Pointer *itemColorPtr*

$0E15
RemoveDItem
(Void)

Removes a specified item from a specified dialog and erases it from the screen. The routine also invalidates the item area, so that any other items behind the specified item are redrawn.
Errors: $150C

Parameters:
GrafPortPtr *theDialogPtr*
Word *itemID*

$0F15
ModalDialog
(Word)

If the frontmost window is a modal dialog box, ModalDialog repeatedly gets and handles events in the dialog's window. After the routine handles an event involving an enabled dialog item, it returns the item ID.
Errors: $150D

Parameters:
WordProcPtr *filterProcPtr*

$1015
IsDialogEvent
(Boolean)

Determines whether a specified event needs to be handled as part of a modeless dialog. If your modeless dialog contains any editLine items, you must call IsDialogEvent (and then DialogSelect), even if GetNextEvent returns FALSE; otherwise, your dialog won't receive null events and the cursor won't blink.

Parameters:
EventRecordPtr *theEventPtr*

$1115
DialogSelect
(Boolean)

Handles an event as part of a specified modeless dialog. You'll normally call DialogSelect when the IsDialogEvent routine returns TRUE, passing the event in the event record pointed to by *theEventPtr*. Normally, when DialogSelect returns TRUE, you'll do whatever is appropriate as a response to the event; and when it returns FALSE, you'll do nothing.

Parameters:
EventRecordPtr *theEventPtr*
GrafPortHndl *resultPtr*
WordPtr *itemHitPtr*

$1215
DlgCut
(Void)

Checks whether a specified dialog has any editLine items and, if so, applies the LineEdit routine LECut to the current editLine item.

Parameters:
GrafPortPtr *theDialogPtr*

$1315
DlgCopy
(Void)

Checks whether a specified dialog has any editLine items and, if so, applies the LineEdit routine LECopy to the current editLine item.

Parameters:
GrafPortPtr *theDialogPtr*

$1415
DlgPaste
(Void)

Checks whether a specified dialog has any editLine items and, if so, applies the LineEdit routine LEPaste to the current editLine item.

Parameters:
GrafPortPtr *theDialogPtr*

$1515
DlgDelete
(Void)

Checks whether a specified dialog has any editLine items and, if so, applies the LineEdit routine LEDelete to the current editLine item.

Parameters:
GrafPortPtr *theDialogPtr*

$1615
DrawDialog
(Void)

Draws the contents of a specified dialog box. Because DialogSelect, ModalDialog, and ModalDialog2 handle dialog window updating, this procedure is useful only in unusual situations. You would call it, for example, to display a dialog box that doesn't require any response, but merely tells the user what's going on during a time-consuming process.

Parameters:
GrafPortPtr *theDialogPtr*

$1715
Alert
(Word)

Invokes an alert defined by a specified alert template. The routine calls the current sound procedure, if any, passing it the sound number specified in the alert template for this stage of the alert.

Parameters:
AlertTempPtr *alertTemplatePtr*
WordProcPtr *filterProcPtr*

$1815
StopAlert
(Word)

Invokes an alert defined by a specified alert template and draws the Stop icon in the upper-left corner of the box.

Parameters:
AlertTempPtr *alertTemplatePtr*
WordProcPtr *filterProcPtr*

$1915
NoteAlert
(Word)

Performs the same functions as the Alert routine, except that before drawing the items of the alert in the alert box, NoteAlert draws the Note icon in the upper-left corner of the box. Your application can make the call as many times as necessary.

Parameters:
AlertTempPtr *alertTemplatePtr*
WordProcPtr *filterProcPtr*

$1A15
CautionAlert
(Word)

Invokes an alert defined by a specified alert template and draws the Caution icon in the upper-left corner of the box.

Parameters:
AlertTempPtr *alertTemplatePtr*
WordProcPtr *filterProcPtr*

$1B15
ParamText
(Void)

Specifies the text for four static text items as Pascal strings. The text in each ParamText string will replace the corresponding symbol ^0, ^1, ^2, or ^3 in all static text items in dialog or alert boxes. An application may make this call as many times as needed.

Parameters:
Pointer *param0Ptr*
Pointer *param1Ptr*
Pointer *param2Ptr*
Pointer *param3Ptr*

$1C15
SetDAFont
(Void)

Specifies the font for the dialog or alert window's GrafPort. SetDAFont affects statText items, editLine items, and standard controls. If you don't call this routine, the system font is used.

Parameters:
FontHndl *fontHandle*

$1E15
GetControlDItem
(CtlRecHndl)

Returns a handle to the control record for a specified item. You can then make calls to the Control Manager to change the behavior of this item.
Errors: $150C

Parameters:
GrafPortPtr *theDialogPtr*
Word *itemID*

$1F15
GetIText
(Void)

Returns the text of a specified statText or editLine item in a specified dialog box. Sufficient space for the returned text must be allocated before you call GetIText.
Errors: $150A $150C

Parameters:
GrafPortPtr *theDialogPtr*
Word *itemID*
Pointer *resultPtr*

$2015
SetIText
(Void)

Provides the text for a specified statText or editLine item in a specified dialog box and draws the item.
Errors: $150A $150C

Parameters:
GrafPortPtr *theDialogPtr*
Word *itemID*
Pointer *theStringPtr*

$2115
SelectIText
(Void)

Sets selection range or insertion point in the specified editLine item in a dialog box.

Parameters:
GrafPortPtr *theDialogPtr*
Word *itemID*
Word *startSel*
Word *endSel*

$2115
SelIText
(Void)

Sets selection range or insertion point in the specified editLine item in a dialog box. A synonym for SelectIText.

Parameters:
GrafPortPtr *theDialogPtr*
Word *itemID*
Word *startSel*
Word *endSel*

$2215
HideDItem
(Void)

Erases a specified item from a specified dialog. The item is not removed from the item list and can be displayed again by calling the ShowDItem routine. If the item is already invisible, HideDItem does nothing.
Errors: $150C

Parameters:
GrafPortPtr *theDialogPtr*
Word *itemID*

$2315
ShowDItem
(Void)

Makes visible a specified item from a specified dialog. The item may have been hidden by HideDItem or may have been invisible when created. If the item is already visible, ShowDItem does nothing.
Errors: $150C

Parameters:
GrafPortPtr *theDialogPtr*
Word *itemID*

$2415
FindDItem
(Word)

Returns the ID of the item located at a specified point in a specified dialog. The point must be expressed in global coordinates. If there is no item at the location or if the specified point is outside the specified dialog, FindDItem returns 0.

Parameters:
GrafPortPtr *theDialogPtr*
Long *thePoint*

$2515
UpdateDialog
(Void)

Redraws the part of a specified dialog that is in a specified update region.

Parameters:
GrafPortPtr *theDialogPtr*
RgnHandle *updateRgnHandle*

$2615
GetDItemType
(Word)

Returns the type of a specified item (buttonItem, radioItem, statText, and so on). If the item is currently disabled, the returned value is the type plus itemDisable.
Errors: $150C

Parameters:
GrafPortPtr *theDialogPtr*
Word *itemID*

$2715
SetDItemType
(Void)

Changes a specified item to a new specified item type. The routine does not redraw the item. This allows you to change the type of several items and then redraw all the changes at the same time.
Errors: $150C

Parameters:
Word *itemType*
GrafPortPtr *theDialogPtr*
Word *itemID*

$2815
GetDItemBox
(Void)

Returns the display rectangle of a specified item.
Errors: $150C

Parameters:
GrafPortPtr *theDialogPtr*
Word *itemID*
RectPtr *itemBoxPtr*

$2915
SetDItemBox
(Void)

Changes the display rectangle of a specified item to a new display rectangle. The routine does not redraw the item. This allows you to change the enclosing rectangle for several items and then redraw all the changes at the same time.
Errors: $150C

Parameters:
GrafPortPtr *theDialogPtr*
Word *itemID*
RectPrt *itemBoxPtr*

$2A15
GetFirstDItem
(Word)

Returns the ID of the first item in a specified dialog. If there is no item in the dialog (for example, immediately following a NewModalDialog or NewModelessDialog call), GetFirstDItem returns 0; thus, you must not have any item with an ID of 0.

Parameters:
GrafPortPtr *theDialogPtr*

$2B15
GetNextDItem
(Word)

Returns the ID of the next item in a specified dialog after a specified item. If the item is the last item in the dialog, GetNextDItem returns 0.

Parameters:
GrafPortPtr *theDialogPtr*
Word *itemID*

$2C15
ModalDialog2
(Long)

If the frontmost window is a modal dialog, ModalDialog2 repeatedly gets and handles events in the dialog's window; after handling an event involving an enabled dialog item, it returns the part code and the item ID.
Errors: $150D

Parameters:
WordProcPtr *filterProcPtr*

$2E15
GetDItemValue
(Word)

Returns the current value of a specified item. For standard controls, itemValue is the current value of the control. For other types of items, itemValue may have special meaning.
Errors: $150C

Parameters:
GrafPortPtr *theDialogPtr*
Word *itemID*

$2F15
SetDItemValue
(Void)

Sets the value of a specified item to a new desired value and redraws the item. For standard controls, *itemValue* is the current value of the control. For other types of items, *itemValue* may have special meaning.
Errors: $150C

Parameters:
Word *itemValue*
GrafPortPtr *theDialogPtr*
Word *itemID*

$3215
GetNewModalDialog
(GrafPortPtr)

Creates a modal dialog and returns a pointer to the port of the new dialog. However, instead of getting its parameters from the stack, the routine gets them from a dialog template.

Parameters:
DlgTempPtr *dialogTemplatePtr*

$3315
GetNewDItem
(Void)

Adds a new item to a specified dialog's item list using a template. You must not have any item with an ID of 0.
Errors: $150A $150C

Parameters:
GrafPortPtr *theDialogPtr*
ItemTempPtr *itemTemplatePtr*

$3415
GetAlertStage
(Word)

Returns the stage of the last occurrence of an alert as a number from 0 to 3.

$3515
ResetAlertStage
(Void)

Resets the stage of the last occurrence of an alert so that the next occurrence of that same alert will be treated as its first stage. This is useful, for example, when you've used the ParamText routine to change the text of an alert such that, from the user's point of view, it becomes a different alert.

$3615
DefaultFilter
(Boolean)

Calls the standard default filter used by the ModalDialog or Alert routine when no user filter procedure is specified. Given a pointer to an event involving dialog items, DefaultFilter filters the Apple-X, Apple-C, and Apple-V keys to make them cut, copy, and paste. The routine also interprets the Return key as a click in the default button.

Parameters:
GrafPortPtr *theDialogPtr*
EventRecordPtr *theEventPtr*
WordPtr *itemHitPtr*

$3715 **GetDefButton** (Word)	Returns the ID of the default button item in a specified dialog. If the dialog does not contain any default button, GetDefButton returns 0.	***Parameters:*** GrafPortPtr *theDialogPtr*
$3815 **SetDefButton** (Void)	Sets the ID of the default button to a specified ID. The *defButtonID* must be the ID of a button item.	***Parameters:*** Word *defButtonID* GrafPortPtr *theDialogPtr*
$3915 **DisableDItem** (Void)	Disables a specified item in a specified dialog. If the item is already disabled, DisableDItem does nothing. ***Errors:*** $150C	***Parameters:*** GrafPortPtr *theDialogPtr* Word *itemID*
$3A15 **EnableDItem** (Void)	Enables a specified item in a specified dialog. If the item is already enabled, EnableDItem does nothing. ***Errors:*** $150C	***Parameters:*** GrafPortPtr *theDialogPtr* Word *itemID*

$0206
EMStartUp
(Void)

Starts up the Event Manager, sets the size of the event queue, and sets minimum and maximum mouse clamp values. Your application must make this call before it makes any other Event Manager calls.
Errors: $0601 $0606 $0607

Parameters:
Word *dPageAddr*
Word *queueSize*
Integer *xMinClamp*
Integer *xMaxClamp*
Integer *yMinClamp*
Integer *yMaxClamp*
Word *userID*

$0306
EMShutDown
(Void)

Shuts down the Event Manager and releases any workspace allocated to it. If your application has started up the Event Manager, the application must make this call before it quits.

$0406
EMVersion
(Word)

Returns the version number of the Event Manager.

$0606
EMStatus
(Boolean)

Indicates whether the Event Manager is active.

$0A06
GetNextEvent
(Boolean)

Returns the next available event of a specified type or types. If the event is in the event queue, GetNextEvent removes the event from the queue. Events in the queue that aren't designated in the mask remain in the queue. Your application can remove the events by calling the FlushEvents routine.

Parameters:
Word *eventMask*
EventRecordPtr *eventPtr*

$0B06
EventAvail
(Boolean)

Allows an application to look at the next available event of a specified type or types. If the event is in the event queue, the event is left there for subsequent retrieval by GetNextEvent. If no event of the specified type or types is available, a null event is returned. EventAvail does not call the Desk Manager.

Parameters:
Word *eventMask*
EventRecordPtr *eventPtr*

$0C06
GetMouse
(Void)

Returns the current mouse location. Gives the location in the local coordinate system of the current GrafPort (for example, the currently active window). In contrast, the mouse location stored in the where field of an event record is always in global coordinates.

Parameters:
PointPtr *mouseLocPtr*

$0D06
Button
(Boolean)

Returns the current state of the specified mouse button. On a one-button mouse, the button number is 0.
Errors: $0605

Parameters:
Word *buttonNum*

$0E06
StillDown
(Boolean)

Tests whether the specified mouse button is still down. On a one-button mouse, the button number is 0. StillDown is a true test of whether the mouse button is still down from the original press.
Errors: $0605

Parameters:
Word *buttonNum*

$0F06
WaitMouseUp
(Boolean)

Tests whether the specified mouse button is still down. If the button is not still down from the original press, WaitMouseUp removes the preceding mouse-up event from the queue before returning FALSE. On a one-button mouse, the button number is 0.
Errors: $0605

Parameters:
Word *buttonNum*

$1006
TickCount
(Long)

Returns the current number of ticks (in sixtieths of a second) since the system was last started.

$1106
GetDblTime
(Long)

Returns the maximum difference (in ticks) between mouse-up and mouse-down events allowed for the mouse clicks to be considered a double-click. The user can adjust the maxTicks value by changing the Double-Click setting in the Control Panel.

$1206
GetCaretTime
(Long)

Returns the time (in ticks) between blinks of the caret (usually indicated by a vertical bar) marking the insertion point in text that can be edited.

$1306
SetSwitch
(Void)

Generates a switch event. Only switcher-type applications should make this call.

$1406
PostEvent
(Word)

Posts an event into the event queue. PostEvent sets the what field to the specified event type, sets the message field to the specified message, and sets the *when, where,* and *modifier* fields to the current time, mouse location, and state of the modifier keys and mouse buttons.
Errors: $0604

Parameters:
Word *eventCode*
Long *eventMsg*

$1506
FlushEvents
(Word)

Removes all queue events of the type or types specified by an event mask up to but not including the first event of any type specified by a stop mask. If the event queue doesn't contain any events of the types specified by *eventMask,* FlushEvents does nothing. To remove all events specified by *eventMask,* specify a stopMask of 0.

Parameters:
Word *eventMask*
Word *stopMask*

$1606
GetOSEvent
(Boolean)

Returns the next available queue event of any type that the mask designates. If no event of the designated types is available, GetOSEvent returns a null event. GetOSEvent doesn't return window or switch events and doesn't call the Desk Manager before returning the event.

Parameters:
Word *eventMask*
EventRecordPtr *eventPtr*

$1706
OSEventAvail
(Boolean)

Allows an application to look at the next available queue event of a specified type or types, but leaves the event in the queue. OSEventAvail returns the next available queue event of any type the mask designates. If no event of the designated types is available, OSEventAvail returns a null event.

Parameters:
Word *eventMask*
EventRecordPtr *eventPtr*

$1806
SetEventMask
(Void)

Specifies the system event mask. The system event mask controls what types of events are posted into the event queue. The Event Manager will post only those event types that correspond to bits set in the system event mask. The Event Manager will not post activate, update, or switch events because those events are not stored in the event queue.

Parameters:
Word *sysEventMask*

$1906
FakeMouse
(Void)

Allows an alternative pointing device, such as a graphics tablet, to be used in place of or in conjunction with the mouse. This call must be made only by a device driver.

Parameters:
Word *changedFlag*
Word *modLatch*
Word *xPos*
Word *yPos*
Word *ButtonStatus*

$1A06
SetAutoKeyLimit
(Void)

Controls whether repeated keystrokes are accepted into the event queue. The default size for the limit is zero, which specifies that autokey events will be enqueued only if no other events are available.

Parameters:
Word *newLimit*

$021B
FMStartUp
(Void)

Starts up the Font Manager. Your application must make this call before it makes any other Font Manager calls.
Errors: $1B01

Parameters:
Word *userID*
Word *dPageAddr*

$031B
FMShutDown
(Void)

Shuts down the Font Manager. All application-generated fonts are made purgeable. All other fonts the Font Manager knows about are disposed of if they are in memory, and then all the memory used by the Font Manager itself is released as well.
Errors: $1B03

$041B
FMVersion
(Word)

Returns the version number of the Font Manager.

$061B
FMStatus
(Boolean)

Indicates whether the Font Manager is active.

$091B
CountFamilies
(Word)

Returns the total number of distinct font families currently available to the Font Manager that match a given specification (either all families or all families that have a plain-styled font available).

Parameters:
Word *famSpecs*

$0A1B
FindFamily
(Word)

Returns the family number and name of a particular font family. The family number is returned on the stack, and the name is returned in the string pointed to by *namePtr*. This routine can be used to step through the list of all available families.

Parameters:
Word *famSpecs*
Word *positionNum*
Pointer *namePtr*

$0B1B
GetFamInfo
(Word)

Returns the name of the font family with a specified family number, placing the name wherever *namePtr* is pointing. It also returns *famStats*, with *apFamBit*, *notBaseBit*, and *notFoundBit* set to the correct values; the other bits of *famStats* are undefined.
Errors: $1B08

Parameters:
Word *famNum*
Pointer *namePtr*

$0C1B
GetFamNum
(Word)

Returns the family number corresponding to a specified font family name. The family name is pointed to by *namePtr*. The name must match exactly, including length, spaces, and uppercase and lowercase distinctions. If the name has a length of 0, an error is returned; if the name is more than 25 characters long, only the first 25 characters are used.
Errors: $1B0A

Parameters:
Pointer *namePtr*

$0D1B
AddFamily
(Void)

Enables the application to add a family number and name to the Font Manager's list of known families. Both *famNum* and the string pointed to by *namePtr* must be unique (that is, not already on the Font Manager's list).
Errors: $1B08 $1B0A

Parameters:
Word *famNum*
Pointer *namePtr*

$0E1B
InstallFont
(Void)

Finds a specified font, or the best-fit available font if the specified font isn't available. Loads the font into memory if necessary. If the best-fit font has been used and scaling has not been disabled, creates a new, scaled font to match the specified font's size. Makes the resulting font current and unpurgeable.
Errors: $1B08 $1B09 $1B0C

Parameters:
Long *desiredID*
Word *scaleWord*

$0F1B
SetPurgeStat
(Void)

Makes a specified font in memory unpurgeable or purgeable. If the routine finds the specified font, it makes it either unpurgeable (if *purgeBit* of *purgeStat* is 0) or purgeable (if *purgeBit* of *purgeStat* is 1); the other bits of *purgeStat* are not used.
Errors: $1B05 $1B06 $1B07 $1B08 $1B09

Parameters:
Long *fontID*
Word *purgeStat*

$101B
CountFonts
(Word)

Returns the number of fonts currently available to the Font Manager that fit a specified description. The *desiredID* parameter supplies a font family number, style, and size. The *fontSpecs* parameter specifies which of the characteristics in *desiredID* must be matched and which can be ignored, whether the fonts must be in memory, and whether the fonts must be real or can be scaled.
Errors: $1B08 $1B09

Parameters:
Long *desiredID*
Word *fontspecs*

$111B
FindFontStats
(Void)

Places the font ID and the FontStatBits of a particular font into a specified FontStatRec. The routine can be used to step through the list of available fonts matching the given specifications.
Errors: $1B08 $1B09

Parameters:
Long *desiredID*
Word *fontSpecs*
Word *positionNum*
FontStatRecPtr *resultPtr*

$121B
LoadFont
(Void)

Finds a particular font with a specified font ID and specifications (like the FindFontStats routine), loads the font into memory (if it is not already there), and makes the font current and unpurgeable. The routine then enlarges the QuickDraw II text buffer (if necessary) to handle the font. Finally, the routine sets the *fontID, txFace,* and *txSize* fields of the current GrafPort to the font ID, style, and size of the font that was loaded. If no such font is found, LoadFont does not change the current font.
Errors: $1B08 $1B09

Parameters:
Long *desiredID*
Word *fontSpecs*
Word *positionNum*
FontStatRecPtr *resultPtr*

$131B
LoadSysFont
(Void)

Makes the system font current, without forcing the application to know its font ID. The routine then enlarges the QuickDraw II text buffer (if necessary) to handle the system font. Finally, the routine sets the *fontID, txFace,* and *txSize* fields of the current GrafPort to the font ID, style, and size of the system font.

$141B
AddFontVar
(Void)

Enables the application to add a variation of a preexisting font family to the Font Manager's collection of available fonts. The *fontHandle* parameter specifies the font to be added. The Font Manager gets the font's family number, style, and size out of the font record itself.
Errors: $1B04

Parameters:
FontHndl *fontHandle*
Word *newSpecs*

$151B
FixFontMenu
(Void)

Appends the names of available font families to a specified menu. The names are appended in alphabetical order, with the first family name assigned a menu item ID of *startingID,* the next family name assigned a menu item ID of *startingID* + 1, and so on.

Parameters:
Word *menuID*
Word *startingID*
Word *famSpecs*

$161B
ChooseFont
(Long)

Displays a dialog box enabling the user to select a new font family, size, and style. When the dialog box is drawn, the family name, style, and size specified by *currentID* will be selected if they exist.
Errors: $1B08 $1B09

Parameters:
Long *currentID*
Word *famSpecs*

$171B
ItemID2FamNum
(Word)

Translates a menu item ID into a font family number.
Errors: $1B0B $1B04

Parameters:
Word *itemID*

$181B
FMSetSysFont
(Void)

Loads a specified font into memory (if it's not already there), makes it unpurgeable, and makes it the system font.
Errors: $1B05 $1B08 $1B09

Parameters:
Long *fontID*

$191B
FMGetSysFID
(Long)

Returns the font ID of the system font. This is the system font as set by the FMStartUp or FMSetSysFont call. If the QuickDraw II SetSysFont call has been used to set the system font, the Font Manager will not have that information.

$1A1B
FMGetCurFID
(Long)

Returns the font ID of the current font.

$1B1B
FamNum2ItemID
(Word)

Translates a font family number into a menu item ID.
Errors: $1B0B $1B04

Parameters:
Word *famNum*

$1C1B
InstallWithStats
(Void)

Installs a font and returns information about that font. When an application requests the installation of a font, the Font Manager attempts to install the requested font, but it may not be available. In such cases, the Font Manager will install the closest match it can find to the requested font. InstallWithStats installs a font just as if the application had called InstallFont, but it returns a FontStatRec in the buffer pinted to by *resultPtr*. This record contains the ID of the installed font, which may be different from the font requested. It also contains the purge status the font had before it was installed. Because purge status can be changed by installation, this information can make it easier to restore a font's purge status. The font's purge status may not be the same after installation. If you need to know an installed font's purge status, use FindFontStats.

Parameters:
Long *desiredID*
Word *scaleWord*
Long *resultPtr*

$020B
IMStartUp
(Void)

Starts up the Integer Math Tool Set. Your application must make this call before it makes any other Integer Math Tool Set calls.

$030B
IMShutDown
(Void)

Shuts down the Integer Math Tool Set. If your application has started up the Integer Math Tool Set, the application must make this call before it quits.

$040B
IMVersion
(Word)

Returns the version number of the Integer Math Tool Set.

$060B
IMStatus
(Boolean)

Indicates whether the Integer Math Tool Set is active.

$090B
Multiply
(Longint)

Multiplies two Integer inputs and produces a Longint result.

Parameters:
Integer *multiplicand*
Integer *multiplier*

$0A0B
SDivide
(IntDivRec)

Divides two Integers and produces a signed Integer quotient and a signed Integer remainder. The sign of the remainder will always be the same as the sign of the dividend.
Errors: $0B01

Parameters:
Integer *dividend*
Integer *divisor*

$0B0B
UDivide
(WordDivRec)

Divides two unsigned Integer inputs and produces an unsigned Integer quotient and an unsigned Integer remainder.
Errors: $0B01

Parameters:
Word *dividend*
Word *divisor*

$0C0B
LongMul
(LongMulRec)

Multiplies two Longint values and produces a 64-bit result.

Parameters:
Long *multiplicand*
Long *multiplier*

$0D0B
LongDivide
(LongDivRec)

Divides two unsigned Longint inputs and produces a Longint unsigned quotient and a Longint unsigned remainder.
Errors: $0B01

Parameters:
Longint *dividend*
Longint *divisor*

$0E0B
FixRatio
(Fixed)

Takes two signed Integers and produces a Fixed number as a ratio of the numerator and denominator. FixRatio doesn't check for the divide-by-zero condition, nor does it cause an error to occur when that condition happens.

Parameters:
Integer *numerator*
Integer *denominator*

$0F0B
FixMul
(Fixed)

Multiplies two Fixed inputs and produces a Fixed result. The result is the same as if two 32-bit integers were multiplied, producing a 64-bit product, and only the middle 32 bits were returned.

Parameters:
Fixed *multiplicand*
Fixed *multiplier*

$100B
FracMul
(Frac)

Multiplies two Frac inputs and returns a rounded Frac result. Overflows are pinned to the most positive or negative value, depending on the XOR of the signs of the inputs.

Parameters:
Frac *multiplicand*
Frac *multiplier*

$110B
FixDiv
(Fixed)

Divides two like inputs and returns a rounded Fixed result (no remainder). Overflows are pinned to the most positive or negative value, depending on the XOR of the signs of the inputs. The inputs can be Frac, Fixed, or signed Longint, but both must be of the same type.

Parameters:
Longint *dividend*
Longint *divisor*

$120B
FracDiv
(Frac)

Divides two like inputs and returns a rounded Frac result (no remainder). Overflows are pinned to the most positive or negative value, depending on the XOR of the signs of the inputs. The inputs can be Frac, Fixed, or signed Longint, but both must be of the same type.

Parameters:
Longint *dividend*
Longint *divisor*

$130B
FixRound
(Integer)

Takes a Fixed input and returns a rounded Integer result.

Parameters:
Fixed *fixedValue*

$140B
FracSqrt
(Frac)

Takes a Frac input and returns a rounded Frac square root. The input is considered to be unsigned with the leading bit significant; that is, the input range is from 0 to almost 4.

Parameters:
Frac *fracValue*

$150B
FracCos
(Frac)

Takes a Fixed input (in radians) and returns its Frac cosine.

Parameters:
Fixed *angle*

$160B
FracSin
(Frac)

Takes a Fixed input (in radians) and returns its Frac sine.

Parameters:
Fixed *angle*

$170B
FixATan2
(Fixed)

Takes two inputs and returns a Fixed arc tangent (in radians) of their coordinates. The inputs can be Frac, Fixed, or signed Longint, but both must be of the same type.

Parameters:
Longint *input1*
Longint *input2*

$180B
HiWord
(Word)

Returns a high-order word of a long input.

Parameters:
Long *longValue*

$190B
LoWord
(Word)

Returns a low-order word of a long input.

Parameters:
Long *longValue*

$1A0B
Long2Fix
(Fixed)

Converts a specified Longint value to its corresponding Fixed value. Overflows are pinned to the most positive or negative value, depending on the sign of the input.

Parameters:
Longint *longValue*

$1B0B
Fix2Long
(Longint)

Converts a Fixed value to its corresponding Longint value. Conversions are rounded.

Parameters:
Fixed *fixedValue*

$1C0B
Fix2Frac
(Frac)

Converts a Fixed value to its corresponding Frac value. Out-of-range values are pinned to the most positive or negative value, depending on the sign of the input.

Parameters:
Fixed *fixedValue*

$1D0B
Frac2Fix
(Fixed)

Converts a specified Frac value to its corresponding Fixed value. Conversions are rounded.

Parameters:
Frac *fracValue*

$1E0B
Fix2X
(Void)

Converts a Fixed value to its corresponding Extended value.

Parameters:
Fixed *fixedValue*
ExtendPtr *extendPtr*

$1F0B
Frac2X
(Void)

Converts a specified Frac value to its corresponding Extended value.

Parameters:
Frac *fracValue*
ExtendPtr *extendPtr*

$200B
X2Fix
(Longint)

Converts an Extended value to its corresponding Fixed value. Conversions are rounded. Overflows, NaNs (Not a Number), and Infinities are pinned to the most positive or negative value, depending on the sign of the input.

Parameters:
ExtendPtr *extendPtr*

$210B
X2Frac
(Longint)

Converts an Extended value to its corresponding Frac value. Conversions are rounded. Overflows, NaNs, and Infinities are pinned to the most positive or negative value, depending on the sign of the input.

Parameters:
ExtendPtr *extendPtr*

$220B
Int2Hex
(Void)

Takes an unsigned Integer and produces an Integer Math string representing the value in hexadecimal format. Pads the string at the left with blanks or zeros. The ASCII characters in the output string have the high-order bit clear.
Errors: $0B04

Parameters:
Word *intValue*
Pointer *strPtr*
Word *strLength*

$230B
Long2Hex
(Void)

Takes an unsigned Longint value and produces an Integer Math string representing the value in hexadecimal format. The returned string consists of digits and blanks. Pads the string at the left with blanks or zeros. The ASCII characters in the output string have the high-order bit clear.
Errors: $0B04

Parameters:
Long *longValue*
Pointer *strPtr*
Word *strLength*

$240B
Hex2Int
(Word)

Takes an Integer Math string representing a hexadecimal value and returns an unsigned Integer. The string must consist of digits and blanks. If the string does not fill up the space, pad the string at the left with blanks or zeros. The ASCII characters in the string may have the high-order bit either set or clear.
Errors: $0B02 $0B03

Parameters:
Pointer *strPtr*
Word *strLength*

$250B
Hex2Long
(Long)

Takes an Integer Math string representing a hexadecimal value and returns an unsigned Longint. The returned string consists of digits and blanks. Pads the string at the left with blanks or zeros. The ASCII characters in the string may have the high-order bit either set or clear.
Errors: $0B02 $0B03

Parameters:
Pointer *strPtr*
Word *strLength*

$260B
Int2Dec
(Void)

Takes a signed or an unsigned Integer and produces an Integer Math string representing the value in decimal format. The returned string consists of digits and blanks. Pads the string at the left with blanks or zeros.
Errors: $0B04

Parameters:
Integer *wordValue*
Pointer *strPtr*
Word *strLength*
Boolean *signedFlag*

$270B
Long2Dec
(Void)

Takes a signed or an unsigned Longint value and produces an Integer Math string representing the value in decimal format. The string must consist of digits and blanks. If the string does not fill up the space, pad the string at the left with blanks or zeros. The ASCII characters in the string have the high-order bit clear. If the longValue is signed and negative, the string will contain an ASCII minus sign to the left of the most significant digit.
Errors: $0B04

Parameters:
Longint *longValue*
Pointer *strPtr*
Word *strLength*
Boolean *signedFlag*

$280B
Dec2Int
(Integer)

Takes an Integer Math string representing a decimal value and returns a signed or an unsigned Integer. The string must consist of digits and blanks. If the string does not fill up the space, pad the string at the left with blanks or zeros. The ASCII characters in the string may have the high-order bit either set or clear. If the signedFlag is nonzero, the string may contain an ASCII plus or minus sign directly in front of the most significant digit.
Errors: $0B02 $0B03

Parameters:
Pointer *strPtr*
Word *strLength*
Boolean *signedFlag*

$290B
Dec2Long
(Longint)

Takes an Integer Math string representing a decimal value and produces a Longint value. The string should be right-justified and may be padded at the left with blanks or zeros. The ASCII characters in the string may have the high-order bit either set or clear.
Errors: $0B02 $0B03

Parameters:
Pointer *strPtr*
Word *strLength*
Boolean *signedFlag*

$2A0B
HexIt
(Long)

Takes an unsigned Integer and returns a 4-byte Integer Math string representing the value in hexadecimal format. The difference between this routine and Int2Hex is that HexIt returns its result on the stack.

Parameters:
Word *intValue*

$0214
LEStartUp
(Void)

Starts up the LineEdit Tool Set and allocates a handle for the LineEdit scrap. The scrap is initially empty. You should make this call even if your application doesn't use LineEdit, so that desk accessories, dialog boxes, and alert boxes will work correctly.
Errors: $1401

Parameters:
Word *userID*
Word *dPageAddr*

$0314
LEShutDown
(Void)

Shuts down the LineEdit Tool Set and discards the LineEdit scrap. If your application has started up the LineEdit Tool Set, the application must make this call before it quits.
Errors: $1403

$0414
LEVersion
(Word)

Returns the version number of the LineEdit Tool Set.

$0614
LEStatus
(Boolean)

Indicates whether the LineEdit Tool Set is active.

$0914
LENew
(LERecHndl)

Allocates space for text, creates and initializes an edit record for that text, and returns a handle to the new edit record. Call LENew once for every edit record you want allocated. The edit record incorporates the drawing environment of the current GrafPort and is initialized with an insertion point at character position 0.

Parameters:
RectPtr *destRectPtr*
RectPtr *viewRectPtr*
Word *maxTextLen*

$0A14
LEDispose
(Void)

Releases the memory allocated for a specified edit record. Call this routine when you're completely through with an edit record. All edit records created by calling LENew must be disposed of by calling LEDispose before calling LEShutDown.

Parameters:
LERecHndl *leRecHandle*

$0B14
LESetText
(Void)

Incorporates a copy of the specified text into the specified edit record. The selection range is set to an insertion point at the end of the text. If the textLength parameter is greater than the maximum text length allowed for the edit record, only the maximum number of characters allowed will be copied into the edit record.

Parameters:
Pointer *textPtr*
Word *textLength*
LERecHndl *leRecHandle*

$0C14
LEIdle
(Void)

Makes a blinking caret appear at the insertion point (if any) in the specified text. The caret appears only when the window and the edit record are active. LineEdit observes a minimum blink interval: No matter how often LEIdle is called, the time between blinks will never be less than the minimum interval. The user can adjust the minimum blink interval with the Control Panel desk accessory.

Parameters:
LERecHndl *leRecHandle*

$0D14
LEClick
(Void)

Controls the placement and highlighting of the selection range as determined by mouse events. Call LEClick whenever a mouse-down event occurs in the view rectangle of the edit record specified by leRecHandle and the window associated with that edit record is active. The eventPtr parameter should be a pointer to the mouse-down event record.

Parameters:
EventRecordPtr *eventPtr*
LERecHndl *leRecHandle*

$0E14
LESetSelect
(Void)

Sets the selection range in the specified text. The text selected is between selStart and selEnd in the text specified by leRecHandle. The old selection range is unhighlighted, and the new one is highlighted. If selStart equals selEnd, the selection range is an insertion point and a caret is displayed.

Parameters:
Word *selStart*
Word *selEnd*
LERecHndl *leRecHandle*

$0F14
LEActivate
(Void)

Highlights the selection range in specified text. If the selection range is an insertion point, the routine displays a caret. Your application will usually call LEActivate when an activate event is reported for a window associated with an edit record.

Parameters:
LERecHndl *leRecHandle*

$1014
LEDeactivate
(Void)

Unhighlights the selection range in the specified text. If the selection range is an insertion point, the routine removes the caret. Your application will usually call LEDeactivate when an activate event (for a window becoming inactive) is reported for a window associated with an edit record.

Parameters:
LERecHndl *leRecHandle*

$1114
LEKey
(Void)

Replaces the selection range in the specified text with a specified character and leaves an insertion point just past the inserted character. If the selection range is an insertion point, LEKey just inserts the character there. LEKey redraws the text as necessary.

Parameters:
Word *theKey*
Word *modifiers*
LERecHndl *leRecHandle*

$1214
LECut
(Void)

Removes the selection range from the specified text and places it in the LineEdit scrap. The text is redrawn as necessary. Anything previously in the scrap is deleted. If the selection range is an insertion point, the scrap is emptied.

Parameters:
LERecHndl *leRecHandle*

$1314
LECopy
(Void)

Copies the selection range from the specified text into the LineEdit scrap. Anything previously in the scrap is deleted. The selection range is not deleted. If the selection range is an insertion point, the scrap is emptied.

Parameters:
LERecHndl *leRecHandle*

$1414
LEPaste
(Void)

Replaces the selection range in the specified text with the contents of the LineEdit scrap and leaves an insertion point just past the inserted text. The text is redrawn as necessary. If the scrap is empty, the selection range is deleted. If the selection range is an insertion point, LEPaste inserts the scrap at that point.

Parameters:
LERecHndl *leRecHandle*

$1514
LEDelete
(Void)

Removes the selection range from the specified text and redraws the text as necessary. LEDelete is the same as LECut except that it doesn't transfer the selection range to the scrap. If the selection range is an insertion point, nothing happens.

Parameters:
LERecHndl *leRecHandle*

$1614
LEInsert
(Void)

Inserts specified text just before the selection range in the specified text and redraws the text as necessary. The routine doesn't affect either the selection range or the scrap. The text pointed to by textPtr should not contain a Pascal-type length byte; the length of the text is passed as the textLength parameter.

Parameters:
Pointer *textPtr*
Word *textLength*
LERecHndl *leRecHandle*

$1714
LEUpdate
(Void)

Redraws the text of the specified edit record. Your application should call LEUpdate every time an update event for a window associated with an edit window is reported. LEUpdate should be called after you call the Window Manager routine BeginUpdate and the QuickDraw II routine EraseRect and before you call the Window Manager routine EndUpdate.

Parameters:
LERecHndl *leRecHandle*

$1814
LETextBox
(Void)

Draws the specified text in a specified rectangle, justifying the text as specified. LETextBox2 supports left, right, and centered justification.

Parameters:
Pointer *textPtr*
Word *textLength*
RectPtr *rectPtr*
Word *just*

$1914
LEFromScrap
(Void)

Copies the desk scrap to the LineEdit scrap. If the number of characters in the desk scrap is greater than 256, an error is returned and the scrap is not copied. The Scrap Manager must have already been loaded and started up.
Errors: $1404

$1A14
LEToScrap
(Void)

Copies the LineEdit scrap to the desk scrap. The Scrap Manager must have already been loaded and started up.

$1B14
LEScrapHandle
(Handle)

Returns a handle to the LineEdit scrap.

$1C14
LEGetScrapLen
(Word)

Returns the length of the LineEdit scrap in bytes.

$1D14
LESetScrapLen
(Void)

Sets the size of the LineEdit scrap to a specified number of bytes. If newLength is greater than 256, it is set to 256.

Parameters:
Word *newLength*

$1E14
LESetHilite
(Void)

Sets the HiliteHook field in the edit record to point to a custom highlighting procedure. LineEdit will use that procedure to both highlight and unhighlight the selection range.

Parameters:
VoidProcPtr *hiliteProcPtr*
LERecHndl *leRecHandle*

$1F14
LESetCaret
(Void)

Sets the CaretHook field in a specified edit record to point to a custom caret-drawing procedure. LineEdit will use that procedure to both draw and erase the caret.

Parameters:
VoidProcPtr *caretProcPtr*
LERecHndl *leRecHandle*

$2014
LETextBox2
(Void)

Draws the specified text in a specified rectangle, justifying the text as specified. LETextBox2 supports left, right, centered, and fill justification and can also support embedded changes.

Parameters:
Pointer *textPtr*
Word *textLength*
RectPtr *rectPtr*
Word *just*

$2114
LESetJust
(Void)

Sets the justification style of the text of the specified edit record. The text is justified to the destination rectangle supplied by the LENew call. After you call LESetJust, call the Window Manager routine InvalRect so that the text will be redrawn using the new justification style.

Parameters:
Word *just*
LERecHndl *leRecHandle*

$2214
LEGetTextHand
(Handle)

Returns a handle to the text of a specified edit record.

Parameters:
LERecHndl *leRecHandle*

$2314
LEGetTextLen
(Word)

Returns the length of the text, in bytes, of a specified edit record.

Parameters:
LERecHndl *leRecHandle*

$021C
ListStartup
(Void)

Starts up the List Manager. Your application must make this call before it makes any other List Manager calls.

$031C
ListShutDown
(Void)

Shuts down the List Manager. If your application has started up the List Manager, the application must make this call before it quits.

$041C
ListVersion
(Word)

Returns the version number of the List Manager.

$061C
ListStatus
(Boolean)

Indicates whether the List Manager is active.

$091C
CreateList
(ListCtlRecHndl)

Calls the Control Manager routine NewControl to create a list control, using a specified list record. The routine also stores the list control's handle in the list record's *listCtl* field and passes the address of the List Manager's list-control definition procedure.

Parameters:
GrafPortPtr *theWindowPtr*
ListRecPtr *listRecPtr*

$0A1C
SortList
(Void)

Alphabetizes a specified list by rearranging the array of member records. If *comparePtr* is NIL, an internal string comparison routine is used that sorts standard alphabetical strings in ascending ASCII order. If your application needs a routine to sort members that are not standard strings, use *comparePtr* to point to that routine.

Parameters:
VoidProcPtr *comparePtr*
ListRecPtr *listRecPtr*

$0B1C
NextMember
(MemRecPtr)

Searches a specified list record, starting with a specified member, and returns the value in the *memPtr* field of the first selected member found.

Parameters:
MemRecPtr *memberPtr*
ListRecPtr *listRecPtr*

$0C1C
DrawMember
(Void)

Draws one or all members of a specified list. If your application goes directly to the member record to change the state of a member, the application should then call DrawMember.

Parameters:
MemRecPtr *memberPtr*
ListRecPtr *listRecPtr*

$0D1C
SelectMember
(Void)

Selects a specified member, deselects any other selected members in the list, and scrolls the list so that the specified member is at the top of the list display. The specified member is not selected if it is disabled.

Parameters:
MemRecPtr *memberPtr*
ListRecPtr *listRecPtr*

$0E1C
GetListDefProc
(LongProcPtr)

Returns a pointer to the list control's definition procedure. Normally, you will not need to use this call.

$0F1C
ResetMember
(MemRecPtr)

Searches a specified list record, starting with the first member, and returns the value in the memPtr field of the first selected member found.

Parameters
ListRecPtr *listRecPtr*

$101C
NewList
(Void)

Resets the list control according to a specified list record. The routine uses the *listSize*, *listStart*, and *listPointer* fields of the list record pointed to by *listRecPtr* to reset the list control. The list control's scroll bar is also readjusted, and the list is redrawn with the new list and the member pointed to by *memberPtr* selected and in view.

Parameters:
MemRecPtr *memberPtr*
ListRecPtr *listRecPtr*

$0202
MMStartUp
(Word)

Starts up the Memory Manager. Your application must make this call before it makes any other Memory Manager calls. It must save the value returned as *userID* to use when it quits.
Errors: $0207

$0302
MMShutDown
(Void)

Shuts down the Memory Manager. The *userID* must be the same as the one returned by the Memory Manager in the MMStartUp call. If your application has started up the Memory Manager, the application must make this call before it quits.

Parameters:
Word *userID*

$0402
MMVersion
(Word)

Returns the version number of the Memory Manager.

$0602
MMStatus
(Boolean)

Indicates whether the Memory Manager is active.

$0902
NewHandle
(Handle)

Creates a new block belonging to *userID* and returns the handle to the block.
Errors: $0201 $0204 $0207

Parameters:
Long *blockSize*
Word *userID*
Word *attributes*
Pointer *locationPtr*

$0A02
ReallocHandle
(Void)

Reallocates a purged block using new attributes. The *locationPtr* parameter is ignored unless the attributes parameter specifies a fixed address or fixed bank.
Errors: $0201 $0203 $0204 $0206 $0207

Parameters:
Long *blockSize*
Word *userID*
Word *attributes*
Pointer *locationPtr*
Handle *theHandle*

$0B02
RestoreHandle
(Void)

Reallocates a purged block using the same attributes, *userID,* and *blockSize* that were associated with the purged handle. The block must not have a fixed address or fixed bank. Any information in the purged block is lost.
Errors: $0201 $0203 $0206 $0208

Parameters:
Handle *theHandle*

$1002
DisposeHandle
(Void)

Discards a specified block and deallocates its handle. The block is purged regardless of locked status and purge level.
Errors: $0206

Parameters:
Handle *theHandle*

$1102
DisposeAll
(Void)

Discards all the handles and blocks belonging to a specified user ID.
Errors: $0207

Parameters:
Word *userID*

$1202
PurgeHandle
(Void)

Purges a specified purgeable block, but does not deallocate the handle. The handle remains allocated to an empty block of memory.
Errors: $0204 $0205 $0206

Parameters:
Handle *theHandle*

$1302 **PurgeAll** (Void)	Purges all the purgeable blocks for a specified user ID. If any of the specified blocks are locked, the call returns an error, but purges the unlocked blocks anyway. ***Errors:*** $0204 $0205 $0207	***Parameters:*** Word *userID*
$1802 **GetHandleSize** (Long)	Returns the size of a specified block in bytes. ***Errors:*** $0206	***Parameters:*** Handle *theHandle*
$1902 **SetHandleSize** (Void)	Changes the size of a specified block. The block can be made larger or smaller. The specified block must be unlocked. ***Errors:*** $0201 $0202 $0204 $0206	***Parameters:*** Long *newSize* Handle *theHandle*
$1A02 **FindHandle** (Handle)	Returns the handle of the block containing a specified address. If the block is not locked, it may later move. If *locationPtr* is not in any handle, then NIL is returned.	***Parameters:*** Pointer *locationPtr*
$1B02 **FreeMem** (Long)	Returns the total number of free bytes in memory, not counting memory that can be freed by purging. Because of memory fragmentation, it might not be possible to allocate a block as large as the size indicated by this routine.	
$1C02 **MaxBlock** (Long)	Returns the size of the largest free block in memory, not counting memory that can be freed by purging or compacting.	
$1D02 **TotalMem** (Long)	Returns the size of all memory, including the main 256K.	
$1E02 **CheckHandle** (Void)	Checks a handle to see whether it is valid. If the Memory Manager does not recognize *theHandle*, an error occurs. The routine is intended primarily as a debugging aid. ***Errors:*** $0206	***Parameters:*** Handle *theHandle*
$1F02 **CompactMem** (Void)	Compacts memory space. Has no effect if called during an interrupt.	
$2002 **HLock** (Void)	Locks a block specified by a handle. A locked block cannot be relocated or purged during memory compaction. ***Errors:*** $0206	***Parameters:*** Handle *theHandle*
$2102 **HLockAll** (Void)	Locks all the blocks belonging to a specified user ID. Locked blocks cannot be moved or purged during memory compaction. ***Errors:*** $0207	***Parameters:*** Word *userID*
$2202 **HUnlock** (Void)	Unlocks a specified block. Unlocked blocks can be moved or purged during memory compaction. ***Errors:*** $0206	***Parameters:*** Handle *theHandle*

$2302
HUnlockAll
(Void)

Unlocks all the blocks for a specified user ID. Unlocked blocks can be moved or purged during memory compaction.
Errors: $0207

Parameters:
Word *userID*

$2402
SetPurge
(Void)

Sets the purge level of a specified block. The purge level is a 2-bit number in the attributes word specifying the purging priority of the block. An application should use only purge levels 0 to 2. Level 3 is reserved for the System Loader.
Errors: $0206

Parameters:
Word *newPurgeLevel*
Handle *theHandle*

$2502
SetPurgeAll
(Void)

Sets the purge level of all blocks associated with a specified user ID. The purge level is a 2-bit number in the attributes word specifying the purging priority of the block. An application should use only purge levels 0 to 2. Level 3 is reserved for the System Loader.
Errors: $0207

Parameters:
Word *newPurgeLevel*
Word *userID*

$2802
PtrToHand
(Void)

Copies a specified number of bytes from a source specified by a pointer to a destination specified by a handle. PtrToHand works correctly even if the source and destination overlap or cross bank boundaries. No address validation is performed; PtrToHand simply overwrites everything.
Errors: $0202 $0206

Parameters:
Pointer *sourcePtr*
Handle *destHdndle*
Long *count*

$2902
HandToPtr
(Void)

Copies a specified number of bytes from a source to a destination, with the source specified by a handle and the destination specified by a pointer. HandToPtr works correctly even if the source and destination overlap or cross bank boundaries. No address validation is performed; HandToPtr simply overwrites everything.
Errors: $0202 $0206

Parameters:
Handle *sourceHandle*
Pointer *destPtr*
Long *count*

$2A02
HandToHand
(Void)

Copies a specified number of bytes from a source to a destination, using handles. HandToHand works correctly even if the source and destination overlap or cross bank boundaries. No address validation is performed; HandToHand simply overwrites everything.
Errors: $0202 $0206

Parameters:
Handle *sourceHandle*
Handle *destHandle*
Long *count*

$2B02
BlockMove
(Void)

Copies a specified number of bytes from a source to a destination. BlockMove works correctly even if the source and destination overlap or cross bank boundaries. No address validation is performed; BlockMove simply overwrites everything.

Parameters:
Pointer *sourcePtr*
Pointer *destPtr*
Long *count*

$2F02
RealFreeMem
(Long)

Returns the number of bytes in memory that are free, plus the number that could be made free by purging. FreeMem returns only the number of bytes that are actually free, ignoring memory that is occupied by unlocked purgeable blocks; so RealFreeMem provides a more accurate picture of available memory.

$020F
MenuStartUp
(Void)

Starts up the Menu Manager. Makes a system menu bar with no menus the current menu bar, calls Desktop in the Window Manager to reserve space for the menu bar, opens a GrafPort, and calls DrawMenuBar to draw the empty system menu bar. Your application must make this call before it makes any other Menu Manager calls.

Parameters:
Word *userID*
Word *dPageAddr*

$030F
MenuShutDown
(Void)

Shuts down the Menu Manager. The routine closes the Menu Manager's port and frees any allocated menus. If your application has started up the Menu Manager, the application must make this call before it quits.

$040F
MenuVersion
(Word)

Returns the version number of the Menu Manager.

$060F
MenuStatus
(Boolean)

Indicates whether the Menu Manager is active.

$090F
MenuKey
(Void)

Maps a character to the associated menu and item for that character. When your application receives a key-down event while the user is holding down the Apple key—or while the user is holding down the Apple key and another key if the command being invoked can be repeated—the application should call MenuKey with the character that was typed.

Parameters:
WmTaskRecPtr *taskRecPtr*
CtlRecHndl *barHandle*

$0A0F
GetMenuBar
(CtlRecHndl)

Returns the handle of the current menu bar.

$0B0F
MenuRefresh
(Void)

Called when the application is not using the Window Manager and the Menu Manager cannot restore the screen under a menu. In the attempt to recover, the Menu Manager will try to allocate a buffer large enough to save the screen part before it draws the menu; the screen will be restored from it, and then the memory buffer will be deallocated. If the buffer cannot be allocated, the Menu Manager tries to call the Window Manager (via the call the Window Manager made to MenuRefresh during initialization) to refresh the screen when the menu goes away. If neither strategy works, the Menu Manager will call the routine pointed to by *redrawRoutinePtr* to refresh the screen.

Parameters:
VoidProcPtr *redrawRoutinePtr*

$0C0F
FlashMenuBar
(Void)

Flashes the entire current menu bar by first redrawing it using the colors specified as *newInvertColor* by the SetBarColors routine and then redrawing it again using normal colors.

$0D0F
InsertMenu
(Void)

Inserts a specified menu in the menu list after a specified menu or at the front of the list. After you call this routine, you should call the DrawMenuBar routine to redraw the new menu bar.

Parameters:
CtlRecHndl *addMenuHandle*
Word *insertAfter*

$0E0F
DeleteMenu
(Void)

Removes a specified menu from the menu list. The memory for the menu is not deallocated; if you want to deallocate the memory, call the DisposeMenu routine. You should also call the DrawMenuBar routine to redraw the new menu bar.

Parameters:
Word *menuNum*

$0F0F
InsertMItem
(Void)

Inserts a menu item in a menu after a specified menu item or at the front of the list. Call CalcMenuSize to resize the menu if necessary.

Parameters:
Pointer *addItemPtr*
Word *insertAfter*
Word *menuNum*

$100F
DeleteMItem
(Void)

Removes a specified item from the current menu. Call CalcMenuSize to resize the menu if necessary.

Parameters:
Word *itemNum*

$110F
GetSysBar
(CtlRecHndl)

Returns the handle of the current system menu bar.

$120F
SetSysBar
(Void)

Sets a new system bar. The new system menu bar becomes the current menu bar.

Parameters:
CtlRecHndl *barHandle*

$130F
FixMenuBar
(Word)

Computes standard sizes for the menu bar and menus. The routine searches all the menu title fonts and uses the tallest one to compute the height of the menu bar. It also sets the *titleWidth* field in the menu record for every *titleWidth* specified as 0 and calls CalcMenuSize for each menu in the menu bar.

$140F
CountMItems
(Word)

Returns the number of items, including any dividing lines, in a specified menu.

Parameters:
Word *menuNum*

$150F
NewMenuBar
(CtlRecHndl)

Creates a default menu bar with no menus. MenuStartup calls NewMenuBar to create a default system menu bar. The upper-left corner of the default menu bar matches the port, and the width is the width of the screen. The height of the bar is 13 pixels. The menu bar is visible and has default colors of black text on a white background.

Parameters:
GrafPortPtr *theWindowPtr*

$160F
GetMHandle
(CtlRecHndl)

Returns a handle to a menu record.

Parameters:
Word *menuNum*

$170F
SetBarColors
(Void)

Sets the normal, inverse, and outline colors of the current menu bar. The *newInvertColor* parameter is the color of an item when selected; *newOutColor* is the color of the menu bar outline, menu outline, underlines, and dividing lines. Any negative values will not change the specified color.

Parameters:
Word *newBarColor*
Word *newInvertColor*
Word *newOutColor*

$180F **GetBarColors** (Long)	Returns the colors for the current menu bar.	
$190F **SetMTitleStart** (Void)	Sets the starting position for the leftmost title within the current menu bar. For square-cornered menu bars, *xStart* should be at least 1 (0 overwrites the left line of the menu bar).	***Parameters:*** Word *xStart*
$1A0F **GetMTitleStart** (Word)	Returns the starting position for the leftmost title within the current menu bar.	
$1B0F **GetMenuMgrPort** (GrafPortPtr)	Returns a pointer to the Menu Manager's port.	
$1C0F **CalcMenuSize** (Void)	Sets menu dimensions, either manually or automatically. The Menu Manager will calculate the width if *newWidth* is 0 and the height if *newHeight* is 0. To compute the width, the Menu Manager finds the widest item in the menu and then adds room for a mark and a command key. A default width is used if the menu does not contain any item text.	***Parameters:*** Word *newWidth* Word *newHeight* Word *menuNum*
$1D0F **SetMTitleWidth** (Void)	Sets the width of a title in pixels. The title width defines the area in which the user can select the menu and the area that is inverted when the title is highlighted.	***Parameters:*** Word *newWidth* Word *menuNum*
$1E0F **GetMTitleWidth** (Word)	Returns the width of a menu title. The width defines the area where the user can select the menu and the area that is inverted when the title is highlighted.	***Parameters:*** Word *menuNum*
$1F0F **SetMenuFlag** (Void)	Sets the menu to a specified state. If you change a flag that affects the appearance of a menu title, call the DrawMenuBar routine after the SetMenuFlag call.	***Parameters:*** Word *newValue* Word *menuNum*
$200F **GetMenuFlag** (Word)	Returns the menu flag for a specified menu.	***Parameters:*** Word *menuNum*
$210F **SetMenuTitle** (Void)	Specifies the title for a menu.	***Parameters:*** Pointer *newStrPtr* Word *menuNum*
$220F **GetMenuTitle** (Pointer)	Returns a pointer to the title of a menu.	***Parameters:*** Word *menuNum*
$230F **MenuGlobal** (Word)	Specifies a mask that determines how the Menu Manager performs tasks. If *menuGlobalMask* has bit 15 set to 1, the mask will be ANDed with the global flag. If *menuGlobalMask* has bit 15 set to 0, the mask will be ORed with the global flag. MenuGlobal also returns the current state of the global flag in *menuGlobalFlag*.	***Parameters:*** Word *menuGlobalMask*

$240F
SetMItem
(Void)

Specifies the name for a menu item by pointing to an item line. The menu item with *itemNum* has its pointer replaced by *newItemLinePtr*. All other parameters of the item, such as ID number, text style, marked status, and keyboard equivalent, remain the same.

Parameters:
Pointer *newItemLinePtr*
Word *itemNum*

$250F
GetMItem
(Pointer)

Returns a pointer to the name of an item.

Parameters:
Word *itemNum*

$260F
SetMItemFlag
(Void)

Sets a specified item number to be underlined or not underlined and sets the highlighting style.

Parameters:
Word *newValue*
Word *itemNum*

$270F
GetMItemFlag
(Word)

Returns the values for a specified item, such as whether it is disabled, underlined, or highlighted.

Parameters:
Word *itemNum*

$280F
SetMItemBlink
(Void)

Determines how many times menu items should blink when selected.

Parameters:
Word *count*

$290F
MenuNewRes
(Void)

Adjusts screen resolution and redraws the current system menu bar. Call this routine when the screen resolution changes.

$2A0F
DrawMenuBar
(Void)

Draws the current menu bar, along with any menu titles on the bar.

$2B0F
MenuSelect
(Void)

Draws highlighted titles, pulls down menus, and handles user interaction when a mouse button is clicked on a menu bar (see the Window Manager FindWindow routine if you're using the Window Manager). These tasks are handled automatically for the system menu bar when using TaskMaster in the Window Manager.

Parameters:
WmTaskRecPtr *taskRecPtr*
CtlRecHndl *barHandle*

$2C0F
HiliteMenu
(Void)

Highlights or unhighlights the title of a specified menu. The routine should be called with *hiliteFlag* FALSE and the *menuNum* of the selected menu after your application has finished acting on a menu selection.

Parameters:
Boolean *hiliteFlag*
Word *menuNum*

$2D0F
NewMenu
(CtlRecHndl)

Allocates space for a menu list and its items. A menu string that describes menu titles, items, and special flags must be passed to the routine. The menu pointed to by *menuStringPtr* can be inserted in the default system menu bar by an InsertMenu call.

Parameters:
Pointer *menuStringPtr*

$2E0F
DisposeMenu
(Void)

Disposes of the memory allocated for a specified menu. The menu list will no longer be usable.

Parameters:
CtlRecHndl *menuHandle*

$2F0F
InitPalette
(Void)

Reinitializes the palettes needed for the color Apple logo in the system menu bar. The routine changes the scan-line byte for lines 2 through 9 to the first color from color tables 1 through 6.

$300F **EnableMItem** (Void)	Sets a specified item to display normally and allows it to be selected.	***Parameters:*** Word *itemNum*
$310F **DisableMItem** (Void)	Sets a specified item to display in dimmed characters and does not allow it to be selected.	***Parameters:*** Word *itemNum*
$320F **CheckMItem** (Void)	Sets a menu item to display or not display a check mark to the left of the item.	***Parameters:*** Boolean *checkedFlag* Word *itemNum*
$330F **SetMItemMark** (Void)	Sets a menu item to display or not display a specified character to the left of the item. The character will appear to the left of the item's text; it will not appear or will be removed if it already appears if *mark* is 0.	***Parameters:*** Word *mark* Word *itemNum*
$340F **GetMItemMark** (Word)	Returns the current character that is displayed to the left of a specified menu item.	***Parameters:*** Word *itemNum*
$350F **SetMItemStyle** (Void)	Sets the text style for a specified menu item. If you need to change only one of the characteristics, call the GetItemStyle routine and use the current states for the attributes that should stay the same.	***Parameters:*** TextStyle *textStyle* Word *itemNum*
$360F **GetMItemStyle** (TextStyle)	Returns the text style for a specified menu item.	***Parameters:*** Word *itemNum*
$370F **SetMenuID** (Void)	Specifies a new menu number.	***Parameters:*** Word *newMenuNum* Word *curMenuNum*
$380F **SetMItemID** (Void)	Specifies the ID number of a menu item.	***Parameters:*** Word *newItemNum* Word *curItemNum*
$390F **SetMenuBar** (Void)	Sets the current menu bar. If you want the system menu bar to be the current menu bar, pass 0 for *barHandle*.	***Parameters:*** CtlRecHndl *barHandle*
$3A0F **SetMItemName** (Void)	Specifies the name of a menu item. The menu item with *itemNum* will use the string pointed to by *strPtr* whenever the menu is drawn. All other parameters of the item, such as ID number, text style, marked status, and keyboard equivalent, remain the same.	***Parameters:*** Pointer *strPtr* Word *itemNum*

$0220
MidiStartUp
(Void)

Starts up the MIDI Tool Set. Your application must make this call before it makes any other MIDI Tool Set calls.

Parameters:
Word *userID*
Word *directPages*

$0320
MidiShutDown
(Void)

Shuts down the MIDI Tool Set. If your application has started up the MIDI Tool Set, the application must make this call before it quits.

$0420
MidiVersion
(Word)

Returns the version number of the MIDI Tool Set.

$0620
MidiStatus
(Word)

Returns a Boolean value that is TRUE if the MIDI Tool Set is active; FALSE if it is not.

$0920
MidiControl
(Void)

Performs eighteen different control functions required by the MIDI Tool Set.The funcNum parameter selects which function is to be performed, and the arg parameter passes any argument required by that function.

Parameters:
Word *space*
Word *funcNum*
Long *arg*

$xx20
MidiInputPoll
(Void)

Polls the active MIDI device for arriving data and buffers them in the MIDI Tool Set buffer. Used for polling MIDI devices during interrupt time without re-enabling interrupts. If the application making this call is using Sound Tools with a version number below 2.3 then this call must not be made unless the MIDI Tool Set has been loaded and started up. This call is a jump vector ($E101B2), not a tool call, and is not accessible through the Tool Locator.

$0A20
MidiDevice
(Void)

Allows an application to select, load, and unload device drivers for use with the MIDI Tool Set. The call interprets the driverInfo parameter as that address of the driver to be loaded. The funcNum parameter specifies whether the driver is to be 0: displayed; 1: loaded; 2: unloaded.

Parameters:
Word *funcNum*
Pointer *driverInfo*

$0B20
MidiClock
(Void)

Controls operation of the optional time-stamp clock. The clock ticks once every 76 microseconds and allows events to be scheduled for precise timing. The *funcNum* parameter specifies which clock function to perform, and the *arg* parameter provides the argument to the selected function. 0: set clock value; 1: start clock; 2: stop clock; 3: set clock frequency.

Parameters:
Word *funcNum*
Long *arg*

$0C20 **MidiInfo** (Long)	Returns certain information about the state of the MIDI Tool Set. The *funcNum* parameter can specify nine different functions, whose results are returned in result. 0: number of bytes in next input packet; 1: number of bytes of data in input buffer; 2: number of bytes of data in output buffer; 3: maximum number of bytes of data in input buffer; 4: maximum number of bytes of data in output buffer; 5: address of packet being recorded by MidiRecordSeq (not yet implemented); 6: address of packet being played by MidiPlaySeq (not yet implemented); 7: time-stamp clock value; 8: time-stamp clock frequency.	***Parameters:*** Word *funcNum*
$0D20 **MidiReadPacket** (Word)	Returns the length of a packet of MIDI data that it has transferred from the input buffer to the indicated array. If no packet is available, the call returns a 0.	***Parameters:*** Pointer *arrayAddr* Word *arraySize*
$0E20 **MidiWritePacket** (Word)	Queues the specified MIDI packet into the output buffer.	***Parameters:*** Pointer *arrayAddr*

$0203
MTStartUp
(Void)

Starts up the Miscellaneous Tool Set. Your application must make this call before it makes any other Miscellaneous Tool Set calls.

$0303
MTShutDown
(Void)

Shuts down the Miscellaneous Tool Set.

$0403
MTVersion
(Word)

Returns the version number of the Miscellaneous Tool Set.

$0603
MTStatus
(Boolean)

Indicates whether the Miscellaneous Tool Set is active.

$0903
WriteBRam
(Void)

Writes 252 bytes of data from a specified memory location, plus 4 checksum bytes, to the Battery RAM.

Parameters:
Pointer *bufferPtr*

$0A03
ReadBRam
(Void)

Reads 252 bytes of data from the Battery RAM, plus 4 checksum bytes, and writes the data into a specified memory location.

Parameters:
Pointer *bufferPtr*

$0B03
WriteBParam
(Void)

Writes data to a specified parameter in Battery RAM. The data is written to the location specified by the *paramRefNum*.

Parameters:
Word *theData*
Word *paramRefNum*

$0C03
ReadBParam
(Word)

Reads data from a specified parameter in Battery RAM. The data is read from the location specified by the *paramRefNum*.

Parameters:
Word *paramRefNum*

$0D03
ReadTimeHex
(Void)

Returns the current time in hexadecimal format.

Parameters:
TimeRecPtr *resultRecPtr*

$0E03
WriteTimeHex
(Void)

Sets the current time using hexadecimal format. The value for curYear cannot be 0, 1, 2, or 3.

Parameters:
Byte *month*
Byte *day*
Byte *curYear*
Byte *hour*
Byte *minute*
Byte *second*

$0F03
ReadAsciiTime
(Void)

Reads the elapsed time since 00:00:00, January 1, 1904, converts it to ASCII time output, and places it at the specified address.

Parameters:
Pointer *bufferPtr*

$1003
SetVector
(Void)

Sets the vector address for the interrupt manager or handler specified by the *vectorRefNum*. You can retrieve the current vector address for a specified interrupt manager or handler with a GetVector call.

Parameters:
Word *vectorRefNum*
Pointer *vectorPtr*

$1103
GetVector
(Pointer)

Returns the vector address for the interrupt manager or handler for a specified vector reference number. You can use the SetVector routine to set the vector address for an interrupt manager or handler.

Parameters:
Word *vectorRefNum*

$1203
SetHeartBeat
(Void)

Installs a specified task into the Heartbeat Interrupt Task queue. The *taskPtr* parameter is the pointer to the task header.
Errors: $0303 $0304 $0305

Parameters:
Pointer *taskPtr*

$1303
DelHeartBeat
(Void)

Deletes a specified task from the Heartbeat Interrupt Task queue.
Errors: $0305 $0306

Parameters:
Pointer *taskPtr*

$1503
SysFailMgr
(Void)

Displays system failure message and halts application execution.

Parameters:
Word *errorCode*
Pointer *strPtr*

$1603
GetAddr
(Pointer)

Returns an address of a byte, word, or long parameter referenced by the firmware.
Errors: $0301

Parameters:
Word *refNum*

$1703
ReadMouse
(MouseRec)

Returns mouse position, status, and mode.
Errors: $0309

$1803
InitMouse
(Void)

Initializes mouse clamp values to $000 minimum and $3FF maximum and clears the mouse mode and status.
Errors: $0302

Parameters:
Word *mouseSlot*

$1903
SetMouse
(Void)

Sets the mouse mode.

Parameters:
Word *mouseMode*

$1A03
HomeMouse
(Void)

Positions the mouse at the minimum clamp position.

$1B03
ClearMouse
(Void)

Sets the X and Y axis to $0000 if minimum clamps are negative or to the minimum clamp position if the clamps are positive.

$1C03
ClampMouse
(Void)

Sets clamp values to new values and then sets the mouse position to the minimum clamp values. The clamp values limit the maximum and minimum X and Y coordinates for the mouse position.

Parameters:
Word *xMinClamp*
Word *xMaxClamp*
Word *yMinClamp*
Word *yMaxClamp*

$1D03
GetMouseClamp
(ClampRec)

Returns the current mouse clamp values. The values can be set by a ClampMouse call.

$1E03
PosMouse
(Void)

Positions the mouse at specified coordinates.

Parameters:
Integer *xPos*
Integer *yPos*

$1F03
ServeMouse
(Word)

Returns the mouse interrupt status.

$2003
GetNewID
(Word)

Creates a new user ID. The user ID marks memory segments as belonging to a specific application or desk accessory.
Errors: $0301 $030B

Parameters:
Word *idTag*

$2103
DeleteID
(Void)

Deletes all references to a specified user ID.

Parameters:
Word *idTag*

$2203
StatusID
(Void)

Indicates whether a specified user ID is active.
Errors: $030B

Parameters:
Word *idTag*

$2303
IntSource
(Void)

Enables or disables certain interrupt sources, as specified by the *srcRefNum* parameter.

Parameters:
Word *srcRefNum*

$2403
FWEntry
(FWRec)

Allows some Apple II emulation-mode entry points to be supported from full native mode. FWEntry preserves the state of the data bank and direct page registers before it dispatches to the firmware entry point. During the execution of the firmware task, the data bank and direct page registers are set to 0; the registers are restored on return from the firmware entry point.

Parameters:
Word *aRegValue*
Word *xRegValue*
Word *yRegValue*
Word *eModeEntryPt*

$2503
GetTick
(Long)

Returns the current value of the tick counter. The value will be incremented only if the Heartbeat Interrupt Handler is installed (always true if the Event Manager is active) and VBL interrupts are enabled.

$2603
PackBytes
(Word)

Packs bytes into a special format that uses less storage space. When the call is finished, the pointer to the area to be packed is moved forward to the next packable byte and the size of area pointed to by the second input parameter is reduced by the number of bytes traversed.

Parameters:
Handle *startHandle*
WordPtr *sizePtr*
Pointer *bufferPtr*
Word *bufferSize*

$2703
UnPackBytes
(Word)

Unpacks data from the packed format used by PackBytes. When the call is finished, the pointer to the unpacked data is positioned 1 byte past the last unpacked byte and the size of the area is reduced by the amount unpacked.

Parameters:
Pointer *bufferPtr*
Word *bufferSize*
Handle *startHandle*
WordPtr *sizePtr*

$2803
Munger
(Word)

Manipulates bytes in a string of bytes. The basic operation is to search a destination string for a target string and, if one is found, replace it with a replacement string, truncating or padding as necessary.

Parameters:
PointerPtr *destPtr*
WordPtr *destLenPtr*
Pointer *targPtr*
Word *targLen*
Pointer *replPtr*
Word *replLen*
Pointer *padPtr*

$2903
GetIRQEnable
(Word)

Returns with the hardware interrupt enable states for the interrupt sources that can be controlled by the Miscellaneous Tool Set.

$2A03
SetAbsClamp
(Void)

Sets clamp values for an absolute device to new values. The clamp values limit the X and Y position of the absolute device to the specified minimum and maximum values.

Parameters:
Word *xMinClamp*
Word *xMaxClamp*
Word *yMinClamp*
Word *yMaxClamp*

$2B03
GetAbsClamp
(ClampRec)

Returns the current values for the absolute device clamps.

$2C03
SysBeep
(Void)

Calls the Apple II monitor entry point BELL1. The bell routine can be patched out using the SetVector routine to patch out the bell vector. Any bell routine installed will be called in native mode (8-bit m and x registers). The routine must return with the carry flag cleared via an RTL instruction.

$021A
SeqStartUp
(Void)

Starts up the Note Sequencer and performs all the necessary initializations for the tool set. It also makes startup calls to the Sound Tool Set and the Note Synthesizer, so your application must make this call before it makes any other Note Sequencer calls.

Parameters:
Word *dPageAddr*
Word *mode*
Word *updateRate*
Word *increment*

$031A
SeqShutDown
(Void)

Shuts down the Note Sequencer. It frees any buffers that the tools may have allocated. If your application has started up the Note Sequencer, the application must make this call before it quits.

$041A
SeqVersion
(Word)

Returns the version number of the Note Sequencer.

$061A
SeqStatus
(Boolean)

Indicates whether the Note Sequencer is active. If it is active, the flag is non-zero; otherwise, it is zero.

$091A
SetIncr
(Void)

Sets the Note Sequencer's increment value. An application may use this facility to control the tempo of a sequence. The call will function only while a sequence is playing; but there is also a SeqItem Control command, which allows the programmer to set the increment value, so that a sequence may be written with a specified tempo. If the increment value is set to 0, the sequence will halt.

Parameters:
Word *increment*

$0A1A
ClearIncr
(Word)

Sets the Note Sequencer's increment value to 0, halting the current sequence, and returns the previous increment value. Setting the increment to 0 does not disable the Note Sequencer's interrupts, so envelopes are still updated. This means that, while the sequence will not progress, notes being played when the increment was set to 0 may hang. This call is valid only while a sequence is playing.

$0B1A
GetTimer
(Word)

Returns the value of the Note Sequencer's tick counter. While the counter is advancing, the value returned will necessarily be somewhat inexact because the value will change while the call is being executed. The call is valid only while a sequence is playing.

$0C1A
GetLoc
(LocRec)

Returns certain information about the sequence that is playing. It provides the current pattern, an index to the SeqItem that is executing, and the nesting level. Nesting level indicates how deeply control has passed into a structure with patterns nested within patterns. A nesting level of 0 indicates that the Note Sequencer is playing the top-level phrase. GetLoc is valid only while a sequence is playing.

$0D1A
SeqAllNotesOff
(Void)

Switches off all notes that are playing, but does not stop the sequence. Thus, any notes that are being held will be turned off, but the sequence will then continue. This is the correct call to use to temporarily silence all instrument voices while a sequence is active. An application can also send a MIDI AllNotesOff command to external MIDI devices by issuing this call with the high bit of the mode set.

$0E1A
SetTrkInfo
(Void)

An application should call SetTrkInfo for each track it will use before starting to play a sequence. The call assigns instruments in the current instrument table to logical tracks and determines the instruments' priorities so that the Note Sequencer can allocate generators to them correctly. The application may set the Note Sequencer to play all voices on external devices and none on the internal voices on the Apple IIGS by issuing this call with the highest bit of the *instIndex* parameter set. You must use SetInstTable to assign instruments to their respective tracks before issuing this call.

Parameters:
Word *priority*
Word *instIndex*
Word *trackNum*

$0F1A
StartSeq
(Void)

Starts the interpretation of a series of SeqItems stored at sequence.

Parameters:
VoidProcPtr *errHndlrRoutine*
VoidProcPtr *compRtnPtr*
Handle *seqHandle*

$101A
StepSeq
(Void)

Advances the Note Sequencer to the next SeqItem in the current sequence, executing the current SeqItem.

$111A
StopSeq
(Void)

Halts the interpretation of a series of SeqItems. The *next* parameter specifies the next SeqItem to be executed if the sequence is started again.

Parameters:
Word *next*

$121A
SetInstTable
(Void)

Sets the current instrument table to the one specified in *instTable.*

Parameters:
Handle *instTable*

$131A
StartInts
(Void)

Enables interrupts. Use this call to restore normal functioning after a call to StopInts.

$141A
StopInts
(Void)

Disables Note Synthesizer and Note Sequencer interrupts.

$0219
NSStartUp
(Void)

Starts up the Note Synthesizer. Your application must make this call before it makes any other Note Synthesizer calls.
Errors: $1901 $1902

Parameters:
Word *updateRate*
Pointer *userUpdateRtnPtr*

$0319
NSShutDown
(Void)

Shuts down the Note Synthesizer and turns off all generators. If your application has started up the Note Synthesizer, the application must make this call before it quits.
Errors: $1923

$0419
NSVersion
(Word)

Returns the version number of the Note Synthesizer.

$0619
NSStatus
(Boolean)

Indicates whether the Note Synthesizer is active.

$0919
AllocGen
(Word)

Requests the allocation of a sound generator. Returns a generator number from 0 to 13. The call will reallocate a generator if all generators are allocated and the specified *requestPriority* exceeds that of one of the previously allocated generators.
Errors: $1921

Parameters:
Word *requestPriority*

$0A19
DeallocGen
(Void)

Sets the named generator's allocation priority to 0. Any subsequent allocation request with a valid *requestPriority* will then succeed.
Errors: $1922

Parameters:
Word *genNumber*

$0B19
NoteOn
(Void)

Initiates the generation of a note on a specified generator. The *genNumber* parameter is the value returned by a successful call to AllocGen.
Errors: $1924

Parameters:
Word *genNumber*
Word *semitone*
Word *volume*
Pointer *instrumentPtr*

$0C19
NoteOff
(Void)

Switches the specified generator to release mode, which causes the note being generated to die out. When the note's volume is 0, the generator's priority is set to 0 and it is considered to be off.

Parameters:
Word *genNumber*
Word *semitone*

$0D19
AllNotesOff
(Void)

Turns off all Note Synthesizer generators and sets their priorities to 0.

$0E19
NSSetUpdateRate
(Word)

Sets the update rate, which controls how often the Note Synthesizer updates generator envelopes. A value of zero selects the default update rate of 500. Returns the old update rate.

Parameters:
Word *updateRate*

$OF19
NSSetUserUpdateRtn
(VoidProcPtr)

Adds an application-selected routine to the interrupt vectors that are executed during the Note Synthesizer's interrupts.

Parameters:
VoicProcPtr *updateRtn*

$0213
PMStartUp
(Void)

Starts up the Print Manager. Your application must make this call before it makes any other Print Manager calls.

Parameters:
Word *userID*
Word *dPageAddr*

$0313
PMShutDown
(Void)

Shuts down the Print Manager. If your application has started up the Print Manager, the application must make this call before it quits.

$0413
PMVersion
(Word)

Returns the version number of the Print Manager.

$0613
PMStatus
(Boolean)

Indicates whether the Print Manager is active.

$0913
PrDefault
(Void)

Fills the fields of the specified print record with the default values for the current printer.

Parameters:
PrRecHndl *printRecordHandle*

$0A13
PrValidate
(Boolean)

Checks the contents of the specified print record for compatibility with the current version number of the Print Manager and the currently installed printer. If the record is valid, the routine returns FALSE (no change). If the record is invalid, the record is adjusted to the default values for the current printer and the routine returns TRUE.

Parameters:
PrRecHndl *printRecordHandle*

$0B13
PrStlDialog
(Boolean)

Conducts a style dialog with the user to determine the page dimensions and other information needed for page setup. The initial settings displayed in the dialog box are taken from the print record. If the user confirms the dialog, the results of the dialog are saved in the specified print record, the PrValidate routine is automatically called, and the routine returns TRUE. Otherwise, the print record is left unchanged and the routine returns FALSE. Never call PrStlDialog between pages of a document.

Parameters:
PrRecHndl *printRecordHandle*

$0C13
PrJobDialog
(Boolean)

Conducts a job dialog with the user to determine the print quality, range of pages to print, and so on. The initial settings displayed in the dialog box are taken from the printer driver, where they were retained from the previous job, with the exception of the page range, which is set to All, and the number of copies, which is set to 1.

Parameters:
PrRecHndl *printRecordHandle*

$0D13
PrPixelMap
(Void)

Prints all or part of a specified pixel map.

Parameters:
LocInfoPtr *srcLocPtr*
RectPtr *srcRectPtr*
Word *colorFlag*

$0E13
PrOpenDoc
(GrafPortPtr)

Initializes a GrafPort for use in printing a document, makes it the current port, and returns a pointer to the port. You must balance every call to PrOpenDoc with a call to PrCloseDoc.

Parameters:
PrRecHndl *printRecordHandle*
GrafPortPtr *printGrafPortPtr*

$0F13
PrCloseDoc
(Void)

Closes the GrafPort being used for printing. For draft printing, PrCloseDoc ends the printing job. For spool printing, PrCloseDoc ends the spooling process.

Parameters:
GrafPortPtr *printGrafPortPtr*

$1013
PrOpenPage
(Void)

Begins a new page. The page is printed only if it falls within the page range given in the job subrecord. You must balance every call to PrOpenPage with a call to PrClosePage.

Parameters:
GrafPortPtr *printGrafPortPtr*
RectPrt *pageFramePtr*

$1113
PrClosePage
(Void)

Ends the printing of the current page. The routine signals the Print Manager that your application is finished with this page, so that the Print Manager can perform whatever close operations are required for the current printer and printing method.

Parameters:
GrafPortPtr *printGrafPortPtr*

$1213
PrPicFile
(Void)

Prints a spooled document. If spool printing is being used, your application should normally call PrPicFile after it calls the PrCloseDoc routine.

Parameters:
PrRecHndl *printRecordHandle*
GrafPortPtr *printGrafPortPtr*
PrStatusRecPtr *statusRecPtr*

$1413
PrError
(Word)

Returns the last printer error code left during the printing loop by Print Manager routines.

$1513
PrSetError
(Void)

Stores a specified value in the global variable where the Print Manager keeps its printer error code. You can use this procedure to abort a printing operation in progress by setting the error code to prAbort.

Parameters:
Word *errorNumber*

$1613
PrChooser
(Boolean)

Conducts a chooser dialog with the user to determine what printer and port driver to use.

$2313
PrDriverVer
(Word)

Returns the version number of the currently installed printer driver.

$2413
PrPortVer
(Word)

Returns the version number of the currently installed port driver.

$3413
PMUnloadDriver
(Void)

Unloads the current port driver, printer driver, or both, depending on the input parameter. Legal values for the driver parameter are 0: unload both drivers; 1: unload printer driver; 2: unload port driver. The A register is zero if the call was successful. Any other value is an error returned by the Loader call UserShutDown.

Parameters:
Word *whichDriver*

$3513
PMLoadDriver
(Void)

Loads the current printer driver, port driver, or both, depending on the input parameter. The current driver is determined by the settings saved in the Printer.Setup file. Legal values for the driver parameter are 0: unload both drivers; 1: unload printer driver; 2: unload port driver. The A register is zero if the call was successful. Any other value is an error returned by the Loader call InitialLoad.

Parameters:
Word *whichDriver*

$01
CREATE
(Void)

pBlockPtr FileRecPtr
Establishes a new directory entry for an empty file. Used to create all directories except for the volume directory file and Apple II Pascal partitions.
Errors: $07 $10 $27 $2B $40 $44 $45 $46 $47 $48 $49 $4B $52 $53 $58 $59 $5A

Parameter block:
Pointer *pathname*
Word *access*
Word *file_type*
Pointer *aux_type*
Word *storage_type*
Word *create_date*
Word *create_time*

$02
DESTROY
(Void)

pBlockPtr PathNameRecPtr
Deletes the file specified in *pBlockPtr*. Removes the file's directory entry and returns its blocks to the volume bit map.
Errors: $07 $10 $27 $2B $40 $44 $45 $46 $4A $4B $4E $50 $52 $58 $5A

Parameter block:
Pointer *pathname*

$04
CHANGE_PATH
(Void)

pBlockPtr PathNameRecPtr
Moves a file's directory entry from one subdirectory to another within the same volume.

Parameter block:
Pointer *pathname*
Pointer *new_pathname*

$05
SET_FILE_INFO
(Void)

pBlockPtr FileRecPtr
Sets the file information in the file's directory entry. An open file will not recognize changed attributes until the next time it is opened.
Errors: $07 $27 $2B $40 $44 $45 $46 $4B $4E $52 $53 $58

Parameter block:
Pointer *pathname*
Word *access*
Word *file_type* or *total_blocks*
Pointer *aux_type*
Word *storage_type*
Word *create_date*
Word *create_time*
Word *mod_date*
Word *mod_time*
Pointer *blocks_used*

$06
GET_FILE_INFO
(Void)

pBlockPtr FileRecPtr
Returns the file information from the file's directory entry. The information for an open file may not be accurate until the file is closed.
Errors: $07 $27 $40 $44 $45 $46 $4A $4B $52 $53 $58

Parameter block:
Pointer *pathname*
Word *access*
Word *file_type* or *total_blocks*
Pointer *aux_type*
Word *storage_type*
Word *create_date*
Word *create_time*
Word *mod_date*
Word *mod_time*
Pointer *blocks_used*

$08
VOLUME
(Void)

pBlockPtr VolumeRecPtr
Returns the volume name, total number of blocks, number of unallocated blocks, and file system ID number of a named device.
Errors: $07 $10 $11 $27 $28 $2E $2F $40 $45 $4A $52 $55 $57 $58

Parameter block:
Pointer *dev_name*
Pointer *vol_name*
Pointer *total_blocks*
Pointer *free_blocks*
Word *file_sys_id*

$09
SET_PREFIX
(Void)

pBlockPtr PrefixRecPtr
Assigns any of the eight legal prefix numbers (0/ .. 7/) to the specified pathname. The default prefix numbers are assigned at the launch of an application and depend on how the application was launched.
Errors: $07 $40 $53

Parameter block:
Word *prefix_num*
Pointer *prefix*

$0A
GET_PREFIX
(Void)

pBlockPtr PrefixRecPtr
Returns any current prefix specified by number, bracketing it with slashes.
Errors: $07 $53

Parameter block:
Word *prefix_num*
Pointer *prefix*

$0B
CLEAR_BACKUP_BIT
(Void)

pBlockPtr PathNameRecPtr
Clears the backup bit in a file's access byte. ProDOS sets the backup bit any time a file is altered.
Errors: $07 $40 $44 $45 $46 $4A $52 $58

Parameter block:
Pointer *pathname*

$10
OPENPD
(Void)

pBlockPtr OpenRecPtr
Opens a file for reading or writing. Allocates system resources for file access and initializes file position and buffer.
Errors: $07 $27 $40 $42 $44 $45 $46 $4A $4B $50 $52

Parameter block:
Word *ref_num*
Pointer *pathname*
Pointer *io_buffer*

$11
NEWLINE
(Void)

pBlockPtr NewLineRecPtr
Enables or disables newline mode for the READ command using the *enable_mask* passed in the call's parameter block.
Errors: $07 $43

Parameter block:
Word *ref_num*
Word *enable_mask*
Word *newline_char*

$12
READPD
(Void)

pBlockPtr FileIORecPtr
Attempts to transfer the requested number of bytes into the specified buffer. The call also places the actual number of bytes transferred into the parameter block.
Errors: $07 $27 $43 $4C $4E

Parameter block:
Word *ref_num*
Pointer *data_buffer*
Pointer *request_count*
Pointer *ransfer_count*

$13
WRITEPD
(Void)

pBlockPtr FileIORecPtr
Attempts to transfer the specified number of bytes from the specified buffer to the specified file, and returns the number of bytes successfully transferred. Updates the file position mark and, if necessary, the EOF value.
Errors: $07 $27 $2B $43 $48 $4E $5A

Parameter block:
Word *ref_num*
Pointer *data_buffer*
Pointer *request_count*
Pointer *transfer_count*

$14
CLOSEPD
(Void)

pBlockPtr FileIORecPtr
Terminates access to the file and releases all system resources allocated to it.
Errors: $07 $27 $2B $43 $5A

Parameter block:
Word *ref_num*

$15
FLUSH
(Void)

pBlockPtr FileIORecPtr
Empties an open file's buffer and updates its directory. If the reference number in the parameter block is zero, all open files are flushed.

Parameter block:
Word *ref_num*

$16
SET_MARK
(Void)

pBlockPtr MarkRecPtr
Sets the current position for read or write operations in the specified file. The position may not exceed the file's EOF value.
Errors: $07 $27 $43 $4D $5A

Parameter block:
Word *ref_num*
Pointer *position*

$17
GET_MARK
(Void)

pBlockPtr MarkRecPtr
Returns the current position for reading and writing in the specified open file.
Errors: $07 $43

Parameter block:
Word *ref_num*
Pointer *position*

$18
SET_EOF
(Void)

pBlockPtr EOFRecPtr
Sets the size in bytes of a specified write-enabled file.
Errors: $07 $27 $43 $4D $4E $5A

Parameter block:
Word *ref_num*
Pointer *eofposition*

$19
GET_EOF
(Void)

pBlockPtr EOFRecPtr
Returns the size in bytes of a specified open file.
Errors: $07 $43

Parameter block:
Word *ref_num*
Pointer *eofposition*

$1A
SET_LEVEL
(Void)

pBlockPtr WordPtr
Sets the current value of the system file level.
Errors: $07 $59

Parameter block:
Word *level*

$1B
GET_LEVEL
(Void)

pBlockPtr WordPtr
Returns the current value of the system file level.
Errors: $07

Parameter block:
Word *level*

$1C
GET_DIR_ENTRY
(Void)

pBlockPtr DirEntryRecPtr
Returns certain information from a file's directory entry.

Parameter block:
Word *refNum*
Word *reserved*
Word *base*
Word *displacement*
Pointer *nameBuffer*
Word *entryNum*
Word *fileType*
LongInt *endOfFile*
Pointer *blockCount*
TimeRec *createTime*
TimeRec *modTime*
Word *access*
Pointer *auxType*
Word *fileSysID*

$20
GET_DEV_NUM
(Void)

pBlockPtr DevNumRecPtr
Returns the device number of a named device or the volume mounted on it. All other calls except FORMAT refer to the device by number.
Errors: $07 $10 $11 $40 $45

Parameter block:
Pointer *dev_name*
Word *dev_num*

$21
GET_LAST_DEV
(Void)

pBlockPtr WordPtr
Returns the device number of the last device read from or written to.
Errors: $07 $60

Parameter block:
Word *dev_num*

$22
READ_BLOCK
(Void)

pBlockPtr BlockRecPtr
Reads one block of information into memory from a specified device. The parameter block must contain the address of a 512-byte buffer for the block when the routine is called.
Errors: $07 $11 $27 $28 $2E $2F $53

Parameter block:
Word *dev_num*
Pointer *data_buffer*
Pointer *block_num*

$23
WRITE_BLOCK
(Void)

pBlockPtr BlockRecPtr
Transfers one 512-byte buffer to the specified block of the specified device.
Errors: $07 $11 $27 $28 $2B $2F $53

Parameter block:
Word *dev_num*
Pointer *data_buffer*
Pointer *block_num*

$24
FORMAT
(Void)

pBlockPtr FormatRecPtr
Formats the volume in a specified device and gives it a specified name.
Errors: $07 $10 $11 $27 $5D

Parameter block:
Pointer *dev_name*
Pointer *vol_name*
Word *file_sys_id*

$25
ERASE_DISK
(Void)

pBlockPtr EraseDiskRecPtr
Writes the boot blocks and an empty directory and volume bitmap to the specified volume on the specified device.

Parameter block:
Pointer *devName*
Pointer *volName*
Word *fileSysID*

$27
GET_NAME
(Void)

pBlockPtr PathNameRecPtr
Returns the filename of the currently running application. The complete pathname of the application consists of the value returned by this call appended to the value returned by GET_PREFIX for prefix number 1.
Errors: $07

Parameter block:
Pointer *data_buffer*

$28
GET_BOOT_VOL
(Void)

pBlockPtr PathNameRecPtr
Returns the name of the volume from which the current copy of ProDOS was launched. ProDOS is the operating system loader and launches from any reboot or system reset. The volume name returned is identical to the prefix *.
Errors: $07

Parameter block:
Pointer *data_buffer*

$29
QUIT
(Void)

pBlockPtr QuitRecPtr
Terminates the current application, closes all open files, sets the system level to zero, and deallocates any installed interrupt handlers.
Errors: $07 $40 $46 $5C $5D $5E $5F

Parameter block:
Pointer *pathname*
Word *flags*

$2A
GET_VERSION
(Void)

pBlockPtr WordPtr
Places the version number of the ProDOS 16 system that is currently running in the specified parameter block. The returned value's high byte is the major version number, and the low byte is the minor version number.
Errors: $07

Parameter block:
Word *version*

$2C
D_INFO
(Void)

pBlockPtr DInfoRecPtr
Returns general information about a device attached to the system.

$31 **ALLOC_INTERRUPT** (Void)	*pBlockPtr* InterruptRecPtr Places the address of an interrupt handler into the interrupt vector table. ***Errors:*** $07 $25 $53	***Parameter block:*** Word *int_num* Pointer *int_code*
$32 **DEALLOC_INTERRUPT** (Void)	*pBlockPtr* InterruptRecPtr Clears the entry for an interrupt handler from the interrupt vector table. ***Errors:*** $07 $53	***Parameter block:*** Word *int_num*

$0204
QDStartUp
(Void)

Starts up QuickDraw II. Sets the current port to the standard port and clears the screen. Your application must make this call before it makes any other QuickDraw II calls.
Errors: $0401 $0410

Parameters:
Word *dPageAddr*
Word *masterSCB*
Word *maxWidth*
Word *userID*

$0304
QDShutDown
(Void)

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

$0404
QDVersion
(Word)

Returns the version number of QuickDraw II.

$0604
QDStatus
(Boolean)

Indicates whether QuickDraw II is active.

$0904
GetAddress
(Pointer)

Returns a pointer to a specified table. The current *tableIDs* are 1: screenTable; 2: conTable320; 3: conTable640. If your application is using one of these tables, make sure that the application obtains the correct pointer every time it runs. These tables will move as the ROM version changes.

Parameters:
Word *tableID*

$0A04
GrafOn
(Void)

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.

$0B04
GrafOff
(Void)

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.

$0C04
GetStandardSCB
(Word)

Returns a copy of the standard scan-line control byte in the low-order byte of the word.

$0D04
InitColorTable
(Void)

Returns a copy of the standard color table for the current mode.

Parameters:
ColorTable *tablePtr*

$0E04
SetColorTable
(Void)

Sets a 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 4096 colors. The high-order nibble of the high-order byte is ignored.
Errors: $0450

Parameters:
Word *tableNumber*
ColorTable *srcTablePtr*

$0F04
GetColorTable
(Void)

Fills a specified color table with the contents of another color table.
Errors: $0450

Parameters:
Word *tableNumber*
ColorTable *destTablePtr*

$1004
SetColorEntry
(Void)

Sets the value of a color in a specified color table.
Errors: $0450 $0451

Parameters:
Word *tableNumber*
Word *entryNumber*
ColorValue *newColor*

$1104
GetColorEntry
(Word)

Returns the value of a specified color in a specified color table.
Errors: $0450 $0451

Parameters:
Word *tableNumber*
Word *entryNumber*

$1204
SetSCB
(Void)

Sets the scan-line control byte to a specified value.
Errors: $0452

Parameters:
Word *scanLine*
Word *newSCB*

$1304
GetSCB
(Word)

Returns the value of a specified scan-line control byte.
Errors: $0452

Parameters:
Word *scanLine*

$1404
SetAllSCBs
(Void)

Sets all scan-line control bytes to a specified value.

Parameters:
Word *newSCB*

$1504
ClearScreen
(Void)

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). ClearScreen is usually used to clear the screen to a solid color.

Parameters:
Word *colorWord*

$1604
SetMasterSCB
(Void)

Sets the master scan-line control byte to a specified value. The master scan-line control byte 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:
Word *masterSCB*

$1704
GetMasterSCB
(Word)

Returns a copy of the master scan-line control byte.

$1804
OpenPort
(Void)

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:
GrafPortPtr *portPtr*

$1904
InitPort
(Void)

Initializes 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.

Parameters:
GrafPortPtr *portPtr*

$1A04
ClosePort
(Void)

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.
Warning: Never close the current port.

Parameters:
GrafPortPtr *portPtr*

$1B04
SetPort
(Void)

Makes a specified port the current GrafPort.

Parameters:
GrafPortPtr *portPtr*

$1C04
GetPort
(GrafPortPtr)

Returns a pointer to the current GrafPort.

$1D04
SetPortLoc
(Void)

Sets the current port's *locInfo* record to specified location information.

Parameters:
LocInfoPtr *locInfoPtr*

$1E04
GetPortLoc
(Void)

Gets the current port's *locInfo* record and puts it at the specified location.

Parameters:
LocInfoPtr *locInfoPtr*

$1F04
SetPortRect
(Void)

Sets the current GrafPort's port rectangle to the specified rectangle.

Parameters:
RectPtr *rectPtr*

$2004
GetPortRect
(Void)

Returns the current GrafPort's port rectangle.

Parameters:
RectPtr *rectPtr*

$2104
SetPortSize
(Void)

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:
Word *portWidth*
Word *portHeight*

$2204
MovePortTo
(Void)

Changes the location 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:
Integer *h*
Integer *v*

$2304
SetOrigin
(Void)

Adjusts the contents of the port rectangle and the bounds rectangle so that 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.

Parameters:
Integer *h*
Integer *v*

$2404
SetClip
(Void)

Copies a specified region into the clipping region. The handle to the clipping region is not changed.

Parameters:
RgnHandle *rgnHandle*

$2504
GetClip
(Void)

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

Parameters:
RgnHandle *rgnHandle*

$2604
ClipRect
(Void)

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

Parameters:
RectPtr *rectPtr*

$2704
HidePen
(Void)

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

$2804
ShowPen
(Void)

Increments the pen level. A positive pen level indicates that drawing will occur; a negative pen level indicates that drawing will not occur.

$2904
GetPen
(Void)

Returns the pen location.

Parameters:
PointPtr *pointPtr*

$2A04
SetPenState
(Void)

Sets the pen state in the GrafPort to specified values.

Parameters:
PenStatePtr *penStatePtr*

$2B04
GetPenState
(Void)

Returns the pen state from the GrafPort to a specified location.

Parameters:
PenStatePtr *penStatePtr*

$2C04
SetPenSize
(Void)

Sets the current pen size to a specified pen size.

Parameters:
Word *penWidth*
Word *penHeight*

$2D04
GetPenSize
(Void)

Returns the current pen size to a specified location.

Parameters:
PointPtr *pointPtr*

$2E04
SetPenMode
(Void)

Sets the current pen mode to a specified pen mode. The pen mode selected is determined by a set of predefined global constants and include modeCopy, notCopy, modeOr, notOr, modeXOR, notXOR, modeBIC, and notBIC.

Parameters:
Word *penMode*

$2F04
GetPenMode
(Word)

Returns the pen mode from the current GrafPort.

$3004
SetPenPat
(Void)

Sets the current pen pattern to a specified pen pattern.

Parameters:
Pattern *patternPtr*

$3104
GetPenPat
(Void)

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

Parameters:
Pattern *patternPtr*

$3204
SetPenMask
(Void)

Sets the pen mask to a specified mask.

Parameters:
Mask *maskPtr*

$3304
GetPenMask
(Void)

Returns the pen mask to a specified location.

Parameters:
Mask *maskPtr*

$3404
SetBackPat
(Void)

Sets the background pattern to a specified pattern.

Parameters:
Pattern *patternPtr*

$3504
GetBackPat
(Void)

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

Parameters:
Pattern *patternPtr*

$3604
PenNormal
(Void)

Sets the pen state to the standard state. Pen location and visibility are not changed.

$3704
SetSolidPenPat
(Void)

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 scan-line control byte indicates 320 mode, 4 bits are used; if it indicates 640 mode, 2 bits are used.

Parameters:
Word *colorNum*

$3804
SetSolidBackPat
(Void)

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 scan-line control byte indicates 320 mode, 4 bits are used; if it indicates 640 mode, 2 bits are used.

Parameters:
Word *colorNum*

$3904
SolidPattern
(Void)

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 scan-line control byte indicates 320 mode, 4 bits are used; if it indicates 640 mode, 2 bits are used.

Parameters:
Pattern *patternPtr*
Word *colorNum*

$3A04
MoveTo
(Void)

Moves the current pen location to a specified point. The point is specified in local coordinates.

Parameters:
Integer *h*
Integer *v*

$3B04
Move
(Void)

Moves the pen location a specified horizontal and vertical displacement relative to the current pen position.

Parameters:
Integer *dH*
Integer *dV*

$3C04
LineTo
(Void)

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

Parameters:
Integer *h*
Integer *v*

$3D04
Line
(Void)

Draws a line from the current pen location to a new point *dH* horizontal points and *dV* vertical points from the current point.

Parameters:
Integer *dH*
Integer *dV*

$3E04
SetPicSave
(Void)

Sets the *picSave* field in the GrafPort to a specified value. This is an internal routine that should not be used by an application.

Parameters:
Longint *picSaveValue*

$3F04
GetPicSave
(Longint)

Returns the value of the *picSave* field in the GrafPort.

$4004
SetRgnSave
(Void)

Sets the *rgnSave* field in the GrafPort to a specified value. This is an internal routine that should not be used by an application.

Parameters:
Longint *rgnSaveValue*

$4104
GetRgnSave
(Long)

Returns the value of the *rgnSave* field of the GrafPort.

$4204
SetPolySave
(Void)

Sets the *polySave* field in the GrafPort to to a specified value. This is an internal routine that should not be used by an application.

Parameters:
Longint *polySaveValue*

$4304
GetPolySave
(Long)

Returns the value of the *polySave* field of the GrafPort.

$4404
SetGrafProcs
(Void)

Sets the *grafProcs* field of the current GrafPort to a specified value.

Parameters:
QDProcsPtr *grafProcsPtr*

$4504
GetGrafProcs
(QDProcsPtr)

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

$4604
SetUserField
(Void)

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:
Longint *userFieldValue*

$4704
GetUserField
(Longint)

Returns the value of the *userField* field of the GrafPort.

$4804
SetSysField
(Void)

Sets the *sysField* field in the GrafPort to a specified value. This is an internal routine that should not be used by an application.

Parameters:
Longint *sysFieldValue*

$4904
GetSysField
(Longint)

Returns the value of the *sysField* field of the GrafPort.

$4A04
SetRect
(Void)

Sets a specified rectangle to specified values.

Parameters:
RectPtr *rectPtr*
Integer *left*
Integer *top*
Integer *right*
Integer *bottom*

$4B04
OffsetRect
(Void)

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:
RectPtr *rectPtr*
Integer *dH*
Integer *dV*

$4C04
InsetRect
(Void)

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:
RectPtr *rectPtr*
Integer *dH*
Integer *dV*

$4D04
SectRect
(Boolean)

Calculates the intersection of two rectangles and places the intersection in a third rectangle. If the result is not empty, the output is TRUE; if the result is empty, the output is FALSE.

Parameters:
RectPtr *rect1Ptr*
RectPtr *rect2Ptr*
RectPtr *intersectRectPtr*

$4E04
UnionRect
(Void)

Calculates the union of two rectangles and places the union in a third rectangle.

Parameters:
RectPtr *rect1Ptr*
RectPtr *rect2Ptr*
RectPtr *unionRectPtr*

$4F04
PtInRect
(Boolean)

Detects 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.

Parameters:
Pointer *pointPtr*
RectPtr *rectPtr*

$5004
Pt2Rect
(Void)

Copies one specified point to the upper-left corner of a specified rectangle and another point to the lower-right corner of the rectangle.

Parameters:
Pointer *point1Ptr*
Pointer *point2Ptr*
RectPtr *rectPtr*

$5104
EqualRect
(Boolean)

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:
RectPtr *rect1Ptr*
RectPtr *rect2Ptr*

$5204
EmptyRect
(Boolean)

Indicates whether or not the specified rectangle is empty. Returns TRUE if the specified rectangle is not empty. This call is a synonym for NotEmptyRect.

Parameters:
RectPtr *rectPtr*

$5204
NotEmptyRect
(Boolean)

Indicates whether or not the specified rectangle is empty. Returns TRUE if the specified rectangle is not empty.

Parameters:
RectPtr *rectPtr*

$5304
FrameRect
(Void)

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.

Parameters:
RectPtr *rectPtr*

$5404
PaintRect
(Void)

Paints the interior of a specified rectangle using the current pen mode and pen pattern.

Parameters:
RectPtr *rectPtr*

$5504
EraseRect
(Void)

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

Parameters:
RectPtr *rectPtr*

$5604
InvertRect
(Void)

Inverts the pixels in the interior of a specified rectangle.

Parameters:
RectPtr *rectPtr*

$5704
FillRect
(Void)

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

Parameters:
RectPtr *rectPtr*
Pattern *patternPtr*

$5804
FrameOval
(Void)

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.

Parameters:
RectPtr *rectPtr*

$5904
PaintOval
(Void)

Paints the interior of a specified oval using the current pen mode and pen pattern.

Parameters:
RectPtr *rectPtr*

$5A04
EraseOval
(Void)

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

Parameters:
RectPtr *rectPtr*

$5B04
InvertOval
(Void)

Inverts the pixels in the interior of a specified oval.

Parameters:
RectPtr *rectPtr*

$5C04
FillOval
(Void)

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

Parameters:
RectPtr *rectPtr*
Pattern *patternPtr*

$5D04
FrameRRect
(Void)

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

Parameters:
RectPtr *rectPtr*
Word *ovalWidth*
Word *ovalHeight*

$5E04
PaintRRect
(Void)

Paints the interior of a specified round rectangle using the current pen mode and pen pattern.

Parameters:
RectPtr *rectPtr*
Word *ovalWidth*
Word *ovalHeight*

$5F04
EraseRRect
(Void)

Erases 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*.

Parameters:
RectPtr *rectPtr*
Word *ovalWidth*
Word *ovalHeight*

$6004
InvertRRect
(Void)

Inverts the pixels in the interior of a specified round rectangle. The corners of the round rectangle are sections of an oval defined by *ovalHeight* and *ovalWidth*.

Parameters:
RectPtr *rectPtr*
Word *ovalWidth*
Word *ovalHeight*

$6104
FillRRect
(Void)

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 *ovalHeight* and *ovalWidth*.

Parameters:
RectPtr *rectPtr*
Word *ovalWidth*
Word *ovalHeight*
Pattern *patternPtr*

$6204
FrameArc
(Void)

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:
RectPtr *rectPtr*
Integer *startAngle*
Integer *arcAngle*

$6304
PaintArc
(Void)

Paints the interior of a specified arc using the current pen mode and pen pattern.

Parameters:
RectPtr *rectPtr*
Integer *startAngle*
Integer *arcAngle*

$6404
EraseArc
(Void)

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

Parameters:
RectPtr *rectPtr*
Integer *startAngle*
Integer *arcAngle*

$6504
InvertArc
(Void)

Inverts the pixels in the interior of a specified arc.

Parameters:
RectPtr *rectPtr*
Integer *startAngle*
Integer *arcAngle*

$6604
FillArc
(Void)

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

Parameters:
RectPtr *rectPtr*
Integer *startAngle*
Integer *arcAngle*
Pattern *patternPtr*

$6704
NewRgn
(RgnHandle)

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). NewRgn is the only routine that creates a new region; all other routines work with existing regions.

$6804
DisposeRgn
(Void)

Deallocates the memory for a specified region.

Parameters:
Rgnhandle *rgnHandle*

$6904
CopyRgn
(Void)

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

Parameters:
RgnHandle *srcRgnHandle*
RgnHandle *destRgnHandle*

$6A04
SetEmptyRgn
(Void)

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:
Rgnhandle *rgnHandle*

$6B04
SetRectRgn
(Void)

Destroys previous region information by setting a specified region to a rectangle described by the input. 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.

Parameters:
Rgnhandle *rgnHandle*
Integer *left*
Integer *top*
Integer *right*
Integer *bottom*

$6C04
RectRgn
(Void)

Destroys previous region information by setting a specified region to a rectangle described by the input. 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:
Rgnhandle *rgnHandle*
RectPtr *rectPtr*

$6D04
OpenRgn
(Void)

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.
Errors: $0430

$6E04
CloseRgn
(Void)

Completes the region-definition process started by an OpenRgn call. The region must have already been created by a NewRgn call; that call supplies *destRgnHandle*.
Errors: $0431

Parameters:
Rgnhandle *rgnHandle*

$6F04
OffsetRgn
(Void)

Moves a region on the coordinate plane a distance of *dH* horizontally and *dV* vertically. The region retains its size and shape.

Parameters:
Rgnhandle *rgnHandle*
Integer *dH*
Integer *dV*

$7004
InsetRgn
(Void)

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:
Rgnhandle *rgnHandle*
Integer *dH*
Integer *dV*

$7104
SectRgn
(Void)

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:
Rgnhandle *rgn1Handle*
RgnHandle *rgn2Hdndle*
RgnHandle *destRgnHandle*

$7204
UnionRgn
(Void)

Calculates the union of two regions and places the union in a third region. The destination region must already exist; UnionRgn does not allocate it. However, the destination region may be one of the source regions. If both regions are empty, the destination is set to the empty region.

Parameters:
Rgnhandle *rgn1Handle*
RgnHandle *rgn2Handle*
RgnHandle *unionRgnHandle*

$7304
DiffRgn
(Void)

Calculates the difference of the areas enclosed by two regions and places it in a third region. The destination region must already exist; DiffRgn does not allocate it. However, the destination region may be one of the source regions. If the *rgn1Handle* is empty, the destination is set to an empty region.

Parameters:
RgnHandle *rgn1Handle*
RgnHandle *rgn2Handle*
RgnHandle *diffRgnHandle*

$7404
XorRgn
(Void)

Calculates the difference between the union and the intersection of two regions and places the result in a third region. The destination region must already exist; XorRgn does not allocate it. However, the destination region may be one of the source regions. If the regions are not coincident, the destination is set to the empty region.

Parameters:
Rgnhandle *rgn1Handle*
RgnHandle *rgn2Handle*
RgnHandle *xorRgnHandle*

$7504
PtInRgn
(Boolean)

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:
Pointer *pointPtr*
RgnHandle *rgnHandle*

$7604
RectInRgn
(Boolean)

Checks whether a specified rectangle intersects a specified region.

Parameters:
RectPtr *rectPtr*
RgnHandle *rgnHandle*

$7704
EqualRgn
(Boolean)

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:
RgnHandle *rgn1Handle*
RgnHandle *rgn2Handle*

$7804
EmptyRgn
(Boolean)

Indicates whether a specified region is empty.

Parameters:
Rgnhandle *rgnHandle*

$7904
FrameRgn
(Void)

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.

Parameters:
Rgnhandle *rgnHandle*

$7A04
PaintRgn
(Void)

Paints the interior of a specified region using the current pen mode and pen pattern.

Parameters:
Rgnhandle *rgnHandle*

$7B04
EraseRgn
(Void)

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

Parameters:
Rgnhandle *rgnHandle*

$7C04
InvertRgn
(Void)

Inverts the pixels in the interior of a specified region.

Parameters:
Rgnhandle *rgnHandle*

$7D04
FillRgn
(Void)

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

Parameters:
Rgnhandle *rgnHandle*
Pattern *patternPtr*

$7E04
ScrollRect
(Void)

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). The backgound pattern fills the space created by the scroll. The region for *updateRgnHandle* is changed to the area filled with the background pattern.

Parameters:
RectPtr *rectPtr*
Integer *dH*
Integer *dV*
RgnHandle *rgnHandle*

$7F04
PaintPixels
(Void)

Transfers a region of pixels without referencing the current GrafPort. The source and destination are defined in the input, as is the clipping region.
Errors: $0420

Parameters:
PaintParamPtr *paintParamPtr*

$8004
AddPt
(Void)

Adds two specified points together and leaves the result in the destination point.

Parameters:
Pointer *srcPtPtr*
Pointer *destPtPtr*

$8104
SubPt
(Void)

Subtracts the source point from the destination point and leaves the result in the destination point.

Parameters:
Pointer *srcPtPtr*
Pointer *destPtPtr*

$8204
SetPt
(Void)

Sets a point to specified horizontal and vertical values.

Parameters:
Pointer *srcPtPtr*
Integer *h*
Integer *v*

$8304
EqualPt
(Boolean)

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

Parameters:
Pointer *point1Ptr*
Pointer *point2Ptr*

$8404
LocalToGlobal
(Void)

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

Parameters:
PointPtr *pointPtr*

$8504
GlobalToLocal
(Void)

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 *boundsRect* of the GrafPort.

Parameters:
Pointer *pointPtr*

$8604
Random
(Integer)

Returns a pseudorandom number in the range –32768 to 32767. The sequence of numbers generated by repeated calls to this routine depends on the *randomSeed* value set by a SetRandSeed call. In particular, a call to SetRandSeed with a given *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.

$8704
SetRandSeed
(Void)

Sets the seed value for the random-number generator. The algorithm uses a 32-bit seed to produce a 16-bit random number.

Parameters:
Longint *randomSeed*

$8804
GetPixel
(Word)

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, then 2 bits of the word are valid. If the current drawing location has a chunkiness of 4, then 4 bits of the word are valid.

Parameters:
Integer *h*
Integer *v*

$8904
ScalePt
(Void)

Scales a specified point from a source rectangle to a destination rectangle.

Parameters:
Pointer *pointPtr*
RectPtr *srcRectPtr*
RectPtr *destRectPtr*

$8A04
MapPt
(Void)

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

Parameters:
Pointer *pointPtr*
RectPtr *srcRectPtr*
RectPtr *destRectPtr*

$8B04
MapRect
(Void)

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

Parameters:
RectPtr *rectPtr*
RectPtr *srcRectPtr*
RectPtr *destRectPtr*

$8C04
MapRgn
(Void)

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

Parameters:
Rgnhandle *mapRgnHandle*
RectPtr *srcRectPtr*
RectPtr *destRectPtr*

$8D04
SetStdProcs
(Void)

Sets up a specified record of pointers for customizing QuickDraw II operations.

Parameters:
QDProcsPtr *stdProcRecPtr*

$8E04
SetCursor
(Void)

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.

Parameters:
Pointer *cursorPtr*

$8F04
GetCursorAdr
(Pointer)

Returns a pointer to the current cursor record.

$9004
HideCursor
(Void)

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.

$9104
ShowCursor
(Void)

Shows the cursor by incrementing the cursor level (as long as the level is not already 0). A cursor level of 0 indicates the cursor is visible; a cursor level less than 0 indicates the cursor is not visible.

$9204
ObscureCursor
(Void)

Hides the cursor until the mouse moves. This routine can get the cursor out of the way of typing.

$9404
SetFont
(Void)

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. 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:
FontHndl *newFontHandle*

$9504
GetFont
(FontHndl)

Returns a handle to the current font.

$9604
GetFontInfo
(Void)

Returns information about the current font in a specified buffer. The information in the *fontInfo* record reflects current style modifications, but not the values of the *chExtra* and *spExtra* fields of the GrafPort. Your application can use the information returned in the *fontInfo* record to determine the spacing between lines of text.

Parameters:
FontInfoRecPtr *fontInfoRecPtr*

$9704
GetFontGlobals
(Void)

Returns information about the font globals record into a specified buffer. The size of the font globals record is returned by the GetFGSize routine. The information represents the GrafPort's current font and does not reflect style modifications, *chExtra* or *spExtra* fields, and so on. This call is primarily intended to provide backward compatibility. Under normal circumstances, you'll probably prefer to use the GetFontLore routine.

Parameters:
FontGlobalsRecPtr *fgRecPtr*

$9804
SetFontFlags
(Void)

Sets the font flags word to a specified value. The font flags word is used to indicate special operations performed on the text.

Parameters:
Word *fontFlags*

$9904
GetFontFlags
(Word)

Returns the current font flags word.

$9A04
SetTextFace
(Void)

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; for example, bits 0 through 4 represent bold, italic, underline, outline, and shadow, respectively.

Parameters:
TextStyle *textFace*

$9B04
GetTextFace
(TextStyle)

Returns the current text face.

$9C04
SetTextMode
(Void)

Sets the text mode to a specified value. The fastest modes are those that transfer only the foreground to the destination. The fastest of the foreground modes are modeForeOR and modeForeXOR; modeForeBIC is almost as fast, and modeForeCOPY is the slowest. In additon to the text-only modes, the pen modes apply to text.

Parameters:
Word *textMode*

$9D04
GetTextMode
(Word)

Returns the current text mode.

$9E04
SetSpaceExtra
(Void)

Sets the *spExtra* field in the GrafPort to a specified value. The *spExtra* field is used by programs that are trying to justify text to a left and right boundary. When the *spExtra* field is nonzero, its value is added to the width of each space printed in a string.

Parameters:
Fixed *spaceExtra*

$9F04
GetSpaceExtra
(Fixed)

Returns the value of the *spExtra* field from the GrafPort.

$A004
SetForeColor
(Void)

Sets the *fgColor* (foreground color) field in the GrafPort to a specified value. Foreground color has either a 2- or a 4-bit value, depending on the port scan-line control byte. If the port scan-line control byte indicates 320 mode, the lower 4 bits of *foreColor* are used. If the port scan-line control byte indicates 640 mode, the lower 2 bits of *foreColor* are used.

Parameters:
Word *foreColor*

$A104
GetForeColor
(Word)

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

$A204
SetBackColor
(Void)

Sets the *bgColor* (background color) field in the GrafPort to a specified value. Background color has either a 2- or a 4-bit value, depending on the port scan-line control byte. If the port scan-line control byte indicates 320 mode, the lower 4 bits of *backColor* are used. If the port scan-line control byte indicates 640 mode, the lower 2 bits of *backColor* are used.

Parameters:
Word *backColor*

$A304 **GetBackColor** (Word)	Returns the value of the *bgColor* field from the GrafPort.	
$A404 **DrawChar** (Void)	Draws a specified character at the current pen location and updates the pen location.	***Parameters:*** Word *theChar*
$A504 **DrawString** (Void)	Draws a specified string at the current pen location and updates the pen location. The string pointed to by *stringPtr* must contain a length byte followed by the characters of the string.	***Parameters:*** Pointer *stringPtr*
$A604 **DrawCString** (Void)	Draws a specified C string at the current pen location and updates the pen location (C strings are terminated by $00).	***Parameters:*** Pointer *cStringPtr*
$A704 **DrawText** (Void)	Draws specified text at the current pen location and updates the pen location.	***Parameters:*** Pointer *textPtr* Word *textLength*
$A804 **CharWidth** (Integer)	Returns the character width, in pixels (pen displacement), of a specified character.	***Parameters:*** Word *theChar*
$A904 **StringWidth** (Integer)	Returns the sum of all the character widths (pen displacements) of a specified Pascal-type string. This would be the pen displacement if the string were to be drawn.	***Parameters:*** Pointer *stringPtr*
$AA04 **CStringWidth** (Integer)	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:*** Pointer *cStringPtr*
$AB04 **TextWidth** (Integer)	Returns the character width, in pixels (pen displacement), of specified text.	***Parameters:*** Pointer *textPtr* Word *textLength*
$AC04 **CharBounds** (Void)	Places the character bounds rectangle of a specified character into a specified buffer.	***Parameters:*** Word *theChar* RectPtr *resultPtr*
$AD04 **StringBounds** (Void)	Puts the string bounds rectangle of a specified Pascal-type string into a specified space.	***Parameters:*** Pointer *stringPtr* RectPtr *resultPtr*
$AE04 **CStringBounds** (Void)	Places the character bounds rectangle of a specified C string into a specified buffer.	***Parameters:*** Pointer *cStringPtr* RectPtr *resultPtr*
$AF04 **TextBounds** (Void)	Places the character bounds rectangle of specified text into a specified buffer.	***Parameters:*** Pointer *textPtr* Word *textLength* RectPtr *resultPtr*
$B004 **SetArcRot** (Void)	Sets the *arcRot* field in the GrafPort to a specified value.	***Parameters:*** Integer *arcRot*

$B104
GetArcRot
(Integer)

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

$B204
SetSysFont
(Void)

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 *fontHandle* field of each GrafPort when it is opened or initialized.

Parameters:
FontHndl *fontHandle*

$B304
GetSysFont
(FontHndl)

Returns a handle to the current system font.

$B404
SetVisRgn
(Void)

Copies a specified region into the visible region, but does not change the *visRgn* field of the GrafPort.

Parameters:
Rgnhandle *rgnHandle*

$B504
GetVisRgn
(Void)

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

Parameters:
Rgnhandle *rgnHandle*

$B604
SetIntUse
(Void)

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:
Word *useInt*

$B704
OpenPicture
(Handle)

Allocates memory for the recording of drawing commands into a picture definition, and returns a handle to the picture.

Parameters:
Pointer *picFrame*

$B804
PicComment
(Void)

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 *grafProcs* field of the GrafPort.

Parameters:
Integer *kind*
Integer *dataSize*
Handle *dataHandle*

$B904
ClosePicture
(Void)

Completes the picture-definition process begun by an OpenPicture call. Calls to OpenPicture and ClosePicture must be balanced; that is, one ClosePicture call must be made for every OpenPicture call.

$BA04
DrawPicture
(Void)

Takes the drawing commands recorded in the picture definition, maps them from the picture frame into the destination rectangle, and draws them.

Parameters:
Handle *picHandle*
RectPtr *destRect*

$BB04
KillPicture
(Void)

Releases all memory occupied by a specified picture. Use this call only when your application is completely through with a picture.

Parameters:
Handle *pichandle*

$BC04
FramePoly
(Void)

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.

Parameters:
PolyHndl *polyHandle*

$BD04
PaintPoly
(Void)

Paints a specified polygon using the current pen mode and pen pattern.

Parameters:
Handle *polyHandle*

$BE04
ErasePoly
(Void)

Erases 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.

Parameters:
PolyHndl *polyHandle*

$BF04
InvertPoly
(Void)

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

Parameters:
Handle *polyHandle*

$C004
FillPoly
(Void)

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.

Parameters:
Handle *polyHandle*
Pattern *patternPtr*

$C104
OpenPoly
(Handle)

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.
Errors: $0440

$C204
ClosePoly
(Void)

Completes the polygon-definition process started with an OpenPoly call.
Errors: $0441

$C304
KillPoly
(Void)

Disposes of a specified polygon and frees the memory allocated for it.

Parameters:
Handle *polyHandle*

$C404
OffsetPoly
(Void)

Offsets a specified polygon by specified horizontal and vertical displacements. The value of *dH* is added to the left and right; the value of *dV* is added to the top and bottom.

Parameters:
Handle *polyHandle*
Integer *dH*
Integer *dV*

$C504
MapPoly
(Void)

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

Parameters:
Handle *polyHandle*
RectPtr *srcRectPtr*
RectPtr *destRectPtr*

$C604
SetClipHandle
(Void)

Sets the *clipRgn* handle field in the GrafPort to a specified value.

Parameters:
Rgnhandle *rgnHandle*

$C704
GetClipHandle
(RgnHandle)

Returns a copy of the handle to the clipping region.

$C804
SetVisHandle
(Void)

Sets the *visRgn* field in the GrafPort to a specified value.

Parameters:
Rgnhandle *rgnHandle*

$C904
GetVisHandle
(RgnHandle)

Returns a copy of the handle to the visible region.

$CA04
InitCursor
(Void)

Reinitializes the cursor. The cursor is set to the arrow cursor and made visible.

$CB04
SetBufDims
(Void)

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. 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*.

Parameters:
Word *maxWidth*
Word *maxFontHeight*
Word *maxFBRExtent*

$CC04
ForceBufDims
(Void)

Sets the size of the QuickDraw II clipping and text buffers. 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*.

Parameters:
Word *maxWidth*
Word *maxFontHeight*
Word *maxFRBExtent*

$CD04
SaveBufDims
(Void)

Saves QuickDraw II's buffer-sizing information in an 8-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.

Parameters:
BufDimRecPtr *sizeInfoPtr*

$CE04
RestoreBufDims
(Void)

Restores QuickDraw II's internal buffers to the sizes described in the 8-byte record created by the SaveBufDims routine. You can use this routine when you want your application to temporarily change (but be able to restore) the size of the QuickDraw II buffers.

Parameters:
BufDimRecPtr *sizeInfoPtr*

$CF04
GetFGSize
(Word)

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.

$D004
SetFontID
(Void)

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:
Long *fontID*

$D104
GetFontID
(Long)

Returns the *fontID* field of the GrafPort.

$D204
SetTextSize
(Void)

Sets the *txSize* field of the GrafPort to a specified value.

Parameters:
Integer *textSize*

$D304
GetTextSize
(Integer)

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 routine.

$D404
SetCharExtra
(Void)

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.

Parameters:
Fixed *charExtra*

$D504
GetCharExtra
(Fixed)

Returns the *chExtra* field from the GrafPort.

$D604
PPToPort
(Void)

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.
Errors: $0420

Parameters:
LocInfoPtr *srcLocPtr*
RectPtr *srcRectPtr*
Integer *destX*
Integer *destY*
Word *transferMode*

$D704
InflateTextBuffer
(Void)

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:
Word *newWidth*
Word *newHeight*

$D804
GetROMFont
(Void)

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

Parameters:
RomFontRecPtr *recordPtr*

$D904
GetFontLore
(Word)

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, 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.

Parameters:
FontGlobalsRecPtr *recordPtr*
Word *recordSize*

$0212
QDAuxStartUp
(Void)

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

$0312
QDAuxShutDown
(Void)

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

$0412
QDAuxVersion
(Word)

Returns the version number of QuickDraw II Auxiliary.

$0612
QDAuxStatus
(Boolean)

Indicates whether QuickDraw II Auxiliary is active.

$0912
CopyPixels
(Void)

Copies a pixel image from one place to another, stretching or compressing it as necessary to make the source pixels fit the destination rectangle.

Parameters:
LocInfoPtr *srcLocPtr*
LocInfoPtr *destLocPtr*
RectPrt *srcRect*
RectPrt *destRect*
word *xferMode*
RgnHandle *maskRgn*

$0A12
WaitCursor
(Void)

Changes the cursor to a predefined cursor that looks like a watch. You can restore the standard arrow cursor by making the QuickDraw II call InitCursor.

$0B12
DrawIcon
(Void)

Draws a specified icon in a specified mode at a specified location and clips to the current visible and clipping regions. This routine does not contribute to a picture definition, nor does it print the icon.

Parameters:
Pointer *iconPtr*
Word *displayMode*
Word *xPos*
Word *yPos*

$0C12
SpecialRect
(Void)

Frames and fills a rectangle in a single call, making separate calls to FrameRect and FillRect unnecessary. The single call to SpecialRect is considerably faster than separate calls to FrameRect and FillRect.
Errors: $0001: Tool not found.

Parameters:
RectPtr *rectPtr*
Word *frameColor*
Word *fillColor*

$0D12
SeedFill
(Void)

Given a point in a bitmap, SeedFill fills the bitmap with the specified pattern. The pattern fills all blank areas of the bitmap, and stops at the edge of any painted areas. for example, if the call is given a seed in the blank middle of a black circle, it will fill the interior of the circle, and will not fill outside the circle unless there is a "hole" through which the fill can leak.

Parameters:
LocInforPtr *srcLocInfoPtr*
RectPtr *srcRect*
LocInfoPtr *dstLocInfoPtr*
RectPtr *dstRect*
Word *seedH*
Word *seedV*
Word *resMode*
Long *patternPtr*
Long *leakTblPtr*

$0E12
CalcMask
(Void)

Given a pixel image, CalcMask fills a specified color or colors with a specified mask pattern. The leak table specified by leakTblPtr determines the color to be filled. The pattern is specified by patternPtr.

Parameters:
LocInforPtr *srcLocInfoPtr*
RectPtr *srcRect*
LocInfoPtr *dstLocInfoPtr*
RectPtr *dstRect*
Word *resMode*
Long *patternPtr*
Long *leakTblPtr*

$020A
SANEStartUp
(Void)

Starts up the SANE Tool Set. 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,* Second Edition). If you are using assembly language, your application must make this call before making any other SANE Tool Set calls. If you are using APW C, this call is made automatically.

Parameters:
Word *dPageAddr*

$030A
SANEShutDown
(Void)

Shuts down the SANE Tool Set. If your application has started up the SANE Tool Set, the application must make this call before it quits.

$040A
SANEVersion
(Word)

Returns the version number of the SANE Tool Set.

$060A
SANEStatus
(Boolean)

Indicates whether the SANE Tool Set is active.

$090A
SANEFP816
(Void)

Contains basic arithmetic operations, comparisons, conversions, environmental control, and IEEE auxiliary operations. See the *Apple Numerics Manual,* Second Edition, for details.

$0A0A
SANEDecStr816
(Void)

Contains numeric scanners and formatter. See the *Apple Numerics Manual,* Second Edition, for details.

$0B0A
SANEElems816
(Void)

Contains elementary functions, financial functions, and a random-number generator. See the *Apple Numerics Manual,* Second Edition, for details.

$0207
SchStartUp
(Void)

Starts up the Scheduler. Your application must make this call before it makes any other Scheduler calls.

$0307
SchShutDown
(Void)

Shuts down the Scheduler. If your application has started up the Scheduler, the application must make this call before it quits.

$0407
SchVersion
(Word)

Returns the version number of the Scheduler.

$0607
SchStatus
(Boolean)

Indicates whether the Scheduler is active.

$0907
SchAddTask
(Boolean)

Adds a task to the Scheduler's queue. The queue has space for four items, enough to support the Desk Manager as well as other small interrupt handlers. The Scheduler will use 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.

Parameters:
VoidProcPtr *taskPtr*

$0216
ScrapStartUp
(Void)

Starts up the Scrap Manager. Your application must make this call before it makes any other Scrap Manager calls.

$0316
ScrapShutDown
(Void)

Shuts down the Scrap Manager. If your application has started up the Scrap Manager, the application must make this call before it quits.

$0416
ScrapVersion
(Word)

Returns the version number of the Scrap Manager.

$0616
ScrapStatus
(Boolean)

Indicates whether the Scrap Manager is active.

$0916
UnloadScrap
(Void)

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.

$0A16
LoadScrap
(Void)

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; it's just as if you had loaded an empty Clipboard file.

$0B16
ZeroScrap
(Void)

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.

$0C16
PutScrap
(Void)

Appends specified data to the scrap that has the same type as the data. If the scrap is on disk, the scrap is loaded. 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.

Parameters:
unsigned Longint *numBytes*
Word *scrapType*
Pointer *srcPtr*

$0D16
GetScrap
(Void)

Copies scrap information of the appropriate type to a specified handle, setting the handle to the correct size. To copy the desk scrap to the LineEdit scrap, use the LineEdit routine LEFromScrap.
Errors: $1610

Parameters:
Handle *destHandle*
Word *scrapType*

$0E16
GetScrapHandle
(Handle)

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 in situations where memory is in short supply.
Errors: $1610

Parameters:
Word *scrapType*

$0F16
GetScrapSize
(unsigned Longint)

Returns the size of a specified scrap.
Errors: $1610

Parameters:
Word *scrapType*

$1016
GetScrapPath
(Pointer)

Returns a pointer to the pathname used for the Clipboard file.

$1116
SetScrapPath
(Void)

Sets a pointer to the pathname used for the Clipboard file.

Parameters:
Pointer *pathPtr*

$1216
GetScrapCount
(unsigned Integer)

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 may be useful if your application supports display of the Clipboard or has a private scrap.

$1316
GetScrapState
(Word)

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.

$0208
SoundStartUp
(Void)

Starts up the Sound Tool Set. The direct page must be page-aligned and locked until the SoundShutDown call is made. Your application must make this call before it makes any other Sound Tool Set calls.

Parameters:
Word *dPageAddr*

$0308
SoundShutDown
(Void)

Shuts down the Sound Tool Set. If your application has started up the Sound Tool Set, the application must make this call before it quits.

$0408
SoundVersion
(Word)

Returns the version number of the Sound Tool Set.

$0608
SoundToolStatus
(Boolean)

Indicates whether the Sound Tool Set is active.

$0908
WriteRamBlock
(Void)

Writes a specified number of bytes from system RAM into DOC RAM. 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.

Parameters:
Pointer *srcPtr*
Word *docStart*
Word *byteCount*

$0A08
ReadRamBlock
(Void)

Reads a specified number of bytes from DOC RAM into system RAM. 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.

Parameters:
Pointer *destPtr*
Word *docStart*
Word *byteCount*

$0B08
GetTableAddress
(Pointer)

Returns the jump-table address. The jump table provides access to the low-level routines, conversion from generator number to oscillator number, conversion back to generator number, and a pointer to the first generator control block corresponding to a generator.

$0C08
GetSoundVolume
(Word)

Reads the volume setting for a generator. The range of possible values is from $00 to $FF. All 8 bits are valid for DOC volume registers.

Parameters:
Word *genNumber*

$0D08
SetSoundVolume
(Void)

Changes the volume setting for the volume registers in the DOC or changes the system volume. If *genNumber* is specified as $00–$0E, the call sets the volume on the corresponding pair of generators in the DOC. If *genNumber* is specified as $0F or greater, the call sets the system volume control. The range of values for the volume setting is $00–$FF. The generator volume registers use all 8 bits of resolution. The system volume control uses only the upper nibble to determine the setting.

Parameters:
Word *volume*
Word *genNumber*

$0E08
FFStartSound
(Void)

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:
Word *genNumFFSynth*
Pointer *pBlockPtr*

$0F08
FFStopSound
(Void)

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).

Parameters:
Word *genMask*

$1008
FFSoundStatus
(Word)

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.

$1108
FFGeneratorStatus
(Word)

Reads the first 2 bytes of the generator control block that corresponds to a specified generator.

Parameters:
Word *genNumber*

$1208
SetSoundMIRQV
(Void)

Sets up the entry point into the sound interrupt handler. This routine is accessed every time an interrupt is generated by the DOC.

Parameters:
Long *sMasterIRQ*

$1308
SetUserSoundIRQV
(Pointer)

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 and the synthesizer mode is not the user's, then control is passed farther down the chain through this vector. Control is passed through a JSL instruction; therefore, the application must return control through an RTL instruction.

Parameters:
Long *userIRQVector*

$1408
FFSoundDoneStatus
(Boolean)

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:
Word *genNumber*

$0217
SFStartUp
(Void)

Starts up the Standard File Operations Tool Set.

Parameters:
Word *userID*
Word *dPageAddr*

$0317
SFShutDown
(Void)

Shuts down the Standard File Operations Tool Set. If your application has started up the Standard File Operations Tool Set, the application must make this call before it quits.

$0417
SFVersion
(Word)

Returns the version number of the Standard File Operations Tool Set.

$0617
SFStatus
(Boolean)

Indicates whether the Standard File Operations Tool Set is active. SFStatus returns TRUE if SFStartUp has been called and FALSE if it has not.

$0917
SFGetFile
(Void)

Displays the standard Open File dialog box and returns certain information about the user's selection. Specifically, the call returns a reply record indicating whether a file was selected or the dialog box was canceled; and if a file was selected, it contains the ProDOS file type, the auxiliary file type, the name of the file in prefix 0, and the full pathname in prefix 0.

Parameters:
Integer *whereX*
Integer *whereY*
Pointer *promptPtr*
WordProcPtr *filterProcPtr*
Pointer *typeListPtr*
SFReplyRecPtr *replyPtr*

$0A17
SFPutFile
(Void)

Displays the standard Save File dialog box and returns certain information about the user's choice. Specifically, the call returns a reply record indicating whether a save operation was selected or the dialog box was canceled; and if a save was selected, it contains the ProDOS file type, the auxiliary file type, the name of the file in prefix 0, and the full pathname in prefix 0.

Parameters:
Integer *whereX*
Integer *whereY*
Pointer *promptPtr*
Pointer *origNamePtr*
unsigned Integer *maxLen*
SFReplyRecPtr *replyPtr*

$0B17
SFPGetFile
(Void)

Displays a custom Open File dialog box and returns certain information about the user's selection. The call works exactly like the SFGetFile call, except that two parameters—*dlogTempPtr* and *dialogHookPtr*—are added, allowing the programmer to specify custom dialogs.

Parameters:
Integer *whereX*
Integer *wherey*
Pointer *promptPtr*
WordProcPtr *filterProcPtr*
Pointer *typeListPtr*
DlgTempPtr *dlgTempPtr*
VoidProcPtr *dialogHookPtr*
SFReplyRecPtr *replyPtr*

$0C17
SFPPutFile
(Void)

Displays a custom Open File dialog box and returns certain information about the user's selection. The call works exactly like the SFPutFile call, except that two parameters—*dlogTempPtr* and *dialogHookPtr*—are added, allowing the programmer to specify custom dialogs.

Parameters:
Integer *whereX*
Integer *whereY*
Pointer *promptPtr*
Pointer *origNamePtr*
unsigned Integer *maxLen*
DlgTempPtr *dlgTempPtr*
VoidProcPtr *dialogHookPtr*
SFReplyRecPtr *replyPtr*

$0D17
SFAllCaps
(Void)

Allows an application to determine whether filenames will be displayed in all-uppercase letters or in a mix of uppercase and lowercase. If *allCapsFlag* is FALSE, then the first letter of the filename will be uppercase and all subsequent letters will be lowercase, unless the filename contains a period, in which case the first letter after the period will be uppercase.

Parameters:
Boolean *allCapsFlag*

$0411
LoaderVersion
(Word)

Returns the version number of the System Loader.

$0611
LoaderStatus
(Boolean)

Indicates whether the System Loader has been initialized.

$0911
InitialLoad
(InitialLoadOutputRec)

A controlling program calls this routine to request an initial load of the program.
Errors: $1104 $1105 $1109 $110A $110B

Parameters:
Word *userID*
Pointer *loadFileNamePtr*
Boolean *spMemFlag*

$0A11
Restart
(RestartOutRec)

Restarts an application that has been shut down but is still in memory.
Errors: $1101 $1105 $1108

Parameters:
Word *userID*

$0B11
LoadSegNum
(Pointer)

Loads a segment specified by load-file number, load-segment number, and user ID into memory.
Errors: $1101 $1102 $1104 $1105 $1107 $1109 $110A $110B

Parameters:
Word *userID*
Word *loadFileNum*
Word *loadSegNum*

$0C11
UnloadSegNum
(Void)

Unloads a segment specified by load-file number, load-segment number, and user ID from memory.
Errors: $1101 $1105

Parameters:
Word *userID*
Word *loadFileNum*
word *loadSegNum*

$0D11
LoadSegName
(LoadSegNameOut)

Loads a segment specified by its load file's pathname and its segment name into memory.
Errors: $1101 $1104 $1105 $1107 $1109 $110A $110B

Parameters:
Word *userID*
Pointer *loadFileNamePtr*
Pointer *loadSegNamePtr*

$0E11
UnloadSeg
(UnloadSegOutRec)

Unloads the segment containing a specified address.
Errors: $1101 $1105

Parameters:
Pointer *segmentPtr*

$0F11
GetLoadSegInfo
(Void)

Returns the Memory Segment Table Entry corresponding to the specified load-segment number.
Errors: $1101 $1105

Parameters:
Word *userID*
Word *loadFileNum*
Word *loadSegNum*
Pointer *bufferPtr*

$1011
GetUserID
(Word)

Returns the user ID associated with a specified pathname.
Errors: $1101 $1105

Parameters:
Pointer *pathNamePtr*

$1111
GetPathname
(Pointer)

Returns the pathname associated with the specified user ID.
Errors: $1101 $1105

Parameters:
Word *userID*
Word *fileNumber*

$1211
UserShutDown
(Word)

A controlling program calls this routine to shut down a program that has been terminated.
Errors: $1105

Parameters:
Word *userID*
Word *restartFlag*

$020C
TextStartUp
(Void)

Starts up the Text Tool Set. Your application must make this call before it makes any other Text Tool Set calls.

$030C
TextShutDown
(Void)

Shuts down the Text Tool Set. If your application has started up the Text Tool Set, the application must make this call before it quits.

$040C
TextVersion
(Word)

Returns the version number of the Text Tool Set.

$060C
TextStatus
(Boolean)

Indicates whether the Text Tool Set is active.

$090C
SetInGlobals
(Void)

Sets the global parameters for the input device.

Parameters:
Word *andMask*
Word *orMask*

$0A0C
SetOutGlobals
(Void)

Sets the global parameters for the error output device.

Parameters:
Word *andMask*
word *orMask*

$0B0C
SetErrGlobals
(Void)

Sets the global parameters for the error output device.

Parameters:
Word *andMask*
word *orMask*

$0C0C
GetInGlobals
(TxtMaskRec)

Returns the current values for the input device global parameters.

$0D0C
GetOutGlobals
(TxtMaskRec)

Returns the current values for the output device global parameters.

$0E0C
GetErrGlobals
(TxtMaskRec)

Returns the current values for the error output device global parameters.

$0F0C
SetInputDevice
(Void)

Sets the input device to a specified type and location. The routine returns an error if the *deviceType* specified is greater than 2.
Errors: $0C01

Parameters:
Word *deviceType*
Long *ptrOrSlot*

$100C
SetOutputDevice
(Void)

Sets the output device to a specified type and location. The routine returns an error if the *deviceType* specified is greater than 2.
Errors: $0C01

Parameters:
Word *deviceType*
Long *ptrOrSlot*

$110C
SetErrorDevice
(Void)

Sets the error output device to a specified type and location. The routine returns an error if the *deviceType* specified is greater than 2.
Errors: $0C01

Parameters:
Word *deviceType*
Long *ptrOrSlot*

$120C
GetInputDevice
(DeviceRec)

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.

$130C
GetOutputDevice
(DeviceRec)

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.

$140C
GetErrorDevice
(DeviceRec)

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.

$150C
InitTextDev
(Void)

Initializes a specified text device.
Errors: $0C01

Parameters:
Word *deviceNum*

$160C
CtlTextDev
(Void)

Passes a control code to a specified text device. The control codes passed depend on the device used.
Errors: $0C01 $0C02 $0C03 $0C04 $0C05 $0C06 $0C07 $0C08 $0C09 $0C0A $0C0B $0C0C $0C0D $0C0E $0C0F $0C10 $0C40

Parameters:
Word *deviceNum*
Word *controlCode*

$170C
StatusTextDev
(Void)

Executes a status call to a specified text device. The routine returns an error if the device is not ready.
Errors: $0C01 $0C02 $0C03 $0C04 $0C05 $0C06 $0C07 $0C08 $0C09 $0C0A $0C0B $0C0C $0C0D $0C0E $0C0F $0C10 $0C40

Parameters:
Word *deviceNum*
Word *requestCode*

$180C
WriteChar
(Void)

Combines a specified character with the output global AND mask and OR mask and writes the character to the output text device.
Errors: $0C01 $0C02 $0C03 $0C04 $0C05 $0C06 $0C07 $0C08 $0C09 $0C0A $0C0B $0C0C $0C0D $0C0E $0C0F $0C10 $0C40

Parameters:
Word *theChar*

$190C
ErrWriteChar
(Void)

Combines a specified character with the output global AND mask and OR mask and writes the character to the error output text device.
Errors: $0C01 $0C02 $0C03 $0C04 $0C05 $0C06 $0C07 $0C08 $0C09 $0C0A $0C0B $0C0C $0C0D $0C0E $0C0F $0C10 $0C40

Parameters:
Word *theChar*

$1A0C
WriteLine
(Void)

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.
Errors: $0C01 $0C02 $0C03 $0C04 $0C05 $0C06 $0C07 $0C08 $0C09 $0C0A $0C0B $0C0C $0C0D $0C0E $0C0F $0C10 $0C40

Parameters:
Pointer *strPtr*

$1B0C
ErrWriteLine
(Void)

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.
Errors: $0C01 $0C02 $0C03 $0C04 $0C05 $0C06 $0C07 $0C08 $0C09 $0C0A $0C0B $0C0C $0C0D $0C0E $0C0F $0C10 $0C40

Parameters:
Pointer *strPtr*

$1C0C
WriteString
(Void)

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.
Errors: $0C01 $0C02 $0C03 $0C04 $0C05 $0C06 $0C07 $0C08 $0C09 $0C0A $0C0B $0C0C $0C0D $0C0E $0C0F $0C10 $0C40

Parameters:
Pointer *strPtr*

$1D0C
ErrWriteString
(Void)

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.
Errors: $0C01 $0C02 $0C03 $0C04 $0C05 $0C06 $0C07 $0C08 $0C09 $0C0A $0C0B $0C0C $0C0D $0C0E $0C0F $0C10 $0C40

Parameters:
Pointer *strPtr*

$1E0C
TextWriteBlock
(Void)

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 *textPtr* + *offset* parameters, with the length specified by the count parameter.
Errors: $0C01 $0C02 $0C03 $0C04 $0C05 $0C06 $0C07 $0C08 $0C09 $0C0A $0C0B $0C0C $0C0D $0C0E $0C0F $0C10 $0C40

Parameters:
Pointer *textPtr*
Word *offset*
Word *count*

$1F0C
ErrWriteBlock
(Void)

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 *textPtr* + *offset* parameters, with the length specified by the count parameter.
Errors: $0C01 $0C02 $0C03 $0C04 $0C05 $0C06 $0C07 $0C08 $0C09 $0C0A $0C0B $0C0C $0C0D $0C0E $0C0F $0C10 $0C40

Parameters:
Pointer *textPtr*
Word *offset*
Word *count*

$200C
WriteCString
(Void)

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.
Errors: $0C01 $0C02 $0C03 $0C04 $0C05 $0C06 $0C07 $0C08 $0C09 $0C0A $0C0B $0C0C $0C0D $0C0E $0C0F $0C10 $0C40

Parameters:
Pointer *cStrPtr*

$210C
ErrWriteCString
(Void)

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.
Errors: $0C01 $0C02 $0C03 $0C04 $0C05 $0C06 $0C07 $0C08 $0C09 $0C0A $0C0B $0C0C $0C0D $0C0E $0C0F $0C10 $0C40

Parameters:
Pointer *cStrPtr*

$220C
ReadChar
(Word)

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.
Errors: $0C01 $0C02 $0C03 $0C04 $0C05 $0C06 $0C07 $0C08 $0C09 $0C0A $0C0B $0C0C $0C0D $0C0E $0C0F $0C10 $0C40

Parameters:
Word *echoFlag*

$230C
TextReadBlock
(Void)

Reads a block of characters from the input text device, combines the block 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.
Errors: $0C01 $0C02 $0C03 $0C04 $0C05 $0C06 $0C07 $0C08 $0C09 $0C0A $0C0B $0C0C $0C0D $0C0E $0C0F $0C10 $0C40

Parameters:
Pointer *bufferPtr*
Word *offset*
Word *blockSize*
Word *echoFlag*

$240C
ReadLine
(Word)

Reads a character string from the input text device, combines the string with the input global masks, and writes the string to the memory location starting at *bufferPtr*. The character string is terminated by an EOL character or a character count equal to the maximum line length. If *echoFlag* is set to a value of $0001, the characters are also written to the output device. The count of characters received is returned on the stack.
Errors: $0C01 $0C02 $0C03 $0C04 $0C05 $0C06 $0C07 $0C08 $0C09 $0C0A $0C0B $0C0C $0C0D $0C0E $0C0F $0C10 $0C40

Parameters:
Pointer *bufferPtr*
Word *maxCount*
Word *eolChar*
Word *echoFlag*

$0201
TLStartUp
(Void)

Starts up the Tool Locator. Your application must make this call before making any other tool calls.

$0301
TLShutDown
(Void)

Shuts down the Tool Locator. If your application has started up the Tool Locator, the application must make this call before it quits.

$0401
TLVersion
(Word)

Returns the version number of the Tool Locator.

$0601
TLStatus
(Boolean)

Indicates whether the Tool Locator is active. Always returns TRUE: The Tool Locator is always active.

$0901
GetTSPtr
(Pointer)

Returns the pointer to the function pointer table (FPT) of a specified tool set. This call is normally used only if you are writing your own tool set.
Errors: $0001

Parameters:
Word *userOrSystem*
Word *tSNum*

$0A01
SetTSPtr
(Void)

Installs the pointer to a function pointer table (FPT) in the appropriate tool pointer table (TPT). This call is normally used only if you are writing your own tool set.
Errors: $0001

Parameters:
Word *userOrSystem*
word *tsNum*
Pointer *fptPtr*

$0B01
GetFuncPtr
(Pointer)

Returns an entry in the function pointer table (FPT) for a specified function in a specified tool set.
Errors: $0001 $0002

Parameters:
Word *userOrSystem*
Word *funcTSNum*

$0C01
GetWAP
(Pointer)

Gets the pointer to the work area for a specified tool set. This call is normally used only if you are writing your own tool set.
Errors: $0001

Parameters:
Word *userOrSystem*
Word *tSNum*

$0D01
SetWAP
(Void)

Sets the pointer to the work area for a specified tool set. This call is normally used only if you are writing your own tool set.
Errors: $0001

Parameters:
Word *userOrSystem*
Word *tSNum*
Pointer *waptPtr*

$0E01
LoadTools
(Void)

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 containing the total number of tool sets, the number of each tool set needed, and the minimum acceptable version number of each tool set. RAM-based tools are loaded from the TOOLS subdirectory of the SYSTEM directory. Each tool set is a load file of type $BA and is named after its decimal tool set number (Tool 23 is in a file named TOOL023, and so on).
Errors: $0001 $0110

Parameters:
Pointer *toolTablePtr*

$0F01
LoadOneTool
(Void)

Ensures that a specified 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.
Errors: $0001 $0110

Parameters:
Word *toolNumber*
Word *minVersion*

$1001
UnloadOneTool
(Void)

Unloads a specified tool set from memory. The tool set is left in a restartable state; that is, the purge level of all the tool set's memory blocks is set to 3, so that the tool set may be quickly restarted.
Errors: $0001

Parameters:
Word *toolNumber*

$1101
TLMountVolume
(Word)

Displays a simulated dialog box on the Super Hi-Res display that your application can use to ask the user to mount a volume.

Parameters:
Integer *whereX*
Integer *whereY*
Pointer *line1Ptr*
Pointer *line2Ptr*
Pointer *but1Ptr*
Pointer *but2Ptr*

$1201
TLTextMountVolume
(Word)

Displays a simulated dialog box on the 40-column text screen that your application can use to ask the user to mount a volume.

Parameters:
Pointer *line1Ptr*
Pointer *line2Ptr*
Pointer *but1Ptr*
Pointer *but2Ptr*

$1301
SaveTextState
(Handle)

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.

$1401
RestoreTextState
(Void)

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.

Parameters:
Handle *stateHandle*

$1501
MessageCenter
(Void)

Allows applications to communicate with each other by supplying handles to messages. Allows an application to add, delete, or retrieve a specified message.
Errors: $0111

Parameters:
Word *action*
Word *type*
Handle *messageHandle*

$1601
SetDefaultTPT
(Void)

Sets the default tool pointer table (TPT) to the current TPT. Used to permanently install a tool patch.

Parameters:
Pointer *tptPtr*

$020E
WindStartUp
(Void)

Starts up the Window Manager. Your application must make this call before it makes any other Window Manager calls.

Parameters:
Word *userID*

$030E
WindShutDown
(Void)

Shuts down the Window Manager. If your application has started up the Window Manager, the application must make this call before it quits.

$040E
WindVersion
(Word)

Returns the version number of the Window Manager.

$060E
WindStatus
(Boolean)

Indicates whether the Window Manager is active.

$090E
NewWindow
(GrafPortPtr)

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. 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.
Errors: $0E01 $0E02

Parameters:
ParamListPtr *paramListPtr*

$0A0E
CheckUpdate
(Boolean)

Looks for a visible window that needs updating. CheckUpdate is normally called only by the Event Manager and doesn't need to be called by an application.

Parameters:
EventRecordPtr *theEventPtr*

$0B0E
CloseWindow
(Void)

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 finished with a window.

Parameters:
GrafPortPtr *theWindowPtr*

$0C0E
Desktop
(Pointer)

Controls the addition of regions to and subtraction of regions from the desktop and controls the current desktop pattern.

Parameters:
Word *deskTopOP*
Long *dtParam*

$0D0E
SetWTitle
(Void)

Changes the title of a specified window to a specifed title and redraws the window. The string pointed to by *titlePtr* must be a Pascal-type string. The string pointed to by *titlePtr* must not be changed or moved.

Parameters:
Pointer *titlePtr*
GrafPortPtr *theWindowPtr*

$0E0E
GetWTitle
(Pointer)

Returns the pointer to a specified window's title. The string pointed to by *titlePtr* is a Pascal-type string.

Parameters:
GrafPortPtr *theWindowPtr*

$0F0E
SetFrameColor
(Void)

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.

Parameters:
WindColorPtr *newColorPtr*
GrafPortPtr *theWindowPtr*

$100E
GetFrameColor
(WindColorPtr)

Returns the color of a specified window's frame.

Parameters:
GrafPortPtr *theWindowPtr*

$110E
SelectWindow
(Void)

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:
GrafPortPtr *theWindowPtr*

$120E
HideWindow
(Void)

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:
GrafPortPtr *theWindowPtr*

$130E
ShowWindow
(Void)

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.

Parameters:
GrafPortPtr *theWindowPtr*

$140E
SendBehind
(Void)

Changes the position of a specified window, redrawing any exposed windows. If *behindWindowPtr* is –2 ($FFFFFFFE), it sends the specified window behind all other windows. If *behindWindowPtr* is –1 ($FFFFFFFF), it puts the specified window in front of all other windows. If the specified window is the active window, the routine unhighlights the active window, highlights the new active window, and generates the appropriate activate events.

Parameters:
GrafPortPtr *behindWindowPtr*
GrafPortPtr *theWindowPtr*

$150E
FrontWindow
(GrafPortPtr)

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.

$160E
SetInfoDraw
(Void)

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.

Parameters:
VoidProcPtr *infoRecCon*
GrafPortPtr *theWindowPtr*

$170E
FindWindow
(Word)

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.

Parameters:
GrafPortHndl *whichWindowPtr*
Integer *pointX*
Integer *pointY*

$180E
TrackGoAway
(Boolean)

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. 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 (after 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:
Integer *startX*
Integer *startY*
GrafPortPtr *theWindowPtr*

$190E
MoveWindow
(Void)

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:
Integer *newX*
Integer *newY*
GrafPortPtr *theWindowPtr*

$1A0E
DragWindow
(Void)

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.

Parameters:
Word *grid*
Integer *startX*
Integer *startY*
Word *grace*
RectPtr *boundsRectPtr*
GrafPortPtr *theWindowPtr*

$1B0E
GrowWindow
(Long)

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. GrowWindow does not actually resize the window. For that, use SizeWindow.

Parameters:
Word *minWidth*
Word *minHeight*
Integer *startX*
Integer *startY*
GrafPortPtr *theWindowPtr*

$1C0E
SizeWindow
(Void)

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:
Word *newWidth*
word *newHeight*
GrafPortPtr *theWindowPtr*

$1D0E
TaskMaster
(Word)

When passed a pointer to a task record, TaskMaster 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*.
Errors: $0E03

Parameters:
Word *taskMask*
WmTaskRecPtr *taskRecPtr*

$1E0E
BeginUpdate
(Void)

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. Every call to BeginUpdate must be balanced by a call to the EndUpdate routine.

Parameters:
GrafPortPtr *theWindowPtr*

$1F0E
EndUpdate
(Void)

Restores the normal visible region of a specifed window's GrafPort that was changed by a BeginUpdate call.

Parameters:
GrafPortPtr *theWindowPtr*

$200E
GetWMgrPort
(GrafPortPtr)

Returns a pointer to the Window Manager's port.

$210E
PinRect
(Long)

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.)

Parameters:
Integer *theXPt*
Integer *theYPt*
RectPtr *theRectPtr*

$220E
HiliteWindow
(Void)

Highlights or unhighlights a specified window, depending on the value of a specified parameter. If *fHiliteFlag* is TRUE, this routine highlights the window. If *fHiliteFlag* is FALSE, HiliteWindow unhighlights the window. The exact way a window is highlighted and unhighlighted depends on its window-definition procedure.

Parameters:
Boolean *fHiliteFlag*
GrafPortPtr *theWindowPtr*

$230E
ShowHide
(Void)

Shows or hides a window. If *showFlag* is TRUE, ShowHide makes the specified window visible if it's not already visible and has no effect if it is already visible. If *showFlag* is FALSE, ShowHide makes the window invisible if it's not already invisible and has no effect if it is already invisible. ShowHide does not generate activate events or change the highlighting of any windows.

Parameters:
Boolean *showFlag*
GrafPortPtr *theWindowPtr*

$240E
BringToFront
(Void)

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 should call SelectWindow to make a window active; SelectWindow takes care of bringing the window to the front.

Parameters:
GrafPortPtr *theWindowPtr*

$250E
WindNewRes
(Void)

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.

$260E
TrackZoom
(Boolean)

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. If the mouse button is released inside the zoom region, TrackZoom unhighlights the zoom region and returns TRUE (after 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:
Integer *startX*
Integer *startY*
GrafPortPtr *theWindowPtr*

$270E
ZoomWindow
(Void)

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 ZoomWindow was performed. When a SizeWindow or MoveWindow is performed while a window is zoomed, the new size and position become the unzoomed values.

Parameters:
GrafPortPtr *theWindowPtr*

$280E
SetWRefCon
(Void)

Sets a value that is inside a specified window record and is reserved for the application's use.

Parameters:
Longint *wRefCon*
GrafPortPtr *theWindowPtr*

$290E
GetWRefCon
(Long)

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. The *wRefCon* parameter is used to define the contents of a window's content region.

Parameters:
GrafPortPtr *theWindowPtr*

$2A0E
GetNextWindow
(GrafPortPtr)

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:
GrafPortPtr *theWindowPtr*

$2B0E
GetWKind
(Word)

Indicates whether a specified window is a system window or an application window.

Parameters:
GrafPortPtr *theWindowPtr*

$2C0E
GetWFrame
(Word)

Returns the bit flag that describes a specified window's frame type.

Parameters:
GrafPortPtr *theWindowPtr*

$2D0E
SetWFrame
(Void)

Sets the bit flag that describes a specified window's frame type. The window frame is not redrawn. Normally, you won't need to call this routine; instead, you should set up the window frame correctly with the NewWindow routine. The SetWFrame routine is provided for custom window-definition procedures.

Parameters:
Word *wFrame*
GrafPortPtr *theWindowPtr*

$2E0E
GetStructRgn
(RgnHandle)

Returns a handle to a specified window's structure region.

Parameters:
GrafPortPtr *theWindowPtr*

$2F0E
GetContentRgn
(RgnHandle)

Returns a handle to a specified window's content region.

Parameters:
GrafPortPtr *theWindowPtr*

$300E
GetUpdateRgn
(RgnHandle)

Returns a handle to a specified window's update region.

Parameters:
GrafPortPtr *theWindowPtr*

$310E
GetDefProc
(LongProcPtr)

Returns a pointer to the routine that is called to draw, hit test, and otherwise define a window's frame and behavior.

Parameters:
GrafPortPtr *theWindowPtr*

$320E
SetDefProc
(Void)

Sets the pointer to the routine that defines a window's frame and behavior.

Parameters:
LongProcPtr *wDefProcPtr*
GrafPortPtr *theWindowPtr*

$330E
GetWControls
(CtlRecHndl)

Returns a handle to the window's control list. The control list is created when the application calls NewControl to create controls for the window.

Parameters:
GrafPortPtr *theWindowPtr*

$340E
SetOriginMask
(Void)

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. SetOriginMask provides 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:
Word *originMask*
GrafPortPtr *theWindowPtr*

$350E
GetInfoRefCon
(Long)

Returns the value of a specified window's *wInfoRefCon* field (the value associated with the draw information bar draw routine). The field is reserved for application use.

Parameters:
GrafPortPtr *theWindowPtr*

$360E
SetInfoRefCon
(Void)

Sets the value associated with the draw information bar routine for a specified window. That value is reserved for use by the applicaton.

Parameters:
Long *infoRefCon*
GrafPortPtr *theWindowPtr*

$370E
GetZoomRect
(RectPtr)

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, then *wZoomSizePtr* points to a RECT data structure that contains the window's last size and position. Otherwise, *wZoomSizePtr* points to a RECT data structure that contains the size and position of the window's content region when the window is zoomed by a call to ZoomWindow.

Parameters:
GrafPortPtr *theWindowPtr*

$380E
SetZoomRect
(Void)

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 is set to 1 in the wFrame flag—*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, *wZoomSizePtr* should point to a RECT data structure that specifies the zoomed size and position of the window.

Parameters:
RectPtr *wZoomSizePtr*
GrafPortPtr *theWindowPtr*

$390E
RefreshDesktop
(Void)

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:
RectPtr *redrawRect*

$3A0E
InvalRect
(Void)

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.

Parameters:
RectPtr *badRectPtr*

$3B0E
InvalRgn
(Void)

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.

Parameters:
RgnHandle *badRgnHandle*

$3C0E
ValidRect
(Void)

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.

Parameters:
RectPtr *goodRectPtr*

$3D0E
ValidRgn
(Void)

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.

Parameters:
RgnHandle *goodRgnHandle*

$3E0E
GetContentOrigin
(Long)

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:
Pointer *theWindowPtr*

$3F0E
SetContentOrigin
(Void)

Sets the origin of the window's GafPort when handling an update event. TaskMaster uses the values of ***xOrigin*** and ***yOrigin*** to set the origin of the window's GrafPort, and the Window Manager uses them to compute sizes and positions for scroll bars in the window frame.

Parameters:
Word ***xOrigin***
Word ***yOrigin***
GraPortPtr *theWindowPtr*

$400E
GetDataSize
(Long)

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:
GrafPortPtr *theWindowPtr*

$410E
SetDataSize
(Void)

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:
Word *dataWidth*
Word *dataHeight*
GrafPortPtr *theWindowPtr*

$420E
GetMaxGrow
(Long)

Returns the maximum values to which a specified window's content region can grow.

Parameters:
GrafPortPtr *theWindowPtr*

$430E
SetMaxGrow
(Void)

Sets the maximum values to which a specified window's content region can grow.

Parameters:
Word *maxWidth*
Word *maxHeight*
GrafPortPtr *theWindowPtr*

$440E
GetScroll
(Long)

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:
GrafPortPtr *theWindowPtr*

$450E
SetScroll
(Void)

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:
Word *hScroll*
Word *vScroll*
GrafPortPtr *theWindowPtr*

$460E
GetPage
(Long)

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:
GrafPortPtr *theWindowPtr*

$470E
SetPage
(Void)

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:
Word *hPage*
Word *vPage*
GrafPortPtr *theWindowPtr*

$480E
GetContentDraw
(VoidProcPtr)

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.

Parameters:
GrafPortPtr *theWindowPtr*

$490E
SetContentDraw
(Void)

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.

Parameters:
VoidProcPtr *contentDrawPtr*
GrafPortPtr *theWindowPtr*

$4A0E
GetInfoDraw
(VoidProcPtr)

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.

Parameters:
GrafPortPtr *theWindowPtr*

$4B0E
SetSysWindow
(Void)

Marks a specified window as a system window.

Parameters:
GrafPortPtr *theWindowPtr*

$4C0E
GetSysWFlag
(Boolean)

Indicates whether a specified window is a system window or an application window.

Parameters:
GrafPortPtr *theWindowPtr*

$4D0E
StartDrawing
(Void)

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. Do not call StartDrawing between a BeginUpdate and an EndUpdate call. When you have finished drawing, call the QuickDraw II routine SetOrigin(0,0).

Parameters:
GrafPortPtr *theWindowPtr*

$4E0E
SetWindowIcons
(FontHndl)

Sets the icon font for the Window Manager. If you want to use the call to simply retrieve the handle of the current font, specify a negative value for *newFontHandle.*

Parameters:
FontHndl *newFontHandle*

$4F0E
GetRectInfo
(Void)

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.

Parameters:
RectPtr *infoRectPtr*
GrafPortPtr *theWindowPtr*

$500E
StartInfoDrawing
(Void)

Allows an application to draw or hit test outside its information bar definition procedure. 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.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:
RectPtr *infoRectPtr*
GrafPortPtr *theWindowPtr*

$510E
EndInfoDrawing
(Void)

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. Calling any Window Manager routine between a StartInfoDrawing call and an EndInfoDrawing call may result in system failure.

$520E
GetFirstWindow
(GrafPortPtr)

Returns a pointer to the first window in the Window Manager's window list. The returned window may not be the active window. 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.

$530E
WindDragRect
(Long)

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.

Parameters:
VoidProcPtr *actionProcPtr*
Pattern *dragPatternPtr*
Integer *startX*
Integer *startY*
RectPtr *dragRectPtr*
RectPtr *limitRectPtr*
RectPrt *slopRectPtr*
Word *dragFlag*

$550E
DrawInfoBar
(Void)

Redraws the info bar of the window specified by *grafPortPtr*. The method used to redraw the info bar's interior is the routine specified by the *wInfoDefProc* field of the *paramList* passed to NewWindow when the window is created. The Window Manager will automatically clip the info bar drawing to the dimensions of the info bar and to the visible region of the window.

Parameters:
GrafPortPtr *grafPortPtr*

$560E
WindowGlobal
(Word)

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*. All bits except bit 15 and bit 0 are reserved for future use. The only valid values for the window global mask are used to change the Window Manager's normal highlighting procedure.

Parameters:
Word *windowGlobalMask*

$570E
SetContentOrigin2
(Void)

Sets the origin of the window's GrafPort when handling an update event and allows the application to specify whether scrolling is to be enabled. If *scrollFlag* is 0, then this call is identical to SetContentOrigin. If it is 1, then the window's origin will be set without scrolling the contents. TaskMaster uses the values of *xOrigin* and *yOrigin* to set the origin of the window's GrafPort, and the Window Manager uses them to compute sizes and positions for scroll bars in the window frame.

Parameters:
Word *scrollFlag*
Integer *xOrigin*
Integer *yOrigin*
GrafPortPtr *theWindowPtr*

$580E
GetWindowMgrGlobals
(Pointer)

Returns a pointer to the Window Manager global data area.

$590E
AlertWindow
(Word)

Creates an alert window that displays a message pointed to by *alertStrPtr*. The message can be either a C or a Pascal string, as specified by *stringType*. A value of 0 signifies that the message is a C string, and a value of 1 that it is a Pascal string. *subStrPtr* points to an array of substitution strings for use with substitution characters.

Parameters:
Word *stringType*
Long *subStrPtr*
Long *alertStrPtr*

$5A0E
StartFrameDrawing
(Void)

Sets up to draw a window frame. Should be called only by window-definition procedures. Must be balanced by a call to EndFrameDrawing when drawing is completed.

Parameters:
Pointer *windowPtr*

$5B0E
EndFrameDrawing
(Void)

Restores Window Manager variables after a call to StartFrameDrawing.

$5C0E
ResizeWindow
(Void)

Moves, resizes, and draws the window specified by *grafPortPtr*. *rectPtr* is a pointer to the window's content region. *hiddenFlag* is a Boolean parameter. A TRUE value specifies that portions of the window that are covered should not be drawn. If the value is FALSE, the entire content of the window is drawn.

Parameters:
Word *hiddenFlag*
Pointer *rectPtr*
Pointer *grafPortPtr*

$0001 invalidCallNum	Invalid call number
$0002 funcNotFoundErr	Specified function unavailable
$0004 invalidPcount	Invalid parameter count
$0005 badPBlockPtr	Call pointer out of bounds
$0006 pdosActiveErr	ProDOS active
$0007 proDOSactive	ProDOS already active
$0007 gsosActive	GS/OS already active
$000A vcbUnusable	Fatal error—VCB unusable
$000B fcbUnusable	Fatal error—VCB unusable
$000C badBlockZero	Fatal error—block 0 allocated illegally
$000D shdwInterruptErr	Fatal error—interrupt occurred while I/O shadowing off
$0010 devNotFound	Specified device unavailable
$0011 invalidDevNum	Invalid device number
$0015 segLoader1Err	Segment loader error
$0017 sPackage0Err	Can't load needed package
$0018 package1Err	Can't load needed package
$0019 package2Err	Can't load needed package
$001A package3Err	Can't load needed package
$001B package4Err	Can't load needed package
$001C package5Err	Can't load needed package
$001D package6Err	Can't load needed package
$001E package7Err	Can't load needed package
$0020 badReq	Request or command invalid
$0020 drvrBadReq	Request or command invalid
$0021 badCtrlCode	Control or status code invalid
$0022 drvrBadParm	Call parameter incorrect
$0023 drvrNotOpen	Character device not open
$0024 drvrPriorOpen	Character device already open
$0025 irqTableFull	Interrupt table full
$0026 drvrNoResrc	Needed resources not available
$0027 drvrIOError	I/O error
$0028 drvrNoDevice	Specified device not connected
$0029 drvrBusy	Call aborted; driver busy
$002B drvrWrtProt	Specified device write-protected
$002C drvrBadCount	Byte count invalid
$002D drvrBadBlock	Block address invalid
$002E diskSwitchErr	Disk switched
$002F drvrOffLine	Specified device off-line, or no medium present
$0030 devErr0	Device-specific error
$0031 devErr1	Device-specific error
$0032 devErr2	Device-specific error
$0033 devErr3	Device-specific error

$0034 devErr4	Device-specific error
$0035 devErr5	Device-specific error
$0036 devErr6	Device-specific error
$0037 devErr7	Device-specific error
$0038 devErr8	Device-specific error
$0039 devErr9	Device-specific error
$003A devErrA	Device-specific error
$003B devErrB	Device-specific error
$003C devErrC	Device-specific error
$003D devErrD	Device-specific error
$003E devErrE	Device-specific error
$003F devErrF	Device-specific error
$0040 badPathname	Specified pathname invalid
$0041 memMgr16Err	Memory Manager error occurred
$0042 FCBTabFull	FCB table full
$0043 badFileRefNum	Specified file reference number invalid
$0044 pathNotFound	Specified path does not exist
$0045 volumeNotFound	Specified volume unavailable
$0046 fileNotFound	Specified file not available
$0047 dupPathname	Specified filename already exists
$0048 volumeFull	Specified volume full
$0049 volDirFull	Specified volume's directory full
$0049 dirFullErr	Specified directory full
$004A badFileFormat	File format of specified volume incompatible
$004B badStoreType	Storage type incorrect or unsupported
$004C endOfFile	End-of-file encountered
$004D outOfRange	Specified position out of range
$004E invalidAccess	Access to specified file or block not allowed
$004F buffTooSmall	Buffer too small
$0050 fileBusy	File already open
$0051 dirError	Directory structure damaged
$0052 unknownVol	Volume type unknown
$0053 paramRangeErr	Parameter out of range
$0054 outOfMem	No more memory available
$0055 VCBTabFull	VCB table full
$0057 dupVolume	Specified volume name duplicate of existing name
$0058 notBlockDev	Specified device not block device
$0059 invalidLevel	Specified level outside legal range
$005A damagedBitMap	Block number too large
$005B badPathNames	Specified pathnames invalid for ChangePath

$005C notSystemFile	Specified file not an executable file
$005D osUnsupported	Specified operating system not supported
$005E deallocateRamErr	Can't deallocate RAM
$005F stackOverflow	Too many applications on stack
$0060 dataUnavail	Needed data unavailable
$0061 endOfDir	End of directory reached
$0062 invalidClass	FST call class invalid
$0063 resNotFound	File does not contain required resource
$0080 prAbort	Printing interrupted
$0100 stupVolMntErr	Can't mount system startup volume
$0110 toolVersionErr	Required minimum tool version not available
$0111 messNotFoundErr	Specified message unavailable
$0201 memErr	Unable to allocate block
$0202 emptyErr	Specified operation illegal because handle empty
$0203 notEmptyErr	Empty handle expected for this operation
$0204 lockErr	Illegal operation on locked block
$0205 purgeErr	Attempt to purge unpurgeable block
$0206 handleErr	Invalid handle
$0207 idErr	Invalid owner ID
$0208 attrErr	Specified operation illegal on block with given attributes
$0301 badInputErr	Bad input parameter
$0302 noDevParamErr	No device found for input parameter
$0303 taskInstlErr	Task already installed
$0304 noSigTaskErr	No signature in task header
$0305 queueDmgdErr	Queue damaged
$0306 taskNtFdErr	Task not found
$0307 firmTaskErr	Firmware task unsuccessful
$0308 hbQueueBadErr	Heartbeat queue damaged
$0309 unCnctdDevErr	Attempted to dispatch to unconnected device
$030B idTagNtAvlErr	ID tag not available
$0401 alreadyInitialized	QuickDraw II already initialized
$0402 cannotReset	Never used
$0403 notInitialized	QuickDraw II not initialized
$0410 screenReserved	Screen reserved
$0411 badRect	Specified rectangle invalid
$0420 notEqualChunkiness	Chunkiness not equal

$0430 rgnAlreadyOpen	Specified region already open
$0431 rgnNotOpen	Specified region not open
$0432 rgnScanOverflow	Region scan overflow
$0433 rgnFull	Region full
$0440 polyAlreadyOpen	Polygon already open
$0441 polyNotOpen	Polygon not open
$0442 polyTooBig	Polygon too big
$0450 badTableNum	Table number invalid
$0451 badColorNum	Color number invalid
$0452 badScanLine	Specified scan line invalid
$04FF NotImplemented	Not implemented
$0510 daNotFound	Specified desk accessory not found
$0511 notSysWindow	Specified window not Desk Accessory window
$0601 emDupStrtUpErr	Duplicate EMStartup call
$0602 emResetErr	Can't reset Event Manager
$0603 emNotActErr	Event Manager not active
$0604 emBadEvtCodeErr	Event code invalid
$0605 emBadBttnNoErr	Button number invalid
$0606 emQSiz2LrgErr	Queue size too large
$0607 emNoMemQueueErr	Not enough memory for queue
$0681 emBadEvtQErr	Fatal system error—event queue damaged
$0682 emBadQHndlErr	Fatal system error—queue handle damaged
$0810 noDOCFndErr	No DOC chip found
$0811 docAddrRngErr	DOC address out of range
$0812 noSAppInitErr	No SAppInit call made
$0813 invalGenNumErr	Generator number invalid
$0814 synthModeErr	Specified synthesizer incorrect
$0815 genBusyErr	Generator busy
$0817 mstrIRQNotAssgnErr	Master IRQ not assigned
$0818 sndAlreadyStrtErr	Sound tools already started
$08FF unclaimedSndIntErr	Specified sound interrupt not assigned
$0910 cmndIncomplete	Command not completed
$0911 cantSync	Can't synchronize
$0982 adbBusy	Busy (command pending)
$0983 devNotAtAddr	Specified device not present at given address
$0984 srqListFull	List full
$0B01 imBadInptParam	Input parameter incorrect
$0B02 imIllegalChar	Illegal character in string
$0B03 imOverflow	Value too large for its type
$0B04 imStrOverflow	String too long
$0C01 badDevType	Specified device inappropriate

$0C02 badDevNum	Illegal device number
$0C03 badMode	Bad mode—illegal operation
$0C04 unDefHW	Undefined hardware error
$0C05 lostDev	Lost device—device no longer on-line
$0C06 lostFile	File no longer in disk directory
$0C07 badTitle	Illegal filename
$0C08 noRoom	Insufficient space on specified disk
$0C09 noDevice	Specified volume not on-line
$0C0A noFile	File not in specified directory
$0C0b dupFile	Filename already exists
$0C0C notClosed	Specified file already open
$0C0D notOpen	Specified file already closed
$0C0E badFormat	Real or integer value not in correct format
$0C0F ringBuffOFlo	Characters arriving too fast
$0C10 writeProtected	Specified volume write-protected
$0C40 devErr	Read or write failed
$0E01 paramLenErr	First word of parameter list wrong size
$0E02 allocateErr	Unable to allocate window record
$0E03 taskMaskErr	Bits 12–15 not clear in *wmTaskMask* field of *wmTaskRec*
$1001 wmNotStartedUp	Window Manager not started first
$1101 idNotFound	Segment, application, or entry not found
$1104 idNotLoadFile	Indicated file not load file
$1105 idBusyErr	System Loader busy
$1107 idFilVersErr	Specified file invalid version
$1108 idUserIDErr	User ID error
$1109 idSequenceErr	Segment number out of sequence
$110A idBadRecordErr	Illegal load record found
$110B idForeignSegErr	Specified segment foreign type
$1301 missingDriver	Specified driver not in system/drivers directory
$1302 portNotOn	Specified port not selected in Control Panel
$1303 noPrintRecord	No print record given
$1304 badLaserPrep	Laser Prep in LaserWriter incompatible
$1305 badLPFile	Laser Prep in system/drivers directory incompatible
$1306 papConnNotOpen	Can't connect to LaserWriter
$1307 papReadWriteErr	AppleTalk PAPRead or PAPWrite returned
$1321 startUpAlreadyMade	Low-level startup already made
$1322 invalidCtlVal	Specified control value invalid

$1401 leDupStrtUpErr	Duplicate LEStartup call
$1402 leResetErr	Can't reset LineEdit
$1403 leNotActiveErr	LineEdit not active
$1404 leScrapErr	Desk scrap too big to copy
$150A badItemType	Specified item invalid type
$150B newItemFailed	Creation of item failed
$150C itemNotFound	Specified item not available
$150D notModalDialog	Specified window not modal dialog window
$1610 badScrapType	No scrap of the type
$1901 nsAlreadyInit	Note Synthesizer already initialized
$1902 nsSndNotInit	Sound Tool Set not initialized
$1921 nsNotAvail	Requested generator not available
$1922 nsBadGenNum	Specified generator number invalid
$1923 nsNotInit	Note Synthesizer not initialized
$1924 nsGenAlreadyOn	Specified generator already on
$1A00 noRoomMidiErr	No memory available to execute specified MIDI command
$1A01 noCommandErr	Can't understand current SeqItem
$1A02 noRoomErr	Sequence more than 12 levels deep
$1A03 startedErr	Note Sequencer already started
$1A04 noNoteErr	Can't find note to be turned off by current SeqItem
$1A05 noStartErr	Note Sequencer not started
$1A06 instBndsErr	Instrument number out of range
$1B01 fmDupStartUpErr	Duplicate FMStartUp call
$1B02 fmResetErr	Can't reset Font Manager
$1B03 fmNotActiveErr	Font Manager not active
$1B04 fmFamNotFndErr	Specified font family not available
$1B05 fmFontNtFndErr	Specified font not available
$1B06 fmFontMemErr	Specified font not in memory
$1B07 fmSysFontErr	System font can't be purgeable
$1B08 fmBadFamNumErr	Specified family number invalid
$1B09 fmBadSizeErr	Specified size invalid
$1B0A fmBadNameErr	Font family name illegal length
$1B0C fmMenuErr	Font menu not fixed
$1B0C fmScaleSizeErr	Scaled size of font invalid
$2001 miPacketErr	Incorrect length for received MIDI command
$2002 miArrayErr	Designated array had insufficient or illegal size
$2003 miFullBufErr	Input buffer overflowed

$2004 miToolsErr	Required tools not started up or were invalid versions
$2005 miOutOffErr	MIDI output must first be enabled
$2007 miNoBufErr	No buffer currently allocated
$2008 miDriverErr	Designated file not legal MIDI device driver
$2009 miBadFreqErr	MIDI clock can't attain requested frequency
$200A miClockErr	MIDI clock value wrapped to 0
$200B miConflictErr	Conflicting processes for MIDI input
$200C miNoDevErr	No MIDI device driver loaded
$2080 miDevNotAvail	Requested device not available
$2081 miDevSlotBusy	Requested slot already in use
$2082 miDevBusy	Requested device already in use
$2083 miDevOverrun	Device overrun by incoming MIDI data
$2084 miDevNoConnect	No connection to MIDI
$2085 miDevReadErr	Framing error in received MIDI data
$2086 miDevVersion	ROM version incompatible with device driver
$2087 miDevIntHndlr	Conflicting interrupt handler installed
$FF80 memFullErr	No more memory available
$FFE5 ioAbort	I/O unexpectedly interrupted

Glossary of Primitive Data Types

Byte	8 bits
Word	16 bits (2 bytes)
Long or Longword	32 bits (4 bytes)
Pointer	32 bits (4 bytes)
Handle	32 bits (4 bytes)
LongInt	4 bytes
Integer	2 bytes
Boolean	2 bytes
unsigned Long or unsigned LongInt	4 bytes
Rect	8 bytes (4 integers)
Point	4 bytes
Pattern	32 bytes
Mask	8 bytes
Fixed	4 bytes
TextStyle	2 bytes

AlertTemplate
atBoundsRect / *rect*
atAlertID / *word*
atStage1 / *byte*
atStage2 / *byte*
atStage3 / *byte*
atStage4 / *byte*
atItemList[atItemListLength] / *itemtempptr*

BarColors
barOutline / *word*
barNorArrow / *word*
barSelArrow / *word*
barArrowBack / *word*
barNorThumb / *word*
barSelThumb / *word*
barPageRgn / *word*
barInactive / *word*

BlockRec
blockDevNum / *word*
blockDataBuffer / *ptr*
blockNum / *longint*

BoxColors
boxReserved / *word*
boxNor / *word*
boxSel / *word*
boxTitle / *word*

BttnColors
bttnOutline / *word*
bttnNorBack / *word*
bttnSelBack / *word*
bttnNorText / *word*
bttnSelText / *word*

BufDimRec
maxWidth / *word*
textBufHeight / *word*
textBufferWords / *word*
fontWidth / *word*

ClampRec
yMaxClamp / *word*
yMinClamp / *word*
xMaxClamp / *word*
xMinClamp / *word*

CtlRec
ctlNext / *struct**CtlRec
ctlOwner / *grafportptr*
ctlRect / *rect*
ctlFlag / *byte*
ctlHilite / *byte*
ctlValue / *integer*
ctlProc / *longprocptr*
ctlAction / *longprocptr*
ctlData / *longint*
ctlRefCon / *longint*
ctlColor / *pointer*

DeviceRec
ptrOrSlot / *longword*
deviceType / *word*

DevNumRec
devName / *ptr*
devNum / *word*

DialogTemplate
dtBoundsRect / *rect*
dtVisible / *boolean*
dtRefCon / *longint*
dtItemList[dtItemListLength] / *itemtempptr*

DInfoRec
devNum / *word*
devName / *pointer*

DirEntryRec
refNum / *word*
reserved / *word*
base / *word*
displacement / *word*
nameBuffer / *ptr*
entryNum / *word*
fileType / *word*
endOfFile / *long*
blockCount / *longword*
createTime / *timerec*
modTime / *timerec*
access / *word*
auxType / *longword*
fileSysID / *word*

Envelope
st1BkPt / *byte*
st1Increment / *word*
st2BkPt / *byte*
st2Increment / *word*
st3BkPt / *byte*
st3Increment / *word*
st4BkPt / *byte*
st4Increment / *word*
st5BkPt / *byte*
st5Increment / *word*
st6BkPt / *byte*
st6Increment / *word*
st7BkPt / *byte*
st7Increment / *word*
st8BkPt / *byte*
st8Increment / *word*

EOFRec
eofRefNum / *word*
eofPosition / *longint*

FileIORec
fileRefNum / *word*
dataBuffer / *ptr*
requestCount / *longint*
transferCount / *longint*

FileRec
pathname / *ptr*
fAccess / *word*
fileType / *word*
auxType / *longint*
storageType / *word*
createDate / *word*
createTime / *word*
modDate / *word*
modTime / *word*
blocksUsed / *long*

Font
offseToMF / *word*
family / *word*
style / *textstyle*
size / *word*
version / *word*
fbrExtent / *word*

FontGlobalsRecord
fgFontID / *word*
fgStyle / *textstyle*
fgSize / *word*
fgVersion / *word*
fgWidMax / *word*
fgFBRExtent / *word*

FontID
famNum / *word*
fontStyle / *byte*
fontSize / *byte*

FontInfoRecord
Iascent / *integer*
Idescent / *integer*
IwidMax / *integer*
Ileading / *integer*

GetLInfoPB
sfile / **char*
dfile / **char*
parms / **char*
istring / **char*
merr / *char*
merrf / *char*
lops / *char*
kflag / *char*
long mflags / *unsigned*
long pflags / *unsigned*
long org / *unsigned*

GrafPort
portInfo / *locinfo*
portRect / *rect*
clipRgn / *rgnhandle*
visRgn / *rgnhandle*
bkPat / *pattern*
pnLoc / *point*
pnSize / *point*
pnMode / *word*
pnPat / *pattern*
pnMask / *mask*
pnVis / *word*
fontHandle / *fonthndl*
fontID / *fontid*
fontFlags / *word*
txSize / *integer*
txFace / *textstyle*
txMode / *word*
spExtra / *fixed*
chExtra / *fixed*
fgColor / *word*
bgColor / *word*
picSave / *handle*
rgnSave / *handle*
polySave / *handle*
grafProcs / *qdprocsptr*
arcRot / *integer*
userField / *longint*
sysField / *longint*

InitialLoadOutputRec
userID / *word*
startAddr / *pointer*
dPageAddr / *word*
buffSize / *word*

Instrument
theEnvelope / *envelope*
releaseSegment / *byte*
priorityIncrement / *byte*
pitchBendRange / *byte*
vibratoDepth / *byte*
vibratoSpeed / *byte*
inSpare / *byte*
aWaveCount / *byte*
bWaveCount / *byte*
aWaveList[aWCount] / *waveform*
bWaveList[bWCount] / *waveform*

IntDivRec
quotient / *integer*
remainder / *integer*

InterruptRec
intNum / *word*
intCode / *ptr*

ItemTemplate
itemID / *word*
itemRect / *rect*
itemType / *word*
itemDescr / *pointer*
itemValue / *word*
itemFlag / *word*
itemColor / *pointer*

LColorTable
listFrameClr / *word*
listNorTextClr / *word*
listSelTextClr / *word*
listNorBackClr / *word*
listSelBackClr / *word*

LevelRec
level / *word*

ListCtlRec
ctlNext / *ctlrechndl*
ctlOwner / *grafportptr*
ctlRect / *rect*
ctlFlag / *byte*
ctlHilite / *byte*
ctlValue / *word*
ctlProc / *longprocptr*
ctlAction / *longprocptr*
ctlData / *longword*
ctlRefCon / *longword*
ctlColor / *lcolortableptr*
ctlMemDraw / *voidprocptr*
ctlMemHeight / *word*
ctlMemSize / *word*
ctlList / *memrecptr*
ctlListBar / *ctlrechndl*

LoadSegNameOut
segAddr / *pointer*
fileNum / *word*
segNum / *word*

LocInfo
portSCB / *word*
ptrToPixImage / *pointer*
width / *word*
boundsRect / *rect*

LocRec
curPhrase / *word*
curPattern / *word*
level / *word*

LongDivRec
quotient / *longint*
remainder / *longint*

LongMulRec
lsResult / *longint*
msResult / *longint*

MarkRec
markRefnum / *word*
position / *long*

MemRec
memPtr / *pointer*
memFlag / *byte*

MouseRec
mouseMode / *byte*
mouseStatus / *byte*
yPos / *word*
xPos / *word*

NewlineRec /
newLRefNum / *word*
enableMask / *word*
newlineChar / *word*

OpenRec /
openRefNum / *word*
openPathname / *ptr*
ioBuffer / *handle*

PaintParam
ptrToSourceLocInfo / *locinfotptr*
ptrToDestLocInfo / *locinfotptr*
ptrToSourceRect / **rect*
ptrToDestPoint / **point*
mode / *word*
maskHandle / *handle*

ParamList
paramLength / *word*
wFrameBits / *word*
wTitle / *pointer*
LwRefCon / *ongword*
wZoom / *rect*
wColor / *windcolorPtr*
IwYOrigin / *integer*
IwXOrigin / *integer*
wDataH / *word*
wDataW / *word*
wMaxH / *word*
wMaxW / *word*
wScrollVer / *word*
wScrollHor / *word*
wPageVer / *word*
wPageHor / *word*
LwInfoRefCon / *ongword*
wInfoHeight / *word*
LwFrameDefProc / *ongProcptr*
wInfoDefProc / *voidprocptr*
wContDefProc / *voidprocptr*
wPosition / *rect*
wPlane / *grafportptr*
wStorage / *windrecptr*

PathNameRec
pathname / *ptr*
newPathname / *ptr*

PenState
psPnSize / *point*
psPnMode / *word*
psPnPat / *pattern*
psPnMask / *mask*

PrefixRec
prefixNum / *word*
prefix / *ptr*

PrInfoRec
iDev / *word*
iVRes / *word*
iHRes / *word*
rPage / *rect*

PrJobRec
iFstPage / *word*
iLstPage / *word*
iCopies / *word*
bJDocLoop / *byte*
fFromUser / *byte*
pIdleProc / *wordprocptr*
pFileName / *pointer*
iFileVol / *word*
bFileVers / *byte*
bJobX / *byte*

PrRec /
prVersion / *word*
prInfo / *prinforec*
rPaper / *rect*
prStl / *prstylerec*
prInfoPT[14] / *byte*
prXInfo[24] / *byte*
prJob / *prjobrec*
printX[38] / *byte*
iReserved / *word*

PrStyleRec
wDev / *word*
internA[3] / *word*
feed / *word*
paperType / *word*
crWidth / *word*
reduction / *word*
internB / *word*

QDProcs
stdText / *voidprocptr*
stdLine / *voidprocptr*
stdRect / *voidprocptr*
stdRRect / *voidprocptr*
stdOval / *voidprocptr*
stdArc / *voidprocptr*
stdPoly / *voidprocptr*
stdRgn / *voidprocptr*
stdPixels / *voidprocptr*
stdComment / *voidprocptr*
stdTxMeas / *voidprocptr*
stdTxBnds / *voidprocptr*
stdGetPic / *voidprocptr*
stdPutPic / *voidprocptr*

RadioColors
radReserved / *word*
radNor / *word*
radSel / *word*
radTitle / *word*

ReadConfigRec
rcRepeatDelay / *byte*
rcLayoutOrLang / *byte*
rcADBAddr / *byte*

RomFontRec
rfFamNum / *word*
rfFamStyle / *word*
rfSize / *word*
rfFontHandle / *fonthndl*
rfNamePtr / *pointer*
rfFBRExtent / *word*

SetConfigRec
scADBAddr / *byte*
scLayoutOrLang / *byte*
scRepeatDelay / *byte*

SFReplyRec
good / *boolean*
fileType / *word*
auxFileType / *word*
filename[16] / *char*
fullPathname[129] / *char*

SoundParamBlock
waveStart / *pointer*
waveSize / *word*
freqOffset / *word*
docBuffer / *word*
bufferSize / *word*
SoundParamBlock *nextWavePtr / *struct*
volSetting / *word*

SynchRec
synchMode / *byte*
synchKybdMouseAddr / *byte*
synchLayoutOrLang / *byte*
synchRepeatDelay / *byte*

TxtMaskRec
orMask / *word*
andMask / *word*

VolumeRec
deviceName / *ptr*
volName / *ptr*
totalBlocks / *long*
freeBlocks / *long*
fileSysID / *word*

Waveform
wfTopKey / *byte*
wfWaveAddress / *byte*
wfWaveSize / *byte*
wfDocMode / *byte*
wfRelPitch / *word*

WindColor
frameColor / *word*
titleColor / *word*
tBarColor / *word*
growColor / *word*
infoColor / *word*

WindRec
wNext / *grafportptr*
port / *grafport*
wPadding[16] / *byte*
wStrucRgn / *rgnhandle*
wContRgn / *rgnhandle*
wUpdateRgn / *rgnhandle*
wControls / *ctlrechndl*
wFrameCtrls / *ctlrechndl*
wFrame / *word*

WmTaskRec
wmWhat / *word*
wmMessage / *longword*
wmWhen / *longword*
wmWhere / *point*
wmModifiers / *word*
wmTaskData / *longword*
wmTaskMask / *longword*

ACEBootInit	Initializes ACE Tool Set.
ACEReset	Resets ACE Tool Set; called only when system is reset.
ADBBootInit	Initializes ADB Tool Set; called only by Tool Locator.
ADBReset	Resets ADB Tool Set; called only when system is reset.
ChooseCDA	Activates Desk Manager and displays CDA menu.
ClrHeartBeat	Clears the heartbeat routine queue.
CtlBootInit	Initializes Control Manager; called only by Tool Locator.
CtlReset	Resets Control Manager; called only when system is reset.
DeskBootInit	Initializes Desk Manager; called only by Tool Locator.
DeskReset	Resets Desk Manager; called only when system is reset.
DeskStartUp	Starts up Desk Manager for use by application.
DialogBootInit	Initializes Dialog Manager; called only by Tool Locator.
DialogReset	Resets Dialog Manager; called only when system is reset.
DoWindows	Returns address of direct-page work area used by Event Manager.
EMBootInit	Initializes Event Manager; called only by Tool Locator.
EMReset	Returns error if Event Manager is active; otherwise, does nothing.
FMBootInit	Initializes Font Manager; called only by Tool Locator.
FMReset	Resets Font Manager; called only when system is reset.
IMBootInit	Initializes Integer Math Tool Set; called only by Tool Locator.
IMReset	Resets Integer Math Tool Set; called only when system is reset.
LEBootInit	Initializes LineEdit Tool Set; called only by Tool Locator.
LEReset	Returns error if LineEdit is active.
ListBootInit	Initializes List Manager; called only by Tool Locator.
ListReset	Resets List Manager; called only when system reset occurs.
LoaderInitialization	Initializes the System Loader.
LoaderReset	Resets System Loader.
LoaderShutDown	Shuts down the System Loader.
LoaderStartUp	Starts up the System Loader.
MenuBootInit	Initializes Menu Manager; called only by Tool Locator.
MenuReset	Resets Menu Manager; called only when system is reset.
MidiBootInit	Initializes MIDI Tool Set; called only by Tool Locator.
MidiReset	Resets MIDI Tools; called by system reset.
MMBootInit	Initializes Memory Manager; called only by Tool Locator.
MMReset	Resets Memory Manager; called only when system is reset.
MTBootInit	Initializes Miscellaneous Tool Set; called only by Tool Locator.

MTReset	Resets Miscellaneous Tool Set; called when system is reset.
NSBootInit	Initializes Note Synthesizer.
NSReset	Performs same function as NSShutDown.
PMBootInit	Initializes Print Manager; called only by Tool Locator.
PMReset	Resets Print Manger; called only when system is reset.
QDAuxBootInit	Initializes QuickDraw II Auxiliary; called only by Tool Locator.
QDAuxReset	Resets QuickDraw II Auxiliary; called only when system is reset.
QDBootInit	Initializes QuickDraw II; called only by Tool Locator.
QDReset	Resets QuickDraw II; called only when system is reset.
RestAll	Restores all variables that Desk Manager preserves when CDA menu is activated.
RestScrn	Restores screen area saved by Desk Manager.
SANEBootInit	Initializes SANE Tool Set; called only by Tool Locator.
SANEReset	Resets SANE Tool Set; called only when system is reset.
SaveAll	Saves all variables that Desk Manager preserves when CDA menu is activated.
SaveScrn	Saves 80-column text screens in banks $00, $01, $E0, and $E1.
SchBootInit	Initializes Scheduler; called only by Tool Locator.
SchFlush	Flushes all tasks in Scheduler's queue.
SchReset	Resets Scheduler; called only when system is reset.
ScrapBootInit	Initializes Scrap Manager; called only by Tool Locator.
ScrapReset	Resets Scrap Manager; called only when system is reset.
SeqBootInit	Dummy routine provided for consistency with design conventions of Apple IIGS Toolbox.
SeqReset	Note Sequencer's system reset handler.
SetDefaultTPT	Sets default tool pointer table to current TPT.
SFBootInit	Inittializes Standard File Operations Tool Set; made only by Tool Locator.
SFReset	Resets Standard File Operations Tool Set; called only when system is reset.
SoundBootInit	Initializes Sound Tool Set; called only by Tool Locator.
SoundReset	Resets Sound Tool Set; called only when system is reset.
SystemEvent	Entry point Event Manager uses to get into Desk Manager.
TextBootInit	Initializes Text Tool Set; called only by Tool Locator.
TextReset	Resets Text Tool Set; called only when system is reset.
TLBootInit	Initializes Tool Locator and all other ROM-based tool sets.
TLReset	Initializes Tool Locator and all other ROM-based tool sets when system is reset.
WindBootInit	Initializes Window Manager; called only by Tool Locator.
WindReset	Resets Window Manager; called only when system is reset.

Index of Tool Calls by Tool Set

ProDOS

QuickDraw II

Index of Tool Calls in Alphabetical Order

THE APPLE PUBLISHING SYSTEM

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh® computers and Microsoft® Word. Proof and final pages were created on the Apple LaserWriter® printers. POSTSCRIPT®, the LaserWriter page-description language, was developed by Adobe Systems Incorporated.

Text type and display are Apple's corporate font, a condensed version of Garamond. Bullets are ITC Zapf Dingbats®.