# Programmer's Reference for System 6.0.1

**Edited by Mike Westerfield**

# Contents

# Acknowledgments

This book was developed from Apple's Engineering Requirements Specification (ERS) documents for System 6.0.1. The source material included:

*Apple IIGS System Software 6.0.1*, Version 1.0d1, November 29 1993, David A. Lyons
*GS/OS MS-DOS File System Translator External ERS*, Version 0.04, Greg Branche

The source material is quoted heavily. All source material is Copyright 1993, Apple Computer, Inc. It is used here with permission.

Technical documentation is notoriously hard to get right. After working for months on a project, it's hard to force yourself to really read all of those arcane technical details carefully enough to make sure they are right. The often thankless job of reviewing draft documentation is very important, so I want to thank those who took their time to read all of this one last time. They are Greg Branche, Matt Deatherage, Dave Lyons, Jim Murphy and Steve Stephenson.

4700 Irving Blvd N.W. Suite 207
Albuquerque, N.M. 87114
(505) 898-8183

# Chapter 1 **Toolbox Changes**

## New Features of the Control Manager

- There are four new Control Manager calls: `SetCtlValueByID`, `GetCtlValueByID`, `InvalOneCtlByID`, and `HiliteCtlByID`.

### Static Text Controls

- The Static Text control supports a new `ctlFlag` bit. If bit 4 ($0010), `fSquishText`, is set as well as `fBlastText`, the control will draw the text with `DrawStringWidth` (in QuickDraw II Auxiliary) to compress and truncate on the right as needed to make the text fit inside the control rectangle. If you set the `fSquishText` bit, you must also set the `fBlastText` bit.

### Thermometer Controls

- Setting a thermometer control's value no longer draws anything if the control is invisible.

### Pop-Up Menu Controls

- For enhancements to Pop-Up Menu controls, see the Menu Manager update.

### Line Edit Controls

- For enhancements to Line Edit controls, see the Line Edit update.

### Icon Button Controls

- The Icon Button control now supports "sticky" icon controls. If bit 4 of the `ctlFlag` field is set and the mouse button is released when the cursor is inside the control, the control stays highlighted to show that it is "selected." The `ctlValue` field contains $0001 when the icon is in the selected state, and $0000 when it is not. An extra one-word field, #12, has been added to the control template to allow for an initial value word for this type of control.

### Scroll Bar Controls

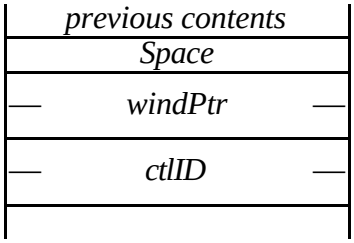
- `CtlStartUp` removes the `RefreshDesktop` run queue routine, so the desktop doesn't refresh an extra time when starting an application in a different resolution from the one you used last (the scroll bars thought they had to redraw in 6.0, even though they really didn't).

## GetCtlValueByID $3D10

Returns the current value of the control that has the specified control ID in the specified window. This is just like `GetCtlValue`, except you pass a control ID instead of a control handle.

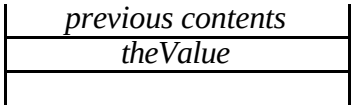
### Parameters

Stack before call




**Word**—Space for result
**Long**—Window containing the control (NIL = front)
**Long**—Control ID of the control
**<—SP**

Stack after call




**Word**—Current control value
**<—SP**

**Errors** Returned unchanged from `GetCtlHandleFromID` and `GetCtlValue.`

**C**

```
extern pascal Word GetCtlValueByID (windPtr, ctlID);
WindowPtr windPtr;
Long ctlID;
```

`windPtr` Pointer to the window containing the control. If the control is in the front window, you may pass NIL.

`ctlID` Control ID for the control.

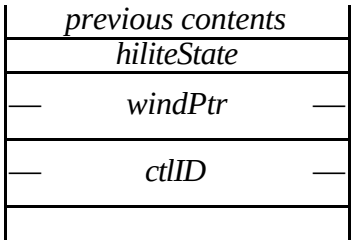
`theValue` Current value of the control.

## `HiliteCtlByID` $3F10

Changes the way a specified control is highlighted, just as if you called `HiliteControl`, except you specify the control by window pointer and control ID.

### Parameters

Stack before call



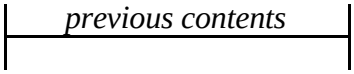

**Word**—New highlight value

**Long**—Window containing the control (NIL = front)

**Long**—Control ID of the control

**<—SP**

Stack after call




**<—SP**

**Errors** Returned unchanged from `GetCtlHandleFromID` and `HiliteControl`.

**C**

```
extern pascal void HiliteCtlByID (hiliteState, windPtr, ctlID);
Word hiliteState;
WindowPtr windPtr;
Long ctlID;
```

`hiliteState` New value for the control's highlight flag.

`windPtr` Pointer to the window containing the control. If the control is in the front window, you may pass NIL.

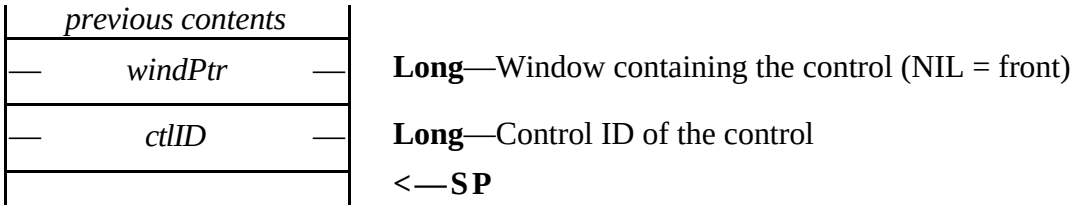
`ctlID` Control ID for the control.

## InvalOneCtlByID $3E10

Invalidates a control's rectangle, just as if you called `InvalRect` on the control's rectangle. This causes the control to get redrawn later, when your application has a chance to process an update event for the window. You specify the control by its window pointer and control ID.

### Parameters

Stack before call




Stack after call


**Errors** Returned unchanged from `GetCtlHandleFromID`.

**C**

```
extern pascal void InvalOneCtlByID (windPtr, ctlID);
WindowPtr windPtr;
Long ctlID;
```

`windPtr` Pointer to the window containing the control. If the control is in the front window, you may pass NIL.

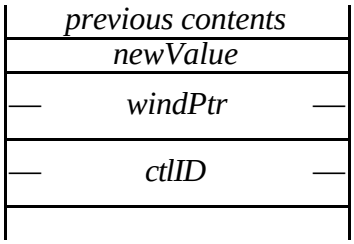
`ctlID` Control ID for the control.

## SetCtlValueByID $3C10

Sets the value of the control that has the specified control ID in the specified window. This is just like `SetCtlValue`, except you pass a control ID instead of a control handle.

### Parameters

Stack before call



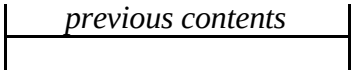

**Word**—New value for the control

**Long**—Window containing the control (NIL = front)

**Long**—Control ID of the control

**<—SP**

Stack after call




**<—SP**

**Errors** Returned unchanged from `GetCtlHandleFromID` and `SetCtlValue.`

**C**

```
extern pascal void SetCtlValueByID (newValue, windPtr, ctlID);
Word newValue;
WindowPtr windPtr;
Long ctlID;
```

`newValue` New control value.

`windPtr` Pointer to the window containing the control. If the control is in the front window, you may pass NIL.

`ctlID` Control ID for the control.

## New Features of the Desk Manager

### Classic Desk Accessory changes

- If bit 7 of Battery RAM byte $59 is set, the system installs the Memory Peeker and Visit Monitor CDAs for ROM 1 systems (just like ROM 3 always has).

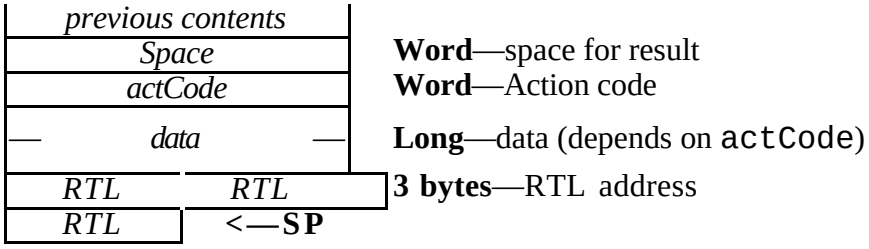
### New Desk Accessory changes

- `OpenNDA` sends `reOpenAction` ($000C) to the action routine of an already-open NDA to give the NDA a chance to do something other than just have the window come to the front. If the NDA wants the system to take no further action (that is, skip the normal `SelectWindow` call), it should store a $0001 into the word pointed to by the data parameter (passed in the X and Y registers, or on the stack as shown below).

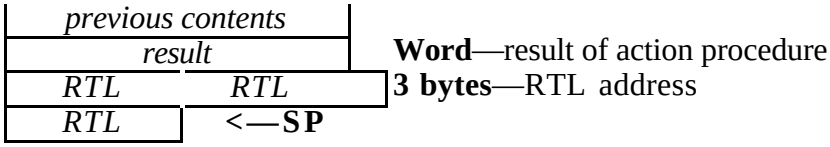
- On ROM 3 only, `DeskShutDown` sets $07FC to zero if slot 4 is set to internal. This stops the mouse from freezing in desktop applications after visiting the CDA menu when you have previously run an application that left a non-zero value in $07FC. This was a problem on ROM 3 systems only.

- If you set bit 31 of an action procedure pointer (for an NDA or a system window), the system does a stack-based dispatch instead of a register-based dispatch. The stack on entry to your action procedure looks like this:

Stack before call




Before returning, you must remove `actCode` and `data` and set result so that the stack looks like this:




You can prototype your action procedure like this:

**C**

```
pascal Word MyActionProc (actCode, data);
Word actCode;
Long data;
```

## New Feature of the Integer Math Tool Set

- `Int2Dec` and `Long2Dec` now return “zero ” if bit 31 of `stringPtr` is set, the value being converted is zero, and the buffer length is at least 5.

  **Note** The string returned has a total of five characters. The fifth character is a trailing blank.

## New Features of the LineEdit Tool Set

- There is one new call, `LEClassifyKey`.

### Line Edit Controls

- There is a new field in the Line Edit control template. Parameter number 9 is a word called `keyMask`. The control accepts keys only if the `LEClassifyKey` result has some bits set in common with the `keyMask` parameter. The `keyMask` parameter defaults to $0001, which causes the control to accept all keypresses, as usual.

- The `pwChar` field in the Line Edit control template supports a new value. A value of $FFFF now means the control is not for password entry. (Previously, the legal values were $0000 [default password character] and $0001 through $00FF [specific password character]. The parameter's presence implied that the control was for password entry, which is not sufficient now that there is an optional ninth parameter.)

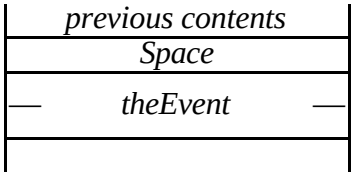
## New Line Edit Calls

### LEClassifyKey $2514

Returns a word with bits indicating what categories a specified event falls into. This is sometimes useful in deciding what events to pass along to LEKey.

**Parameters**

Stack before call

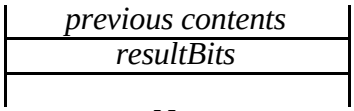



**Word**—Space for result

**Long**—Pointer to event record

**<—SP**

Stack after call




**Word**—Value categorizing the event

**<—SP**

**Errors** None.

**C**

```
extern pascal Word LEClassifyKey (theEvent);
EventRecPtr theEvent;
```

`theEvent` Pointer to the event record to check.

`resultBits` Collection of bits, set as follows:

| | |
|---|---|
| bit 15 | Special editing key. (LEKey will do something special; it will not insert this key into the text.) |
| bit 14 | digit (0-9) |
| bit 13 | hex digit (a-f, A-F) |
| bit 12 | letter (A-Z, a-z) |
| bit 11 | any non-control key |
| bits 10-1 | reserved (ignore) |
| bit 0 | any key |

If the event is not a keyDown or autoKey event, all currently-defined bits will be zero.

## New Features of the List Manager

- Setting flag bit 15 in the `CompareStrings` flags now makes it compare GS/OS strings instead of Pascal strings.

- Fixed a problem affecting `ListKey`, `CompareStrings`, `SortList`, and `SortList2` with a `compareProc` of 1. Characters $20 to $3F (including digits), and $60 were being accidentally "uppercased." For example, in System 6.0 "5" would map into right-arrow, which made `ListKey` move down one item.

- The standard item-draw procedure uses `DrawStringWidth`, with flags allowing horizontal compression and truncation on the right with an ellipsis.

## Clarifications of Previous Media Control Tool Set Documentation

- `MCGetStatus` accepts two selector values that are not mentioned in the call description on page 68 of Programmer's Reference To System 6.0, but are mentioned in the chapter summary on page 97. These are `mcSVolumeL` and `mcSVolumeR`.

- `MCStop` is documented incorrectly (page 90). Actually, `MCStop` takes a single input parameter, `mcChannelNo`.

## New Features of the Menu Manager

- When a Pop-up menu control receives a `ctlHandleEvent` message, now it only sends `keyDown` and `autoKey` events to `MenuKey`. It also preserves the menu bar around the `MenuKey` call, so the menu bar is not accidentally left set to the Pop-up menu control.

- Pop-up menu controls now draw the current item using `DrawStringWidth` (in QuickDraw II Auxiliary), so that long item names are compressed or center-truncated.

- Page 104 of *Programmers Reference For System 6.0* should make a distinction between menu records and menu templates. The structure identified as "Menu Item Record" is actually a template. (The system uses it to create a menu item, not to keep track of the item's state once it has been created.)

## New Features of the Miscellaneous Tool Set

- There are two new calls: `DoSysPrefs` and `AlertMessage`.

### SysFailMgr Enhancement

- If you pass NIL for the message string, `SysFailMgr` now provides the following default messages for the specified error codes:

| | |
|---|---|
| $27: | "Could not read or write disk. The disk may be damaged." |
| $201: | "Out of memory (or required memory area was already in use)." |
| $308, $681, $682: | "Detected trashed memory. Software bug or (less likely) bad RAM." |

### SysBeep2 Enhancements

- `SysBeep2` now sends a new `SendRequest` code, `systemSaysForceUndim`, as part of handling all `SysBeep2` codes *except* $006x (screen blanking, screen unblanking).

- The following new `SysBeep2` codes have been defined. The system does not do anything special to support them.

| | |
|---|---|
| `$0070 sbBeginningLongOperation` | A lengthy modal operation is starting. |
| `$0F80 sbFileTransferred` | Upload/download finished. |
| `$0F81 sbRealtimeMessage` | A real-time message needs the user's attention. |
| `$1000 sbConnectedToService` | Connected to an interactive service. |
| `$1001 sbDisconnectedFromService` | Disconnected from an interactive service. |
| `$1002 sbEnteredRealtimeChat` | Started a real time chat in an interactive service. |
| `$1003 sbLeftRealtimeChat` | Left a real time chat in an interactive service. |
| `$1010 sbFeatureEnabled` | The user enabled a feature in a preferences dialog. |
| `$1011 sbFeatureDisabled` | The user disabled a feature in a preferences dialog. |

### ShowBootInfo

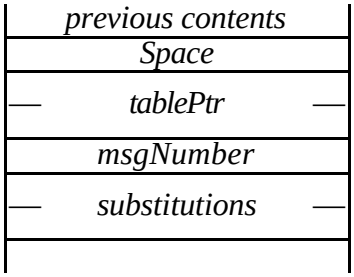
- `ShowBootInfo` now "wraps up" to a new row if you have more than one row of icons. If you wrap off the top of the screen, it starts over at the bottom left, without erasing the screen. (It used to keep recycling the bottom row, wiping it to periwinkle blue every time it filled up.)

## AlertMessage $3E03

AlertMessage displays a message on either the text screen or the Super Hi-Res screen and makes the user choose one of up to three buttons. AlertMessage works in the GS/OS environment only, not while ProDOS 8 is active.

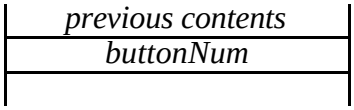
### Parameters

Stack before call




**Word**—Space for result
**Long**—Pointer to the message table
**Word**—Message index number (0, 1, 2...)
**Long**—Pointer to the string substitution table
**<—SP**

Stack after call




**Word**—Button chosen by the user
**<—SP**

**Errors**

```
$0377    onlyFromGSOS      You called AlertMessage from ProDOS 8.
```

**C**

```
extern pascal Word AlertMessage (tablePtr, msgNumber,
   substitutions);
Pointer tablePtr, substitutions;
Word msgNumber;
```

`tablePtr` Points to a table formatted as follows:

```
      dc    i'messageZeroText-*, messageZeroGraphics-*-2'
      dc    i'messageOneText-*, messageOneGraphics-*-2'
      ...
```

There is a pair of offsets for each message. Each offset counts the number of bytes from its own location to the message string. The first offset of each pair is used on the text screen, and the second is used on the Super Hi-Res screen (using AlertWindow).

Each message is an AlertWindow string plus three characters to map the buttons into return values. The three characters should be '0' to '9', indicating what values to return when the first, second, and third buttons are chosen, respectively.

Both string offsets can point to the same string if you want, but the text version does not do word wrapping for you.

The text messages support *0..*9 substitutions and "^" to mark the default button, but they do not support the "#" substitutions that you automatically get (courtesy of `CompileText`) for `AlertWindow`.

`msgNumber` Selects the message to display.

`substitutions` An array of pointers to Pascal strings. See `AlertWindow` for details.

## DoSysPrefs $3F03

DoSysPrefs clears and then sets specified bits in the GS/OS system preferences word, and then returns the original preference word so that you can restore it later.

A typical sequence is:

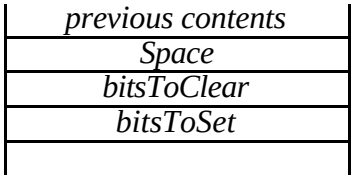
```
/* Clear the $2000 bit to avoid suppressing dialog */
/* Set the force-volume-mount and no-cancel bits */
oldPrefs = DoSysPrefs($2000,$C000);

/* Do some preference-bit-dependent stuff here */

/* Now restore the preferences */
/* Clear all bits, then set the ones that were originally set */
ignore = DoSysPrefs($FFFF, oldPrefs);
```

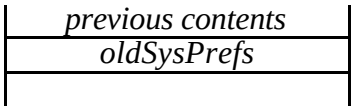
### Parameters

Stack before call




**Word**—Space for result
**Word**—System preference bits to clear
**Word**—System preference bits to set
**<—SP**

Stack after call




**Word**—Previous system preference word
**<—SP**

**Errors** $0301 badInputError Requires GS/OS

**C**

```
extern pascal Word DoSysPrefs (bitsToClear, bitsToSet);
Word bitsToClear, bitsToSet;
```

`bitsToClear` Any bit that is set in this word will force the corresponding bit in the system preference word to zero.

`bitsToSet` Any bit that is set in this word will set the corresponding bit in the system preference word.

`oldSysPrefs` The original system preference word is returned.

## New Features of QuickDraw II

- `QDVersion` is now $0308. `QDVersion` is a standard reference for distinguishing system versions, so it had to change.

## Clarifications of Previous QuickDraw II Documentation

- Starting in System 6.0, `QDShutDown` examines bit 8 of the `masterSCB` word. If the bit is set, QuickDraw leaves the Super Hi-Res screen turned on even after QuickDraw has shut down. (`ShutDownTools` took advantage of this in System 6.0 to help implement smooth transitions between applications, but the mechanism was not spelled out.)

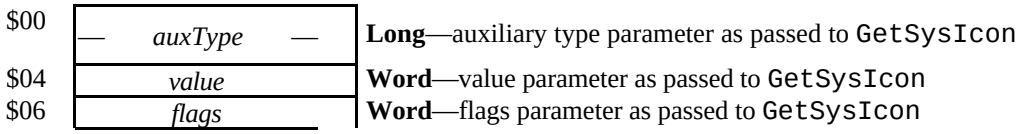
- `GetPixel` does not work past the first 64K of a pixel map (it never has).

## New Features of QuickDraw II Auxiliary

- There are three new calls: `DrawStringWidth`, `UseColorTable`, and `RestoreColorTable`.

### GetSysIcon Enhancements

- `GetSysIcon` now calls `SendRequest` with a new request code, `systemSaysGetSysIcon` ($1201), to allow utilities and applications to override or extend the built-in set of icons. The `dataIn` parameter points to a structure formatted as follows:




  Your request procedure (installed using `AcceptRequests` in the Tool Locator) should decide whether it will provide an icon for the given input parameters. If not, simply reject the request. If you will handle it, put an icon pointer at offset +002 in the `dataOut` buffer and accept the request.

- `GetSysIcon` has built-in icons for five additional file types: text ($04), source file ($B0), AppleSoft BASIC program ($FC), archive ($E0), and binary file ($06). The complete set is now:

| Kind | File Type |
|---|---|
| Folder, open or closed | $000F |
| Application | $00B3 or $00FF |
| Stack | $0055 |
| Text | $0004 |
| Source file | $00B0 |
| AppleSoft BASIC program | $00FC |
| Archive file | $00E0 |
| Binary file | $0006 |
| Document | any other file type |

## Clarifications of Previous QuickDraw II Auxiliary Documentation

- Toolbox Reference 3, page 44-15, for `SpecialRect`, says that the low-order 4 bits of `frameColor` and `fillColor` specify the colors. Actually, all 16 bits are significant. To get solid patterns, use $0000, $1111, ..., $EEEE, $FFFF.

## New QuickDraw II Auxiliary Calls

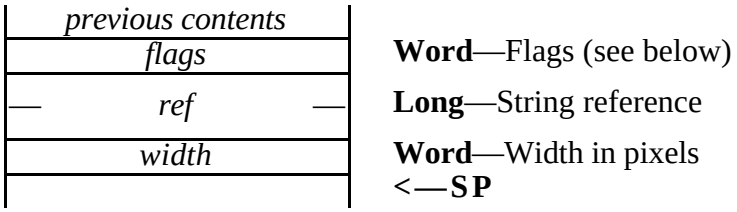
### DrawStringWidth $1512

DrawStringWidth draws a string in a specified horizontal width on a single line. The string is compressed and truncated as necessary, if allowed.

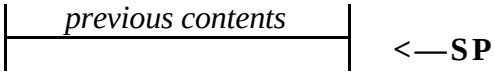
The string can be in Pascal, C, or GS/OS format, and you can reference it by pointer, handle, or resource ID.

**Parameters**

Stack before call




Stack after call




**Errors** $1231 badQDAuxValue Illegal input values.
LoadResource errors are returned unchanged.

**C**

```
extern pascal void DrawStringWidth (flags, ref, width);
Word  flags, width;
Long  ref;
```

flags Selects various options, as follows:

- bit 15 prevent compression
  - 0 = Allow string to be drawn with the characters scrunched together if the full width doesn't fit (uses SetCharExtra(-1.0)).
  - 1 = Don't allow compression.
- bits 14-13 type of truncation
  - 00 = none (Truncates on the right, but does not indicate the truncation with an ellipsis character.)
  - 01 = left (Replace beginning of string with ellipsis, if necessary.)
  - 10 = center (Replace middle of string with ellipsis, if necessary.)
  - 11 = right (Replace end of string with ellipsis, if necessary.)
- bits 12-4 reserved (use 0)
- bits 3-2 type of string
  - 00 = Pascal (leading length byte)
  - 01 = C (terminating null character)
  - 10 = GS/OS (leading length word)
  - 11 = reserved (don't use)

bits 1-0 type of reference to string
00 = pointer
01 = handle
10 = resource ID
11 = reserved (don't use)

`ref` String reference. What you pass here depends on bits 0-3 of `flags`.

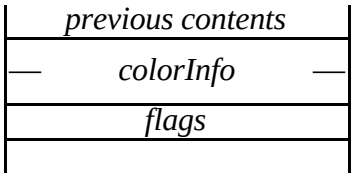
`width` Width of the destination area, in pixels. The string is forced to this width using the method specified by bits 13-15 of `flags`.

## RestoreColorTable $1712

RestoreColorTable undoes the effects of UseColorTable. See UseColorTable for more information.

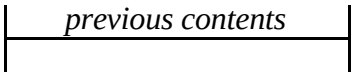
### Parameters

Stack before call




**Long**—Color information from UseColorTable
**Word**—Flags (See below)
**<—SP**

Stack after call




**<—SP**

**Errors** DisposeHandle errors are returned unchanged.

**C**

```
extern pascal void RestoreColorTable (colorInfo, flags);
Long colorInfo;
Word flags;
```

`flags` Defined as follows:

| | |
|---|---|
| bit 15 | reserved (use zero) |
| bit 14 | 1 = skip the normal call to CtlNewRes |
| bit 13 | 1 = change the SCBs for the menu bar, too |
| bits 12..0 | reserved (use zero) |

`colorInfo` Value returned by UseColorTable.

## UseColorTable $1612

UseColorTable preserves Scanline Control Bytes (SCBs) and sets them to use a color table you specify. It also preserves the old contents of that color table and sets the color table to the data you specify, or to a standard color table.

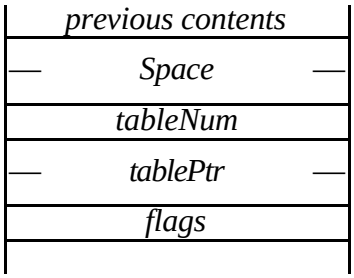
UseColorTable returns a value that you later pass to RestoreColorTable to restore the color table and SCBs. Typically, you might call UseColorTable when handling a window's activate event, and call RestoreColorTable when handling the window's deactivate event.

The colorInfo value returned should be used *once* in a RestoreColorTable call. If you make a UseColorTable call and for some reason wind up not making a corresponding RestoreColorTable call, you should call DisposeHandle on the colorInfo value.

UseColorTable normally calls CtlNewRes for you to cause controls (scroll bars, for example) to redraw as needed for the new colors. There is a flag bit to override this behavior. Normally, all SCBs except those for the menu bar are affected. There is a flag bit you can set to include all the SCBs.

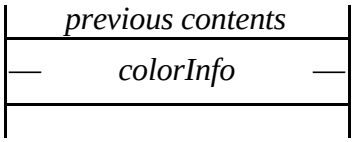
### Parameters

Stack before call




**Long**—Space for result
**Word**—Color table number (0..15)
**Long**—Pointer to color table (NIL = standard)
**Word**—Flags (See below)
**<—SP**

Stack after call




**Long**—Value to pass to RestoreColorTable
**<—SP**

**Errors** NewHandle errors are returned unchanged.

**C**
```
extern pascal Long UseColorTable (tableNum, tablePtr, flags);
Word tableNum, flags;
ColorTablePtr tablePtr;
```

`tableNum` Number of the color table to change.

`tablePtr` Pointer to the new color table. Pass NIL for the default color table.

`flags` Defined as follows:

| | |
|---|---|
| bit 15 | use the standard 640-mode color set, even in 320 mode (ignores `tablePtr`) |
| bit 14 | 1 = skip the normal call to `CtlNewRes` |
| bit 13 | 1 = change the SCBs for the menu bar, too |
| bits 12..0 | reserved (use zero) |

`colorInfo` Handle of the information `RestoreColorTable` will use to restore the original color table and SCB. If `RestoreColorTable` is not called, call `DisposeHandle` to dispose of this buffer.

## New Features of the Resource Manager

- There are two new calls: `OpenResourceFileByID` and `CompactResourceFile`.
- Fixed a string-comparison problem in `RMFindNamedResource` and `RMLoadNamedResource`. Sometimes in System 6.0 you could wind up loading a resource whose name began with the name you asked for, but contained additional characters after the characters you asked for.
- `OpenResourceFile` now makes sure the resource map was entirely read. If it runs off the end of the file while trying to read the map, it returns a GS/OS `eofEncountered` error.
- Added a new bit to `mapFlag` in the in-memory copy of the resource map. Bit 0 is now defined as `fileReadWrite`. When a file is opened, it gets set to 1 if the file is opened read/write. If it's opened with read-only access, the bit is set to 0. This bit is for examination only.
- `AddResource`, `RemoveResource`, `WriteResource`, and `MarkResourceChange` now verify that the target file can be written to before actually doing anything. They all return a GS/OS `invalidAccess` error if the file cannot be written to. The exception to this is `MarkResourceChange` when the resource in question is being marked unchanged; it is allowed because it won't eventually cause a write.
- Fixed `WriteResource` to write the size of the resource as it appears on disk, rather than the size of the resource's handle in memory. This properly allows for converters to write resources that are smaller than their in-memory size without destroying the file.
- `CloseResourceFile` returns error `resFileNotFound` ($1E07), instead of no error, on a non-zero argument that doesn't match an open file ID.

## Clarifications of Previous Resource Manager Documentation

- On page 215 of Programmer's Reference For System 6.0, the `rType` and `rID` parameter descriptions for `RMSetResourceName` should read "...for the resource to name" (not "...for the resource to load").
- For `RMSetResourceName`, note that the resource to be named must already exist, or you will get error $1E06, `resourceNotFound`.
- The system does not log in a resource converter for the `rCodeResource` type (it never has, and it never will). If your application needs to use resources of type `rCodeResource`, you must explicitly use `ResourceConverter` to log in an application resource converter (usually the one returned by `GetCodeResConverter`).

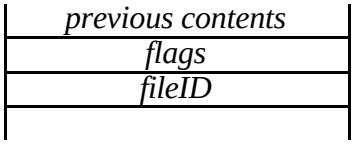
## New Resource Manager Calls

### CompactResourceFile $2F1E

`CompactResourceFile` consolidates all free blocks in an open resource file into a single free block at the end.

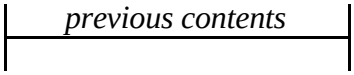
**Parameters**

Stack before call




**Word**—Flags word (Reserved; use $0000)
**Word**—ID of resource file to compact
**<—SP**

Stack after call




**<—SP**

**Errors**

| | | |
|---|---|---|
| $1E07 | resFileNotFound | The specified resource file was not found. |
| $004E | invalidAccess | The file is not opened with write access. |

GS/OS errors are returned unchanged.
Memory Manager errors are returned unchanged.

**C**

```
extern pascal void CompactResourceFile (flags, fileID);
Word flags, fileID;
```

`flags` This parameter is reserved for future expansion. For now, always pass $0000.

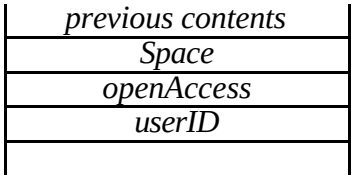
`fileID` File ID for the resource file to compact.

## OpenResourceFileByID $2E1E

OpenResourceFileByID starts the Resource Manager for you if it isn't already started under the specified user ID (and it makes that user ID the current resource application in any case). Then it uses LGetPathname2 to find pathname for the specified user ID and calls OpenResourceFile for you on that file. Note that the oldResApp result is valid even if you get an error.

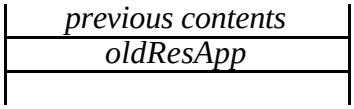
### Parameters

Stack before call




**Word**—Space for result
**Word**—Request access
**Word**—Application user ID
**<—SP**

Stack after call




**Word**—Previous CurResourceApp value
**<—SP**

**Errors** LGetPathname2 and OpenResourceFile errors are returned unchanged.

**C**

```
extern pascal Word OpenResourceFileByID (openAccess, userID);
Word openAccess, userID;
```

openAccess Open access flags. See Open in *Apple IIGS GS/OS Reference*.

userID User ID for the application.

oldResApp CurResourceApp value before this call.

## New Features of Scrap Manager

- There is one new call, `ShowClipboard`.
- `PutScrap` now changes the scrap count, as returned by `GetScrapCount` (for polling to see if the clipboard contents changed).

## New Scrap Manager Calls

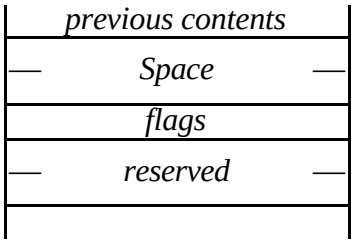
### ShowClipboard $1516

ShowClipboard creates a System window that takes care of the clipboard display for you. (It's used in Finder and Teach, for example.) It displays Text, Picture, and Sound scraps.

To open the Clipboard window, use a flags value of $8000. To close the window, use a flags value of $4000. To find the WindowPtr of the Clipboard window without opening or closing it, use a flags value of $0000. In all cases, the windowPtr result is either a valid WindowPtr or NIL (for no window).

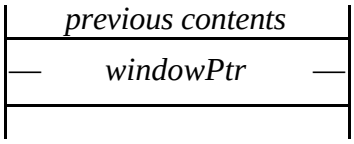
**Parameters**

Stack before call




**Long**—Space for result
**Word**—Flags (See below)
**Long**—Reserved; use zero
**<—SP**

Stack after call




**Long**—Pointer to the clipboard window or NIL
**<—SP**

**Errors** Errors from SelectWindow and NewWindow2 are returned unchanged.

**C**
```
extern pascal WindowPtr ShowClipboard (flags, reserved);
Word flags;
Long reserved;
```

flags — Defined as follows:
- bit 15 — 1 = open the Clipboard window (or bring to front if already open)
- bit 14 — 1 = close the Clipboard window if it's open
- bits 13..0 — reserved, use 0

reserved — This parameter is reserved for future expansion. For now, always pass 0.

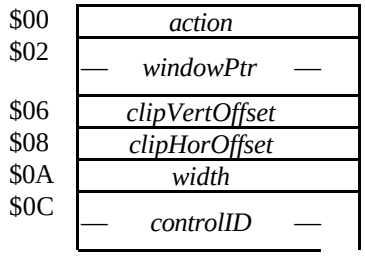
windowPtr — Pointer to the clipboard window. If the clipboard windows is closed, windowPtr will be NIL.

**Side Effects**

The clipboard window calls SendRequest with request code $000C, systemSaysDoClipboard, to allow utilities and applications to display additional types of

data in the system's clipboard window. (You can use `AcceptRequests`, in the Tool Locator, to register a request procedure to receive `systemSaysDoClipboard` requests.)

`dataIn` points to a buffer with the following format:




**Word**—Action code (0=draw contents, 1=hit a control, 2=killing controls)

**Long**—Clipboard window pointer

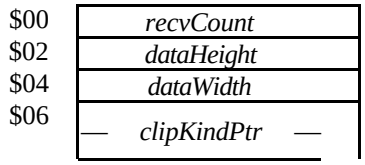
**Word**—Top of the area to draw in

**Word**—Left edge of the area to draw in

**Word**—Suggested maximum width to draw in

**Long**—control ID of control hit (when actionCode = 1)

`dataOut` is only used on draw actions. In that case, it points to a buffer with the following format:




**Word**—set by `SendRequest`

**Word**—height of content

**Word**—width of content

**Long**—C string defining the kind of data drawn

On receiving a draw-contents action, your request procedure should examine the clipboard (using Scrap Manager calls such as `GetIndScrap`). If there is no data that you want to draw, simply reject the request. If there is data you want to draw, retrieve the data, draw it, and accept the request.

You may also use the Control Manager to create controls in the Clipboard window to help draw your content. In that case, create the controls on the first draw contents action you accept, then use those same controls until you receive a kill controls action. If you create any controls, you should always call `DrawControls` when you accept a draw contents action.

You must fill in the `dataHeight` and `dataWidth` fields of `dataOut` to indicate the size of content you drew, so the system can adjust the Clipboard window's scroll bars as needed. Finally, you must set the `clipKindPtr` field to a pointer to a C-style string that describes the type of data you drew. This string will appear after "Clipboard contents:" in the Clipboard window's information bar.

On receiving a hit-a-control action, your request procedure should do anything appropriate, given the control ID in the `dataIn` record, and then accept the request.

On receiving a kill controls action, your request procedure should do anything appropriate, given that the system is about to do a `KillControls` on the Clipboard window. For example, if you allocated any extra memory as a result of a draw-contents action, you should dispose of that memory here. The procedure should always accept this request.

## New Features of Standard File Operations Tool Set

- SFReScan now makes a DInfo and Volume call on the volume in prefix 8 and updates all of the controls accordingly. Also, SFReScan now works in the volumes list as well as the files list.

- Fixed a problem where SFGetFile (but not SFGetFile2) would loop forever when prefix zero was empty.

- Changed the way Standard File handles multiple edit line items in "put file" dialogs, so that there can be more than just the single edit line item.

## Clarification of Previous Standard File Operations Tool Set Documentation

- Apple IIGS Toolbox Reference Volume 3, page 48-9 describes name as "Filename string, containing (nameLength = 2) bytes of data, not to exceed 253 characters." It should read "nameLength - 2".

## New Features of TextEdit Tool Set

- `TEPaintText` now properly fully-justifies text.
- Fixed a problem with non-targetable TextEdit controls. They could start out active (with a usable scroll bar for example), and then become inactive when the window became inactive, but the control would not get reactivated when the window came back to the front.
- When `TEStartUp` calls `FMStatus`, it now pushes pre-zeroed result space, in case the Font Manager is not loaded. The result is that you get a `TEStartUp` error reliably now, instead of just sometimes, if the Font Manager isn't available.

## New Features of the Tool Locator

### StartUpTools/ShutDownTools enhancements

- `StartUpTools` now returns any error from `ResourceStartUp` (and returns a NIL result).
- `ShutDownTools` tolerates errors from `SFShutDown` (for compatibility with errant NDAs that shut down Standard File during `DeskShutdown` even if they did not own it).
- `ShutDownTools` no longer calls `HideCursor` if QuickDraw is not active. (In 6.0, it can crash if an application calls `ShutDownTools` with QuickDraw inactive.) If the new-for-6.0 "no Resource Manager" flag bit is set, there is no problem, since it was already skipping the `HideCursor` call.
- `ShutDownTools` checks for a NIL input and behaves sanely. It also shuts down the Resource Manager even if you get some other error.

### New Request Codes

- See the Scrap Manager chapter for a description of the `systemSaysDoClipboard` request.
- See the QuickDraw II Auxiliary chapter for a description of the `systemSaysGetSysIcon` request.

### systemSaysForceUndim

- Request $000D, `systemSaysForceUndim`, requests that screen savers return the screen to a normal state. The system sends this request in some circumstances. If you send the request yourself, you should pass `dataIn` and `dataOut` values of NIL, and you should broadcast the request to all available request procedures. If you receive the request, you should ignore the `dataIn` and `dataOut` parameters and simply un-dim the screen.

  GS/OS sends `systemSaysForceUndim` before deciding whether to put a message on the text screen or the Super Hi-Res screen.

### srqQuit

- Request $0011, `srqQuit`, asks an application to quit at its next opportunity. Typically, an application will set a global flag that tells it to quit when it eventually gets back to its main event loop. The application may not actually quit even after accepting this request, since the user may elect to cancel because there are documents open that have not been saved.

### srqOpenOrPrint

- Request $0010, `srqOpenOrPrint`, requests that an application re-check the Message Center for messages of type $0011, GS/OS pathnames of files to open or print, and handle the message as if the application had just been launched by the Finder.

  **Note** Teach 1.1 accepts `srqOpenOrPrint` but does not respond to it reliably.

## New Features of the Window Manager

- There is one new call, `UpdateWindow`.

- Fixed `TaskMaster` to handle the `tmNoGetNextEvent` bit correctly (bit 21, $0020/0000, in `wmTaskMask`). This bit tells `TaskMaster` to skip its `GetNextEvent` call and simply assume that the task record you pass in already contains an event that it should process. (The `tmNoGetNextEvent` bit has been defined since System 5.0.3, but before System 6.0.1 it only worked correctly with ROM version 1. Now it works with both ROM 1 and ROM 3.)

- There are two flag bits in the high-order byte of `wContDraw` pointers for application windows. By using these bits, you allow the system to redraw your window at certain times when it could not do so before (like behind modal Standard File dialogs and `AlertWindow` messages).

  bit 31 If your content-draw routine is self-contained, so that it can be called from any environment (unknown Bank and Direct Page register, unknown ResourceApp setting, unknown CtlParamPtr values), then you may set bit 31 of your `wContDraw` pointer.

  bit 30 If your content-draw routine does not depend on making GS/OS calls (even indirectly, by going to disk to get resources), then you may set bit 30 of your `wContDraw` pointer.

### DoModalWindow

- `DoModalWindow` now uses `UpdateWindow`. If the `mwUpdateAll` bit is clear, it passes flags of $0000 for the dialog window, but $8000 (background update) for other windows—so the other windows update only if it's safe for them to update in the background. If the `mwUpdateAll` bit is set, it always passes flags of $0000.

- `DoModalWindow` in System 6.0 did not set a background window's origin to correspond with its scroll bars before redrawing it (if the `mwUpdateAll` was set). Background windows update correctly in System 6.0.1, even if they are scrolled.

- `DoModalWindow` no longer invalidates controls on an activate event if the window's `fCtlTie` bit is set, saying that the control states are independent of the window state.

- After `DoModalWindow` does an `LECut`, `LEPaste`, `TECut`, or `TEPaste`, it now returns the control ID of the control that was just edited.

### AlertWindow enhancements

- `ErrorWindow` and `AlertWindow` now use `UpdateWindow`, so that windows behind the alerts can redraw when the environment allows it.

## Clarifications of Previous Window Manager Documentation

- On page 291 of *Programmer's Reference for System 6.0*, the use of the `fFlex` bit is documented backwards. In fact, you should set `fFlex` when you don't want the system to provide an alert frame for you.

- On page 300 of *Programmer's Reference for System 6.0*, the return value for `FindCursorCtl` is incorrectly identified as a part code. It is actually just a word: zero if no control was found, or non-zero if a control was found.

- The description of the `windPtr` parameter to `FindCursorCtl` says you can pass NIL to find a control in the frontmost window. This is incorrect. NIL is invalid here.

## New Window Manager Calls

### UpdateWindow $6C0E

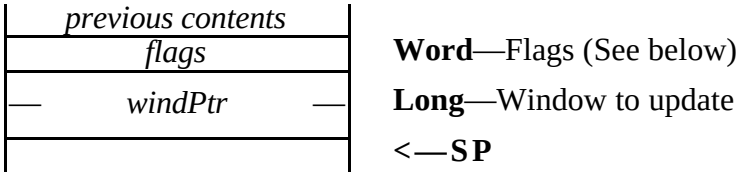
Updates the specified window, if it has a content draw routine and the environment permits. `UpdateWindow` is roughly equivalent to calling `BeginUpdate`, `StartDrawing`, the content-draw routine, `SetOrigin(0,0)`, `EndUpdate`.

Bits 31 and 30 in application window `wContDraw` pointers are significant. See the description under "New Features of the Window Manager."

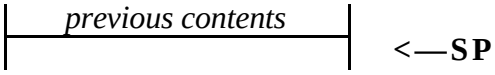
`UpdateWindow` never operates on a system window (one that has been flagged by `SetSysWindow`). System windows update during `GetNextEvent` calls (when `GetNextEvent` calls `SystemEvent`).

**Parameters**

Stack before call




Stack after call




**Errors**

| | | |
|---|---|---|
| `$0E05` | `cantUpdateErr` | Cannot update the specified window in the current environment. |

**C**

```
extern pascal void UpdateWindow (flags, windPtr);
WindowPtr windPtr;
Word flags;
```

`flags` Defined as follows:

bit 15 — 1 = Background update (window's expected environment may not already be set up). Requires the window's `wContDraw` bit 31 to be set for anything to happen, saying that the `wContDraw` routine can take care of itself.

bit 14 — Normally, applications do not need to worry about this bit, and should set it to zero.

0 = Allow `UpdateWindow` to try calling window update routines which need to make GS/OS calls. (`UpdateWindow` automatically checks whether GS/OS is busy; there is no need to set this bit just because GS/OS is busy.)

1 = Do not allow `UpdateWindow` to call window update routines that may need to make GS/OS calls, even if GS/OS is not busy.

bits 13-0 Reserved, use 0.

`windPtr` Window to update.

# Chapter 2 **GS/OS Changes**

## Device Dispatcher

- There is a new driver characteristics bit to allow Apple to safely add new $0000..$7FFF-range subcalls to `DStatus` and `DControl`.

- Drivers have always been required to validate all call requests that are sent to them, and return an error if they do not support the call. This includes the main driver commands as well as all subcalls. Unfortunately, some driver authors didn't feel that the guidelines needed to be followed, and they do random things if given a call they don't know about (some also crash).

- If bit 4 ($0010) is set in a device's characteristics word, the driver indicates that it properly follows the driver guidelines in the *GS/OS Reference* and *GS/OS Device Driver Reference*. If this bit is not set, drivers will never receive a call that is not documented in the GS/OS reference (except for device-specific calls, $8000..$FFFF, which are always passed on to drivers).

## System Loader

- File types $0030 to $003F (all auxiliary types) are now allowed for load files. File type/auxiliary type combinations must be assigned by Developer Technical Support, as usual.

## GS/OS Drivers

### SCSI HD Driver
### SCSI Tape Driver
### SCSI Scanner Driver

- Added support for the new Apple-defined `DStatus` subcall. Call $4000, `GetSCSITargetPriority`, returns a word indicating the SCSI ID (0..7) of a given GS/OS SCSI device. Use a `request_count` of $0002.

### SCSI CD Driver

- Added support for the new Apple-defined `DStatus` subcall. Call $4000, `GetSCSITargetPriority`, returns a word indicating the SCSI ID (0..7) of a given GS/OS SCSI device. Use a `request_count` of $0002.

- Changed default command bitmap to match the AppleCD 300, which does not return a command bitmap in the Inquiry call.

### Compatibility with the AppleCD 300 Drive

- With the 6.0.1 SCSI CD Driver, you can read data, but you can't play audio tracks, and you can't take advantage of the drive's ability to read audio tracks as data. With the 6.0 SCSI CD Driver, you can read data if and only if a CD was online when you booted. You can't play audio tracks.

  Under ProDOS 8 with the Apple High-Speed SCSI card, the AppleCD 300 works just as well as the previous models. You can't play audio tracks.

### Console Driver

- Fixed the driver to store the correct addresses in the fast-I/O vectors (as returned by the `DStatus` subcall `GetVectors`); they don't work in System 6.0. The lowest Console Driver version number where the fast-I/O vectors work is $3040.

### AppleDisk 3.5 Driver

- Fixed the `DStatus` subcall `get_format_options` to work correctly on an Apple SuperDrive connected to an Apple II SuperDrive Controller Card. Now it returns as much data as it can and returns a real transfer count. It used to return a transfer count of zero (and return no data) if the user's buffer was not big enough.

## FSTs

### AppleShare FST

- Fixed volume-changed notification to pass a valid device number.

- AppleShare volume-changed notifications now occur even if there are no directories open on the volume.

### DOS 3.3 FST

- The FST now ignores zero-length filename entries on DOS 3.3 disks. That is, `GetDirEntry` does not count them and does not return them. Internally, the FST treats a filename field of all blanks (30 $A0 bytes) just like a deleted catalog entry.

- `OpenGS` on a DOS 3.3 disk now works correctly with 15 parameters. In System 6.0 it did not work reliably.

- The FST name as returned from `GetFSTInfo` is "Apple II DOS 3.3" instead of just "DOS 3.3".

- Changed `Open`, `GetFileInfo`, and `GetDirEntry` to return zero for `resourceEOF` and `resourceBlocks` fields for non-extended files (when the fields are present).

### HFS FST

- Fixed a problem where `Write` to any HFS disk would stop prematurely after transferring 512 bytes if the most recent `Read` call to any HFS disk stopped because it hit a newline character (not because it transferred the requested number of bytes or hit the end of the file).

### HS.FST (High Sierra & ISO 9660 FST)

- The `FSTSpecific` subcalls `map_enable` and `set_map_table` now post volume-changed notifications for all of online High Sierra and ISO 9660 volumes.

- The FST can now use volumes with path tables larger than 8K.

## MS-DOS FST

The MS-DOS FST, new for System 6.0.1, is read only. (It does not modify MS-DOS disks, it just reads them.)

Resource forks are supported as defined and implemented by Macintosh PC Exchange. This is done by placing the resource fork of a given file into a normal MS-DOS file of the same name as the original file in a subdirectory named "RESOURCE.FRK" at the same directory level as the original file. The "RESOURCE.FRK" subdirectory does not appear to an application during a directory search on the Apple IIGS, though it does appear normally on an MS-DOS platform.

### GS/OS Calls Supported

The following lists all the GS/OS system calls supported by the MS-DOS FST. Those in bold type perform the indicated function, those in plain type will always return an error.

| Call # | Name | Call # | Name |
|---|---|---|---|
| $01 | Create | **$14** | **Close** |
| $02 | Destroy | **$15** | **Flush** |
| $04 | ChangePath | **$16** | **SetMark** |
| $05 | SetFileInfo | **$17** | **GetMark** |
| **$06** | **GetFileInfo** | $18 | SetEOF |
| **$08** | **Volume** | **$19** | **GetEOF** |
| $0B | ClearBackupBit | **$1C** | **GetDirEntry** |
| **$10** | **Open** | **$20** | **GetDevNum** |
| **$12** | **Read** | $24 | Format |
| $13 | Write | $25 | EraseDisk |
| | | **$33** | **FSTSpecific** |

### File Attributes

The MS-DOS file system stores a file attribute byte in the directory entry for each file. This is similar to the GS/OS access attributes. The FST translates the file attributes as follows:

| File Attribute | GS/OS Interpretation |
|---|---|
| Archive bit set | Bit 5 (the "backup" bit) of the access attributes will be set. |
| Subdirectory bit set | File type will be returned as $000F. |
| Volume Label bit set | Used internally by the FST to apply a volume name to the disk. |
| System File bit set | No special action. |
| Hidden File bit set | Bit 2 (the "Invisible" bit) of the access attributes will be set. |
| Read-Only File bit set | Bits 7, 1, and 0 (the Delete, Write-enable, and Rename) of the access attributes will be cleared (i.e., the file is "locked"). |

### File Types

MS-DOS does not provide a file typing mechanism. This is potentially very limiting since most applications select a particular file type as a filter when calling the standard file tools. Therefore, files from an MS-DOS disk would never be selectable.

The MS-DOS FST provides a partial solution to the problem. The FST searches a translation table for a matching file name extension. If it finds a match , it returns the associated file type and auxiliary type to the caller. For instance, the file "ABC.TXT" will normally be assigned a file type $04 (text) because of the suffix ".TXT". The MS-DOS FST maintains a table of suffixes and their

associated file types and auxiliary types. The `FSTSpecific` calls allow for modification of this table. The default table contains the following entries:

| Extension | File Type | Auxtype | Description |
|---|---|---|---|
| .TXT | $04 | $0000 | text file |
| .BAT | $04 | $0000 | batch file |
| .BIN | $06 | $0000 | binary file |
| .ASC | $04 | $0000 | ASCII text file |
| .C | $04 | $0000 | C language source code |
| .H | $04 | $0000 | C header file |
| .PAS | $04 | $0000 | Pascal language source code |
| .ASM | $04 | $0000 | assembly language source code |
| .LST | $04 | $0000 | listing file |
| .COB | $04 | $0000 | COBOL language source code |
| .FOR | $04 | $0000 | FORTRAN language source code |
| .DOC | $04 | $0000 | documentation file |
| .SRC | $04 | $0000 | source code file |
| .GIF | $C0 | $8006 | Graphics Interchange Format file |

### FSTSpecific ($33)

This call controls file type mapping by the MS-DOS FST. It is unique in that it uses a command number as one of its parameters and is actually four different calls.

Map_Enable

Enables or disables file type mapping. By default, mapping is enabled. The parameter block is as follows:

| | | |
|---|---|---|
| $00 | *pCount* | **Word**—Input value; must be $0003 |
| $02 | *file_sys_id* | **Word**—Input value; $000A (MS-DOS file system ID) |
| $04 | *command_num* | **Word**—Input value; $0000 (Map_Enable) |
| $06 | *enable* | **Word**—Input value; $0000 to disable, $0001 to enable |

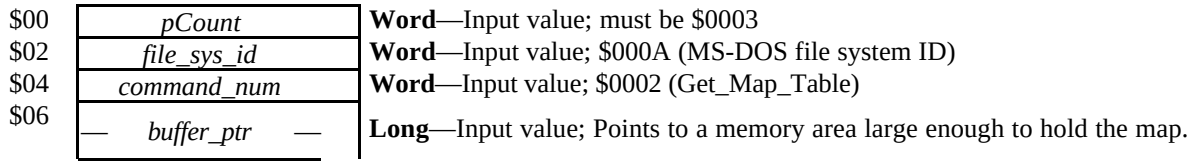
Get_Map_Size

Returns the size of the current map in bytes. The parameter block is as follows:

| | | |
|---|---|---|
| $00 | *pCount* | **Word**—Input value; must be $0003 |
| $02 | *file_sys_id* | **Word**—Input value; $000A (MS-DOS file system ID) |
| $04 | *command_num* | **Word**—Input value; $0001 (Get_Map_Size) |
| $06 | *map_size* | **Word**—Output value; size of the current map table in bytes. |

Get_Map_Table

Returns the current map. The parameter block is as follows:

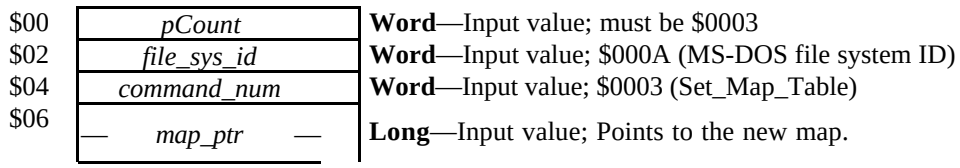



▲ **Warning** `Get_Map_Table` assumes the memory area pointed to by `buffer_ptr` is large enough to hold the map. If it isn't, bad things can happen. ▲

Set_Map_Table

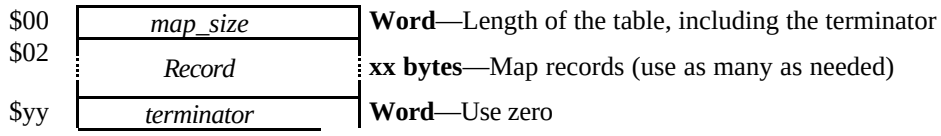
Changes the map. As long as there is space in memory for the new table, it will replace the old one. If there is not enough space, an `out_of_memory` error will be returned and the original table will remain in effect. No validity checking is done on the table.

The parameter block is as follows:




| **Errors** | $04 | `invalidPcount` | parameter count out of range |
|---|---|---|---|
| | $53 | `paramRangeError` | invalid parameter |
| | $54 | `outOfMem` | out of memory |

The format of a map table is as follows:




Map records consist of a text string followed by a zero byte followed by a file type byte and an auxiliary file type word. The text string can be any length and can include any legal characters for an MS-DOS file name (text must be upper case, for example).

## ProDOS FST

- Fixed a problem where `Read` did not work correctly when multiple newline characters were in effect. Each time it read into a new block of data it was forgetting about one more character from the end of the newline list, for the remainder of the `Read` call.

# Chapter 3 **Control Panels**

## Control Panels NDA 2.1

- Fixed the `cpOpenCDev` request to work reliably. In 6.0, it did not always work. (The handling of `finderSaysBeforeOpen` has always worked fine.)

  **Note** When you send `finderSaysBeforeOpen` (or `cpOpenCDev`), you should always pass a fully-expanded pathname (as the Finder does).

## Sound Control Panel

The Sound control panel accepts a new request code, `srqConvertRelPitch` ($8200), which converts a `relPitch` value into a `freqOffset` (suitable for use in a `FFStartSound` parameter block).

The request should be directed to the target string "`Apple~SoundCP~`".

Put the `relPitch` value into the low word of `dataIn` (the high word is reserved and should be zero). After the `SendRequest`, the resulting `freqOffset` word is in your four-byte `dataOut` buffer at offset +002.

# Chapter 4 **Finder 6.0.1**

## Clarifications

- The description of `askFinderAreYouThere` (on page 388 of *Programmer's Reference To System 6.0*) is incorrect. If the Finder is present, `SendRequest` will return no error, and `finderResult` field of `dataOut` will always be $0000. If the Finder is not present, `SendRequest` returns error $0120, `reqNotAccepted`, and `finderResult` is undefined (because the Finder was not around to return a value).

- The Finder now loads Finder Extensions from the @:FinderExtras folder. The @: prefix is not necessarily the same folder that the Finder is in: When a user boots from an AppleShare server, the Finder Extensions come from the user's User folder on the server.

## New Features of the Finder

- Finder accepts the `srqQuit` request, handling it just like a `tellFinderShutDown` with a selector of `kQuit`.

- Implemented the `geekPrefs` option for Shut Down default. Set the low two bits of the X2 word to 0, 1, or 2. (This word was semi-documented with 6.0, but it didn't do anything.)

### Icon Matching changes

- Fixed the `oneDoc` match types `matchCreateDateTime` and `matchModDateTime`. They did not work in 6.0.

- When matching an icon by filename, the case of the string in the Icon file or in the `rBundle` resource no longer matters. Previously, a string with a leading wildcard (like "*PAINT") would only work if all letters were uppercase, regardless of the case of any actual files being matched against.

- When matching an icon by filename and using a leading "*", the "*" can now match zero characters (it works like 5.0.4 again). In 6.0, a leading wildcard accidentally required at least one character to match.

### Finder Extension Changes

- When Finder sends out multiple `finderSaysBeforeOpen` requests (when several icons are opened at once), the modifiers are now correct for all of the requests, not just the first one. In 6.0, the modifiers were accidentally zero for all requests other than the first.

- `tellFinderGetSelectedIcons` (extended) now returns icon heights and widths. This was broken in 6.0 (the work around in *Programmer's Reference For System 6.0* sees that the Finder version is not 6.0 and automatically does nothing).

- Finder no longer forgets what menu title to unhighlight if an Extras menu handler calls `tellFinderMItemSelected` (even with no-highlight).

- Finder now sends a `finderSaysBeforeOpen` any time it launches an application. This means that you can now properly trap the Finder any time it was about to quit to another

application. (Finder 6.0 neglected to send a `finderSaysBeforeOpen` when the user chose the application using Standard File.)

- Added a tick count parameter (long) to `finderSaysKeyHit`. The parameter count is now three. The third parameter tells you the system tick count at the time the key was pressed.

- `tellFinderOpenWindow` had a bad exit path if `ExpandPath` returned an error. Now it works even if the pathname you pass in causes an error from `ExpandPath`.

- The parameter count for `finderSaysBeforeOpen` and `finderSaysOpenFailed` is now 7. (It was accidentally only 6 in 6.0. Oops. But all the parameters were there anyway.)

- For your convenience, `finderSaysIdle` now passes the idle ticks (as would be returned from `askFinderIdleHowLong`) as `dataIn`.

- You can now set bit 31 of `dataIn` on a `tellFinderSetSelectedIcons` call. This tells the Finder to deselect all selected icons before selecting the ones specified.

- `tellFinderSpecialPreferences` now returns a valid `finderResult` ($0000).

- `tellFinderRemoveFromExtras` validates the menu item number and returns error `fErrFailed` if it is invalid. (If you make the mistake of passing `itemID` 0 to Finder 6.0, it happily removes the first (remaining) item from the Apple menu!)

- Fixed `tellFinderAddBundle` so that passing a zero to grab the first available `rBundle` actually works (this case always failed with an error in 6.0).

# Chapter 5 **Battery RAM Update**

Two Battery RAM locations are defined for use with hierarchical menus. (The system does not provide support for hierarchical menus, but it defines these Battery RAM locations for consistency among applications that do.)

$65 Drag delay for hierarchical menus (number of ticks you can be out of the menu while moving diagonally before it “gives up” on you).

$66 Delay in ticks until a hierarchical menu pops up.

**U**



**W**