Chapter 28
Window Manager

Changes in Behavior

DeskTop Drawing

The window manager uses the same default desktop drawing scheme that the finder uses. When the window manager is started up, it looks in the message center for the DeskMessage. The DeskMessage has a message type of 2 and is in the following format:

Reserved	LONG	Used by message center.
MessType	Word 2	2 is the DeskMessage
DrawType	Word	0 = Pattern, 1 = Picture
DrawData	DATA	32 bytes of pattern data or 32000 bytes of picture data.

Task Master

Bits 16, 17, 18, 19 and 20 of the task mask are now defined. NOTE: If you use any of the new taskMask bits you must also be using the new extended task record described below, this record is longer than the previous Task Record and taskmaster will store data into them every time taskmaster is called. The task record is defined as follows:

Original Task Record		New Task Record	
wmWhat	WORD	wmWhat	WORD
wmMessage	LONG	wmMessage	LONG
wmWhen	LONG	wmWhen	LONG
wmWhere	LONG	wmWhere	LONG
wmModifiers	WORD	wmModifiers	WORD
wmTaskData	LONG	wmTaskData	LONG
wmTaskMask	LONG	wmTaskMask	LONG
		wmLastClickTick	LONG
		wmClickCount	WORD
		wmTaskData2	LONG
		wmTaskData3	LONG
		wmTaskData4	LONG
		wmLastClickPt	POINT

Bit 16 of the task mask is defined as tmContentControls. When the bit is set, TaskMaster will find and track controls in the content region of a window. TaskMaster does this by making a call to the internal routine TaskMasterContent after it finds the mouse down in the content region of a window. See TaskMasterContent below for details.

Bit 17 of the task mask is defined as tmControlKey. When this bit is set, TaskMaster will feed keystrokes to controls in the active window. TaskMaster does this by making a call to the internal routine TaskMasterKey, after it determines that an event is a key stroke. See TaskMasterKey below for details.

Bit 18 of the task mask is defined as tmControlMenu. When this bit is set, TaskMaster will feed menu selections to controls in the active window. TaskMaster does this by calling SendEventToControl. This way a control can get and handle the standard edit events without any extra work from the application. These include cut, copy, paste, clear, undo.

Bit 19 of the task mask is defined as tmMultiClick. When this bit is set, TaskMaster will keep track double and triple clicks.

Bit 20 of the task mask is defined as tmIdleEvents. When this bit is set, TaskMaster will send null events to the active control in the active window. This is used by DefProcs to maintain blinking cursors.

TaskMaster And Desk Accessories

In the past, Desk Accessories were not able to rely on TaskMaster to do any work for them since they need to run with applications that do not use TaskMaster. To let desk accessories use TaskMaster, we add a new entry point: TaskMasterDA. Desk accessories pass a task record to TaskMasterDA using information obtained from the Desk Manager. TaskMasterDA processes that data just like TaskMaster would if the event occured in an application window.

SizeWindow and ResizeWindow

SizeWindow and ResizeWindow make a NotifyCtls call whenever they change the window size. This call allows controls to be written that "stick" to the outer edge of the control frame. For example, an application using a grow control in the lower right corner of the window must move the control have sizing the window. With this call, the growControl defProc could be set up to move the control automatically.

AlertWindow

AlertWindows can also be created by referencing a resource. The input to the AlertWindow call that tells the type of strings are being used has been expanded.

Old Definition:

stringType	: word	0 if CString, 1 if Pascal string.

New Definition

AlertFlags	: word	Bit	Meaning
		0	0 = cString, 1 = pascal string
		1&2	00 = AlertStrRef is pointer
			01 = AlertStrRef is handle
			10 = AlertStrRef is resource ID
		3-15	Must be zero.

TaskMaster

The various task bits and return codes are not all documented in one place any more. This should fix the problem.

Task Master Result Codes

Name	Value	Description
Null	$0000	
mouseDownEvt	$0001	-Event Code -
mouseUpEvt	$0002	Event Code -
keyDownEvt	$0003	Event Code -
autoKeyEvt	$0005	Event Code -
updateEvt	$0006	Event Code -
activateEvt	$0008	Event Code -
switchEvt	$0009	Event Code -
deskAccEvt	$000A	Event Code -
driverEvt	$000B	Event Code -
app1Evt	$000C	Event Code -
app2Evt	$000D	Event Code -
app3Evt	$000E	Event Code -
app4Evt	$000F	Event Code -
wNoHit	$0000	alias for no event
inNull	$0000	alias for no event
inKey	$0003	alias for keystroke
inButtDwn	$0001	alias for button down
inUpdate	$0006	alias for update event
wInDesk	$0010	On Desktop.
wInMenuBar	$0011	On System Menu Bar.
wClickCalled	$0012	SystemClick called (returned only as action).
winContent	$0013	In content region.
wInDrag	$0014	In drag region.
wInGrow	$0015	In grow region, active window only.
wInGoAway	$0016	In go-away region, active window only.
wInZoom	$0017	In zoom region, active window only.
wInInfo	$0018	In information bar.
wInSpecial	$0019	Item ID selected was 250-255.
wInDeskItem	$001A	Item ID selected was 1-249.
wInFrame	$001B	In frame, but not on anything else.
wInactMenu	$001C	Inactive menu item selected.
wClosedNDA	$001D	Desk accessory closed (returned only as action).
wCalledSysEdit	$001E	SystemEdit called (returned only as action).
wTrackZoom	$001F	Zoom box clicked, but not selected (action only).
wHitFrame	$0020	Button down on frame, made active (action only).
wInControl	$0021	Button or keystroke in control (can be returned as event code and as action).
wInSysWindow	$8000	high bit set for system windows

Task Master TaskMask Flags

Name	Value	Bit	Description
tmMenuKey	$00000001	0	Calls MenuKey
tmUpdate	$00000002	1	Handles update events
tmFindW	$00000004	2	Call FindWindow for mouse down events.
tmMenuSel	$00000008	3	Call menuSelect when FindWindow returns inMenu.
tmOpenNDA	$00000010	4	Call OpenNDA when MenuSelect returns the ID of an NDA.
tmSysClick	$00000020	5	Call SystemClick when FindWindow returns button down in system window.
tmDragW	$00000040	6	Call DrawWindow when FindWindow returns button down in drag region.
tmContent	$00000080	7	Activate window on click in content region
tmClose	$00000100	8	Call TrackGoAway when FindWindow returns inClose.
tmZoom	$00000200	9	Call TrackZoom when FindWindow returns inZoom.
tmGrow	$00000400	10	Call GrowWindow when FindWindow returns inGrow.
tmScroll	$00000800	11	Enable scrolling and activate inactive windows when user clicks on scroll bar.
tmSpecial	$00001000	12	Handle special menu items (ids < 256)
tmCRedraw	$00002000	13	Redraw controls upon activate event.
tmInactive	$00004000	14	Return wInactMenu when user selects inacit
tmInfo	$00008000	15	Don't activate the window on click in the info bar.
tmContentControls	$00010000	16	Call FindControl & TrackControl when FindWindow returns inContent and window is already selected.
tmControlKey	$00020000	17	Pass key events to controls in the active window.
tmControlMenu	$00040000	18	Pass menu events to controls in the active window.
tmMultiClick	$00080000	19	Put multiclick information in the task record.
tmIdleEvents	$00100000	20	Pass IdleEvents to the active control in the active window.

New Calls

TaskMasterDA **Call $5F0E**

Input

Space	: WORD	
eventMask	: WORD	Not used
taskRecPtr	: LONG	Pointer to task record

Output

Result	: WORD	taskCode

This is the entry point used by desk accessories after the event is passed from the desk manager.

TaskMasterContent **Call $5D0E**

Input
private

Output
private

This is an internal entry point used to handle events inside the content region of a window. This is a separate call so that it can be easily patched.

How It Works. When TaskMasterContent gets control it does the following:

```
Zero TaskData2, TaskData3 and TaskData4
Call FindControl
Put resulting partCode into low word ofTaskData3
Put resulting ControlHandle into TaskData2
Put resulting ID into TaskData4

IF partCode <> 0 THEN
    CALL TrackControl with ActionProcPtr set to $FFFFFFFF
    Put resulting partCode into hiword of TaskData3
    if  partCode = 0 THEN
        Put wInControl into TaskData
        Return nullEvt ($0000)
    ELSE
        if the control is a check or radio box, Set or clear the value.
        Return wInControl
ELSE
    Return wInContent
```

The idea is that TaskMaster will call FindControl. If the mouse did not go down in a control, TaskMaster returns wInContent. If the mouse went down in a control, TaskMaster calls TrackControl and tells the control manager to use the action proc assciated with the control. When trackControl returns, TaskMaster looks at the partCode. A part code of zero indicates the user changed his mind (mouse came up too far from the control) and TaskMaster returns a result of nullEvent and puts wInContent into the taskData. If the part code is non-zero, the result is wInControl. TaskData2 contains the control handle for the case where the mouse went down in a control. Loword of TaskData3 contains the part code for the control at mouse down time. Hiword of TaskData3 contains the part code for the control at mouse up time. TaskData4 contains the control ID (if there is any).

TaskMasterKey Call $5E0E

Input
private

Output
private

This is an internal entry point used to handle events inside the content region of a window. This is a separate call so that it can be easily patched. An application should never make this call.

How It Works. When TaskMasterContent gets control it does the following:

```
if wmTaskMask bit tmMenuKey = 1 THEN
    Call MenuKey
    IF MenuKey indicates that a selection was made, we are done
    ELSE
        Zero TaskData2 and TaskData3
        Call SendKeyToControl
        Put result code into low word of TaskData3
        Put resulting ControlHandle into TaskData2
        Put resulting ID into TaskData4
        IF resultCode <> 0 THEN
            Put keyDownEvt into TaskData
            Return nullEvt (0000)
```

The idea is that TaskMaster first sees if the Menu Manager wants the key stroke. If the Menu Manager does not want it, TaskMaster looks for a control in the active window that wants the keystroke. If a control claims the event, TaskMaster puts keyDownEvt in TaskData and returns nullEvt. If no control wants the keystroke, TaskMaster returns keyDownEvt.

CompileText Call 600E

Input

HandleSpace	: LONG	space for resulting handle to string
SubType	: WORD	type of substitution strings: 0=cstrings, 1=pascal
SubStringsPtr	: LONG	Pointer to substitution array
SrcStringPtr	: LONG	Pointer to source text
SrcSize	: WORD	Size of source text

Output

StringHandle	: LONG	Handle to compiled string,handle size is size of text.

Possible Errors:
$0E04 - compileTooLarge - destination text is larger than 63k.

This call creates a handle to text starting from text in a source buffer and substituting special characters in the original text with strings from the substitution array or the standard substitution list. The calling routine is responsible for disposing of the handle after it is created. (No handle has been created if an error is returned.)

The substitution array is an array of 1 to 10 long pointers to strings. The strings can be either Pascal or C format. Any * character followed by an ASCII decimal digit (0-9) will be replaced with

the corresponding string from the substitution array. First string pointer in the array would replace *0 in the source string.

The # character followed by an ASCII decimal digit (0-6) will be replaced with a corresponding standard string.

#0 will be substitutes with OK
#1 will be substitutes with Cancel
#2 will be substitutes with Yes
#3 will be substitutes with No
#4 will be substitutes with Try Again
#5 will be substitutes with Quit
#6 will be substitutes with Continue

Both the # and * characters can be made part of the final compiled string by doubling them in the source string.

Error Window — Call $620E

Input

ResultSpace	: WORD	space for button number selected
SubType	: WORD	type of substitution strings: 0=cstrings, 1=pascal
SubStringPtr	: LONG	Pointer to substitution string
ErrNum	: WORD	Error message number

Output

StringHandle	: WORD	Button number selected

Possible Errors:
Resource manager errors are returned unchanged.

This call creates a dialog box containing the specified error message. Error numbers $0 through $71 correspond to GS/OS errors. These can be used by programmers during program development. The errors beyond $71 are more explicit and are used by system and toolbox routines.

The ErrorWindow call reads its messages from a resource file. The error messages have a resource type ID of $8020 and are of the form of an alert string. The resource ID has $07FF as its high word, and the error numbers (see the appendix) in the low word.

Descriptions of the messages in the system resource file (sys.resources) are given in the appendix. Since the system resource file is the last to be searched, it is possible to display custom error messages by opening another resource file containing $8020 resources with the proper ID numbers. If this is the case, the ErrorWindow call will automatically display the message from the current resource file.

NewWindow2 — Call 610E

This call is very similar to NewWindow except that the definition of the window is kept in a resource instead of at a particular address in memory.

Inputs:

HandleSpace	: LONG	space for result.
TitlePtr	: LONG	pointer to Pascal string for window title, NIL to use title in template.
RefCon	: LONG	any value the application desires.

ContentDrawPtr	: LONG	address of content draw routine, NIL if none.
DefProcPtr	: LONG	address of window defProc, NIL for standard.
ParamTableDescriptor	: WORD	indicate what inReference means
ParamTableRef	: LONG	see below.
ResourceType	: WORD	resource type of resource that is window parameter table template.

Output:

GrafPtrPtr	: LONG	pointer to window graphPort, NIL if window not created.

Possible Errors:
Resource Managers errors
Memory Manager error
Window Manager errors from NewWindow
Control Manager errors from NewControl2

The TitlePtr, RefCon, ContentDrawPtr, and DefProcPtr are all provided so that you can have newWindow2 modify the template areas before it calls NewWindow. This way, you can have a standard template and use these parameters to create different windows.

The InputVerb parameter describes what is being passed in the inReference parameter, they are as follows:

$0000 -	InReference is a pointer to a template
$0001 -	InReference is a handle to a template
$0002 -	InReference is the resource ID of the template.

NOTE: You should still use the appropriate ResourceType for the template you are using even if y are passing the template as a pointer or a handle.

Definition of window parameter table resource kind 1

Resource Type	$800E	rWindParam1

Resource Structure:

p1Length	WORD	Number of bytes in parameter list.	$004E
p1Frame	WORD	Type of window, frame flags.	$nnnn
p1Title	LONG	Reference to window's title.	$nnnnnnnn or NIL
p1RefCon	LONG	Reserved for application.	Replaced
p1ZoomRect	RECT	Size and position of zoomed window.	$nnnn...
p1ColorTable	LONG	Reference to window's color table.	$nnnnnnnn or NIL
p1YOrigin	WORD	Content's vertical offset.	$nnnn
p1XOrigin	WORD	Content's horizontal offset.	$nnnn
p1DataHeight	WORD	Height of total data area.	$nnnn
p1DataWidth	WORD	Width of total data area.	$nnnn
p1MaxHeight	WORD	Max grow height.	$nnnn
p1MaxWidth	WORD	Max grow width.	$nnnn
p1VerScroll	WORD	Amount to scroll vertically (arrows).	$nnnn
p1HorScroll	WORD	Amount to scroll horizonally (arrows).	$nnnn
p1VerPage	WORD	Amount to page vertically (page region).	$nnnn
p1HorPage	WORD	Amount to page horizontally (page rgn).	$nnnn
p1InfoText	LONG	Any value, passed to info draw routine.	$nnnnnnnn
p1InfoHeight	WORD	Height of infomation bar.	$nnnn
p1DefProc	LONG	Address of definition procedure.	Replaced
p1InfoDraw	LONG	Address of routine to draw Info Bar.	NIL
p1ContentDraw	LONG	Address of routine to draw content region	Replaced
p1Position	RECT	Size and position of window.	$nnnn...
p1Plane	LONG	Plane window should be in.	NIL or -1
p1ControlList	LONG	Reference to ControlList.	$nnnnnnnn or NIL
p1InVerb	WORD	describes the Reference fields above	$0000

P1InVerb describes the three reference fields p1Title, p1ColorTable, and p1ControlList.

P1Title can be a pointer, handle, resourceID, or NIL. If p1Title is NIL, then the template is assumed not to have a title in it. If you pass a titlePtr when you make the newWindow2 call, the p1Title field will be ignored completely. If you pass a pointer or a handle in p1Title newWindow2 will make a copy of the title and use the copy when creating the window. If you pass a resource ID in p1Title, newWindow2 will load an rPString resource with the given ID and use it as the windows title. To specify what p1Title contains set the appropriate bits (bits 8 and 9) in p1InVerb with one of the following:

titleAsPtr	$0000 - p1title parameter is a pointer to a title
titleAsHandle	$0100 - p1title parameter is a handle to a title
titleAsResource	$0200 - p1title is the resource ID of a rPString resource

NewWindow2 always copies the desired title into a handle for its own use, CloseWindow will dispose of this handle. Window titles that are handles can be differentiated from pointers by handles having their high bit (bit 31) set. All calls to the Window Manager that fetch or set window titles must conform to this convention. SetWTitle will also dispose of the exsisting title if it is a handle.

P1ColorTable can also be a pointer, handle, resourceID, or NIL. If P1ColorTable is NIL, then the template is assumed not to have a color table. If you pass a pointer or a handle in P1ColorTable newWindow2 will make a copy of the title and use the copy when creating the window. If you pass a resource ID in p1Title, newWindow2 will load an rWindColor resource with the given ID and use it as the windows color table. To specify what rWindColor contains set the appropriate bits (bits 10 and 11) in p1InVerb with one of the following:

ColorAsPtr	$0000 - P1ColorTable is a pointer to a color table

ColorAsHandle	$0400 - P1ColorTable is a handle to a color table
ColorAsResource	$0800 - P1ColorTable is the resource ID of an rWindColor resource

CloseWindow will dispose of the color table if it is a handle. SetFrameColor will dispose of the exsisting color table if it is a handle. Format of the rWindColor resource follows (NOTE: it is the same format as color tables required by NewWindow).

rWindColor $8010

Structure of the resource is the same as the window color table defined in Window Manager documentation.

frameColor	WORD
titleColor	WORD
tBarColor	WORD
growColor	WORD
infoColor	WORD

The p1ControlList parameter is passed directly to the newControl2 call, the lower 8 bits of p1InVerb are passed as the verb for that call. Please see the control manager chapter for information on the NewControl2 call.

NOTE: If you are creating a window template resource, and you want to include a title, color table, or control list, you may only pass references to resource ID's.

Definition of window parameter table resource kind 2

Resource Type $800F rWindParam2

Resource Structure:

p2ListID	WORD	Version of resource format.	NIL
p2DefProc	LONG	Address of window defProc.	Replaced
p2Data	BYTE[n]	Defined by version format.	

A valid defProc pointer must be passed to NewWindow2 when using the rWindParam2 template.

APPENDIX

NOTE: The descriptions of error messages beyond $71 in this appendix is preliminary.

The button numbers refer to the following:

0. OK
1. Cancel
2. Yes
3. No
4. Try Again
5. Quit
6. Continue
7. Replace
8. Initialize
9. Eject
10. Skip file
11. Erase
12. Shut Down

Error Number (hex) Message	Icon	Buttons
0. No error occured ($0).	No Icon	0
1. Bad system call number ($1).	No Icon	0
4. Invalid parameter count ($4).	No Icon	0
7. GS/OS already active ($7).	No Icon	0
10. Device not found ($10).	No Icon	0
11. Invalid device number ($11).	No Icon	0
20. Bad request or demand ($12).	No Icon	0
21. Bad control or status code ($21).	No Icon	0
22. Bad call parameter ($22).	No Icon	0
23. Character device not open ($23).	No Icon	0
24. Character device already open ($24).	No Icon	0
25. Interrupt table full ($25).	No Icon	0
26. Resources not available ($26).	No Icon	0
27. I/O Error ($27).	No Icon	0
28. Device not connected ($28).	No Icon	0
29. Driver is busy and not available ($29).	No Icon	0
2B. Device is write protected ($2B).	No Icon	0
2C. Invalid byte count ($2C).	No Icon	0
2D. Invalid block number ($2D).	No Icon	0
2E. Disk has been switched ($2E).	No Icon	0

2F. Device off line/no media present ($2F).	No Icon	0
40. Invalid pathname syntax ($40).	No Icon	0
43. Invalid reference number ($43).	No Icon	0
44. Subdirectory does not exist ($44).	No Icon	0
45. Volume not found ($45).	No Icon	0
46. File not found ($46).	No Icon	0
47. Duplicate path name ($47).	No Icon	0
48. Volume full ($48).	No Icon	0
49. Volume directory full ($49).	No Icon	0
4A. Version error ($4A).	No Icon	0
4B. Bad storage type ($4B).	No Icon	0
4C. End of file encountered ($4C).	No Icon	0
4D. Position out of range ($4D).	No Icon	0
4E. Access not allowed ($4E).	No Icon	0
4F. Buffer too small ($4F).	No Icon	0
50. File is already open ($50).	No Icon	0
51. Directory error ($51).	No Icon	0
52. Unknown volume type ($52).	No Icon	0
53. Parameter out of range ($53).	No Icon	0
54. Out of memory ($54).	No Icon	0
57. Duplicate volume name ($57).	No Icon	0
58. Not a block device ($58).	No Icon	0
59. Specified level is outside legal range ($59).	No Icon	0
5A. Block number too large ($5A).	No Icon	0
5B. Invalid pathnames for change_path ($5B).	No Icon	0
5C. Not an executable file ($5C).	No Icon	0
5D. Operating system no supported ($5D).	No Icon	0
5F. Stack overflow ($5F).	No Icon	0
60. Data unavailable ($60).	No Icon	0
61. End of directory has been reached ($61).	No Icon	0
62. Invalid FST call class ($62).	No Icon	0

63. File does not contain requested resource ($63).	No Icon	0
64. Specified FST is not present in system ($64).	No Icon	0
65. FST does not handle this type of call ($65).	No Icon	0
66. FST handled call, but result is weird ($66).	No Icon	0
67. Internal error ($67).	No Icon	0
68. Device list is full ($68).	No Icon	0
69. Supervisor list is full ($69).	No Icon	0
70. Cannot expand file, resource already exists ($70).	No Icon	0
71. Cannot add resource fork to this type of file ($71).	No Icon	0
80. Error creating the new directory: (reason substitution)	Stop	1
81. Error saving the file: (reason substitution)	Stop	1
82. Insufficient access privaledges to open that folder.	Stop	0
83. The selected folder cannot be opened: (reason substitution)	Stop	1
84. You cannot replace a folder with a file.	Stop	1
85. That file already exists.	Stop	1,7
86. Insufficient memory to perform that operation. About (byte substitution)K additional needed.	Stop	1
87. Initialization failed: Disk write protected.	Stop	1
88. The pathname is too long.	Stop	0
89. The disk is write protected.	Caution	1
8A. The disk is full.	Stop	1
8B. The disk directory is full.	Stop	1
8C. The file is copy-protected and can't be copied.	Stop	1
8D. Memory is full.	Stop	0
8E. There isn't enough memory remaining to complete this operation. Please close some windows and try again.	Stop	0
8F. The item is locked and can't be renamed.	Stop	1
90. An I/O error has occurred while using the disk.	Stop	1
91. This disk seems to be damaged.	Stop	1
92. Not a ProDOS disk.	Stop	0
93. No on-line volumes can be found.	Stop	0

94. Insert the disk (disk name substitution). Swap 1