Chapter 4
Control Manager

New Controls

The Control manager has been modified to provide support for more standard control types. In addition to the original standard controls (SimpleButton, CheckBox, Radio Button, ScrollBar, GrowBox) we have added many new controls that provide further flexability of this tool. In addition to these changes, taskmaster has been modified to act on control hits in windows (see the window manager chapter). The combination of the TaskMaster changes and the new controls makes it possible to easily do everything in a window that was possible in a dialog box.

The new controls are:

StatTextControl	This control allows you to display text messages in a given rectangle with word wrap and styling. Support built into Control manager (NOTE: Requires LineEdit tool set)
PictureControl	This control allows you to have a picture drawn into a given rectangle on the screen. Support built into Control manager (NOTE: Requires QDAux tool set)
IconButtonControl	This control allows you to have an icon drawn into a given rectangle on the screen. Support built into Control manager (NOTE: Requires QDAux tool set, and a SYS.RESOURCES file in the SYSTEM.SETUP directory.)
LineEditControl	This control works just like a standard line edit item with the added benefit of being able to control it through the list manager. Supported through the LineEdit tool set.
TextEditControl	This control works with the new Text Edit tool (see chapter 25). It allows creation of text editing rectangles. Supported through TextEdit tool set.
PopUpControl	Provides multiple selection buttons with scrolling lists.Supported through Menu Manager tool set.
ListControl	Standard list manager lists as before, but now you can use the Control manager to manipulate them. Supported through List Manager tool set.

The StatTextControl uses the LineEdit toolset function LETextBox2 to draw its text, so you must have Line Edit started before you use a control of this type. The PictureControl uses QDAux to draw the picture you provide, so you must start QDAux before you use a PictureControl.

New functions have been added to the Control manager to support these new control types. To create any of the new controls (as well as creating any of the old controls) use the NewControl2 call. Here is a more in depth description of each new control.

StatTextControl

Creating It. This is a control that can put static text in a window. You specify the rectangle that is to contain the text and a reference to the text when you create the control. The text is drawn every time the control is drawn. Since the text is drawn with LETextBox2, you can imbed font, style, size and color changes into the text. This lets you draw text in a wide variety of formats.

Finding and Tracking It. Normally, a click in this kind of control is ignored, but sometimes you may want to know that a user has clicked in the control. FindControl will ignore clicks in any control with the hilite field set to $FF. If you want to know when the mouse is clicked in a static text control, set the hilite field to 0, otherwise set it to $FF. Even when the control is active (hilite set to 0), it will not be drawn any differently.

Special Features. Static text controls can be designed to allow text substitution before they are drawn. To use this feature, you must set the appropriate bit in the control's flag word and make the call to set the control managers pointer to the substitution array.

PictureControl

Creating It. This control can put any QuickDraw picture into a window. You specify the rectangle that is to contain the picture and a reference to the picture when the control is created. The picture is drawn every time the control is drawn.

Finding and Tracking It. Normally, a click in this kind of control is ignored, but sometimes you may want to know that a user has clicked in the control. FindControl will ignore clicks in any control with the hilite field set to $FF. If you want to know when the mouse is clicked in a picture control, set the hilite field to 0, otherwise set it to $FF. Even when the control is active (hilite set to 0), it will not be drawn any differently.

Special Features. The drawPicture call is made with the control rect as the picture destination rectangle. This means that the picture could be scaled if the original picture frame was not the same dimensions as the destination rectangle. If the original control rectangle passed to NewControl2 has a top right corner of 0, the rectangle is adjusted when the control is initialized to be the size of the picture frame. This results in a picture that is not scaled.

IconButtonControl

Creating It. This control can put an icon into a window. You specify the rectangle that is to contain the icon and a reference to the icon when the control is created. The icon is drawn every time the control is drawn.

Finding and Tracking It. This control if similar to a simple button control except an icon as well as text can be placed inside the control rectangle. The difference with this control is that the control rectangle is inset slightly before the button is drawn. In this way an outlined round button stays within the control rectangle (which is not the case for an outlined round simple button). The icon is not clipped to the control rectangle, and if it is larger the erase control call will not erase that part outside the rectangle.

Special Features. The icon and the text are centered in the control rectangle. If control consists of only an icon, the icon is centered. If there is no icon, the control behaves like a simple button control.

LineEditControl

Creating It. This control lets you type single line items in a window. You specify the rectangle that is to contain the line of text, the maximum number of characters it can hold, and an initial value. The text is updated every time the control is drawn.

Finding and Tracking It. This control responds to mouse events. When the mouse is pressed in a LineEditControl, calling track control will track the mouse and do the appropirate text selection. (TaskMaster does this for you if the tmContentControls bit is set in the task mask.)

Special Features. This control also responds to keyboard events. Keyboard events are sent to the control with a SendEventToControl(AllControls,FrontWindow,TaskRecord) call, after first checking to see if the key represents a menu key. (TaskMaster does this for you if the tmControlKey bit is set in the task mask.)

This control also needs to be sent idle events to keep the carot flashing. You do this with a SendEventToControlCall (ActiveOnly,FrontWindow,TaskRecord) call. (TaskMaster does this for you if the tmIdleEvents bit of the task mask is set.)

The LineEdit record handle is kept in the ctlData field of the control record. When you need to make direct line edit calls, you can retrieve the handle from here.

TextEditControl

This is the control built and maintained by the TextEdit Tool Set. Please see the TextEdit documentation for more details.

PopUpControl

Creating It. This control lets you have popup menus in a window. You specify the bounding box of the popup menu, a reference to a menu, an initial value and a number of flags. The menu title becomes the title of the control, the current selection is defined by the intial value. All these are drawn whenever the control is drawn.

Finding and Tracking It. This control responds to mouse events. When the mouse is pressed in a PopUpControl, calling track control will track the mouse and bring up the menu. (TaskMaster does this for you if the tmContentControls bit is set in the task mask.)

Special Features. This control also responds to keyboard events. Keyboard events are sent to the control with a SendEventToControl(AllControls,FrontWindow,TaskRecord) call, after first checking to see if the key represents a menu key. (TaskMaster does this for you if the tmControlKey bit is set in the task mask.)

The CtlValue field of the control record contains the current selection of the pop up menu. Any time the user has used the popup menu, you can obtain his/her selection by reading this value.

ListControl

This is the control built and maintained by the list manager. Please see the list manager documentation for more details.

Changes to Existing Controls

Grow Control

If bit 0 of the flag is set, the track routine will call growWindow.

SimpleButtons, RadioButtons, and CheckBoxes

These three controls can support keystroke equivalents. When a key is pressed that is the control's equivalent, the control is hilited and unhilited. This is handled automatically by the defProc and taskMaster if the control is created as a super control (with NewControl2) and its moreFlags field has the FctlWantsEvents bit set.

Four fields in the control record control the keyboard equivalents. They are

Key1	byte	This is the upper or lower case of the character equivalent.
Key2	byte	This is the lower or upper case of the character equivalent.
Modifiers	word	These are the modifiers that must be set. The word has the same format as the event manager's modifiers word in an event record. Only the modifiers in the high byte are used.
CareBits	word	These are the modifiers that must match. They include the keys that should not be pressed. For example if Command-7 should be an equivalent, but Option-Command-7 should not, then the bit for the option key should be set in the care Bits word. If return and enter should be treated the same, the bit for the keypad should be clear.

These fields are kept in the control record starting at the ctlReserved field described below. They are set in the control record from the template that defines the control.

Control Records

The current control manager's standard control record is $28 bytes. This is extended so that standard controls can contain additional information.

In order to safely indicate whether a control is using this new standard record, the highest bit of the ctlProc field must be set by all DefProcs that use this new record. This change was necessary to prevent the control manager from misinterpreting data in the control record of custom defProcs that are part of existing applications.

The old and new records are shown below:

ctlNext	LONG	ctlNext	LONG
ctlOwner	LONG	ctlOwner	LONG
ctlRect	RECT	ctlRect	RECT
ctlFlag	BYTE	ctlFlag	BYTE
ctlHilite	BYTE	ctlHilite	BYTE
ctlValue	WORD	ctlValue	WORD
ctlProc	LONG	ctlProc	LONG
ctlAction	LONG	ctlAction	LONG
ctlData	LONG	ctlData	LONG
ctlRefCon	LONG	ctlRefCon	LONG
ctlColor	LONG	ctlColor	LONG
		ctlReserved	BLOCK $10
		ctlID	LONG
		ctlMoreFlags	WORD
		ctlVersion	WORD

There are new Control Manager calls to set and get the ctlID, ctlType and ctlMoreFlags fields.

The ctlMoreFlags field has 16 bits. The lower eight bits are reserved for the individual defProc. The upper eight bits are reserved for the control manager. The following control manager bits are definded.

FctlActive	equ $8000 -	if set means this control is the currently active control
FctlCanBeActive	equ $4000 -	when set means that this control can be made the active control.
FctlWantEvents	equ $2000 -	when set, this control can be called when events are passed via the SendEventToCtl call.
FctlProcRefNotPtr	equ $1000 -	When set, the ProcRef field contains an ID of a defproc routine, when clear the ProcRef field contains the pointer to the defproc routine
FctlTellAboutSize	equ $0800 -	Set if this control needs to be notified when the size of the owning window has changed.

In the lower 8 bits the following convention is used to describe references to titles and color tables. Simple buttons, check boxes, radio buttons describe their title references and color table references this way. Scroll bars and grow box controls describe their color tables this way. LineEdit and StatText controls reference their intial values the same way titles are referenced in buttons.

Bits 0 and 1 describe title reference:

TitleIsPtr	equ %00000000
TitleIsHandle	equ %00000001
TitleIsResource	equ %00000010

Bits 2 and 3 describe the color table reference:

ColorTableIsPtr	equ %00000000
ColorTableIsHandle	equ %00000100
ColorTableIsResource	equ %00001000

Control DefProcs

Original DefProcs supported 12 calls. Control DefProcs with new control records will support additional calls.

Value	Message	Description
13	ctlHandleEvent	This message is sent to the defProc to handle key strokes and menu selections.
14	ctlChangeActivity	This message is sent to the defProc of controls that are being made active or inactive.
15	ctlChangeBounds	This message is sent to the defProc when the bounds rect of the control is to change.
16	ctlWindChangeSize	This message is sent to the defProc to indicate the window has been grown or zoomed.
17	ctlHandleTab	This message is sent to the defProc to indicate that the control was tabbed to.

In addition, the inputs to two other messages have changed.

Value	Message	Description
3	initCtl	This message is sent to the defProc to init the control and control record. In the past, Param contained the values param1 and param2 passed to NewControl. Now, the param field is the same if the control was created by NewControl. If the control was created by NewControl2, param contains the pointer to the control template. A DefProc can tell which call was made (NewControl vs NewControl2) by looking at the ctlProc field of the control record. The highest bit will be set if the NewControl2 was used.
7	dragCntl	This message is sent to the control to drag part or all of a control. (The whole control is dragged when DragControl is called, a part is dragged when TrackControl is called after FindControl returned a part code greater than $80.) The result code has been extended for super controls. In the past, a zero result indicated that the control is to be dragged in the standard way by the control manager. A non-zero result indicated the the dragging is done and control should return to the calling application. For super controls, the low word of the result still serves the above function. In addtion the high word is used to return the part code. This change lets a control defProc abort tracking. For example, the popUp menu control returns a part code of zero if the mouse is released outside the pop-up menu.
12	recSize	This message is sent to the defProc to find out how big the control record should be. In the past, Param was undefined. Now it contains a pointer to the control template or zero depending on how the control is made.

Control Titles

Three controls have titles: buttons. check boxes and radio buttons. The current control manager keeps pointers to the titles of these controls and the application has to keep the titles somewhere in fixed memory.

Forcing the application to keep these titles in fixed locations limits the usefulness of the resource manager so we have changed the control manager to make it responsible for keeping the titles. Existing applications will continue to work the way they do, but it will be possible using the new call NewControl2, to refer to titles in three different ways:

Pointer to string
Handle of string
Resource of string

This flexibility makes it possible for the application to free the space devoted to a string after the control is created.

Control Color Tables

All controls have color tables. The current control manager keeps pointers to these color tables and the application has to keep the tables somewhere in fixed memory.

Forcing the application to keep these tables in fixed memory limits the usefulness of the resource manager so we have changed the control manager to make it responsible for keeping the tables. Existing applications will continue to work the way they do, but it will possible using the new NewControl2, to refer to color tables in three different ways:

Pointer to color table
Handle of color table
Resource of color table

This flexibility makes it possible for the application to free the space devoted to a color table after the control is created.

Control and Window Frames

Some controls need to be adjusted when the window is resized. These include scroll bars and grow boxes. Every time the window changes size, they need to be moved. The window manager has been changed to call NotifyCtls whenever the window size changes. For a control to get the ctlWindChangeSize message, it must set the FctlTellAboutSize bit of the MoreFlags field.

Controls, Keystrokes and Menu Selections

It will now be possible for controls to respond to key strokes and menu selections. There are two kinds of controls that respond to keystrokes: edit field controls and others. The edit field controls are fields that you type into. Other controls do not accumulate text but rather occaisionly have keystroke equivalents. Examples of this are a default button that is assumed to be hit when the user pressing return and a cancel button that is assumed to be hit when ESC or Apple-Period are pressed.

Text editing controls need to be able to respond to standard menu selections: cut, copy, paste, clear and undo. Other controls may also want to respond to menu selections.

Controls handle key strokes and menu selections via their DefProcs and the ctlHandleEvent message. When the DefProc gets control the stack would appear as follows:

Stack before call to DefProc

resultSpace	LONG	space for long result
message	WORD	ctlHandleEvent message
param	LONG	Pointer to Extended task record
ctlHandle	LONG	handle to control

Stack after call to defProc

Result	LONG	high word 0. low word 0 if event not accepted, non-zero otherwise

An application would send key strokes to a control with the new SendEventToControl call. This call is described in detail below.

A window can contain more than one edit field control. When this happens, only one control is active at a time. The active control is the one that receives the key strokes. It is up to the defProc to activate itself during drag events.

Resource Use For Data

A number of parts of the control manager will use resources for data. This data can be temporary (as is the case for control templates) or more permanent (as is the case for title strings or pictures). The following philosophy describes how the resources will be used:

Temporary Information:

The resource will be loaded with LoadResource when it is needed. The data in it will be used and the resource will be freed with ReleaseResource.

Permanent Information:

The resource will be loaded every time it is accessed (with LoadResource). The resource will only be freed (with ReleaseResource) when the control is killed. Resources of this type should have the following attributes:

Locked	No
PurgeStat	$2

This way the resource can be purged any time the memory is needed. The resource will remain in memory if the memory is not needed. Under these circumstances, accesses to the resource will be fast.

Resource Use For Code

Control DefProcs can be kept as resources. The control manager will be able to load them as they are needed.

DefProcs in code resources will be loaded each time they are accessed and shutdown into zombie state when they are freed.

New Calls

NewControl2 Call $3110

This new control manager call lets you create one or more controls from a variety of templates. The Verb determines what the inputs to the call look like.

If the verb indicates that a single control is being defined, then the result is the handle to the control. If the verb indicates that multiple controls can be defined, then the result is zero.

The Control Manager will allocate new style control records for all controls created with NewControl2.

Input			
	Space	: LONG	
	OwnerPtr	: LONG	Ptr to control's window
	InputVerb	: WORD	indicates what next long means (see below)
	InputReference	: LONG	Pointer, Handle, ID depending on verb
Output			
	Result	: LONG	Control Handle or Zero

Possible values for InputVerb

singlePtr	0	InputReference is pointer to single ItemTemplate
singleHandle	1	InputReference is handle to single ItemTemplate
singleResource	2	InputReference is Resource ID of single ItemTemplate
ptrToPtr	3	InputReference is pointer to a list of Item Template Pointers
ptrToHandle	4	InputReference is pointer to a list of Item Template Handles
ptrToResource	5	InputReference is pointer to a list of Resource ID's of Item Templates
handleToPtr	6	InputReference is handle to a list of Item Template Pointers
handleToHandle	7	InputReference is handle to a list of Item Template Handles
handleToResource	8	InputReference is handle to a list of Resource ID's of Item Templates
resourceToResource	9	InputReference is resource id of a list of Resource ID's of Item Templates

If the input reference points to a list, the list is of the following format:

Template1Ref
Template2Ref
...
TemplateNRef
0

And of course the template references can be pointers, handles or resource ids.

Each control has its own kind of item Template. All item templates start with the same information. Some controls will have optional fields in their templates. To support this, every template starts with a pCount field. Controls with optional fields will ignore the optional fields if the pCount indicates that they are not there.

Any fields that are missing, but are needed will generate errors.

All templates have the following 7 fields. Specific templates for individual controls are defined later.

pCount	WORD	Number of fields following
ID	LONG	Application Assigned ID
Rect	RECT	Bounding rect for control
ProcRef	LONG	LONG refering to defProc: See details below
Flag	WORD	Control Bit Flags (depends on control type)
MoreFlags	WORD	Additional Bit Flags (depends on control type)
RefCon	LONG	Programmer Defined RefCon

The ProcRef is either a pointer to a DefProc routine or the ID of a DefProc routine. The FctlProcRefNotPtr bit in the MoreFlags field determines what it is. Predefined Control ProcRefs are

Name	Value
simpleButtonControl	$80000000
checkControl	$82000000
radioControl	$84000000
scrallBarControl	$86000000
growControl	$88000000
statTextControl	$81000000
editLineControl	$83000000
editTextControl	$85000000
popUpControl	$87000000
listControl	$89000000
iconButtonControl	$8B000000
pictureControl	$8D000000

These are used with the FctlProcRefNotPtr bit of the MoreFlags field set to 1.

The flag field of each template is defined as follows:

Bits 0 through 6 depend on control type.

Bit 7 of Flag indicates whether the control is invisible or not. Clear means visible, set means invisible.

Bits 8-15 of Flag are used to indicate hi-liting style.

0	Control Active, no highlighted parts
1-254	part code of highlighted part
255	control inactive.

The general bits of the moreFlags field are defined above (in the section about the control record).

The individual templates for each of the 13 control types are given in the Appendix.

FindActiveCtl Call $2610

Input
Space : LONG

Output
Result : LONG Control Handle

This routine searches the control list of the active window for first active control that it finds. An active control is a control with the active bit of the moreFlags field set.

The handle to the active control is returned unless an error occurs and then the result is NIL.

Possible Errors:

NoWindError	$1001
NoCtlError	$1004
NoSuperCtlError	$1005
NoCtlActiveError	$1006

MakeNextCtlActive Call $2710

Input
Space : LONG

Output
Result : LONG Control Handle

This routine searches the control list of the active window for first active control that it finds. It makes this control inactive and then searches for the next control that can be active (as indicated by the FctlCanBeActive bit) and makes that one active.

The handle to the next active control is returned.

Both the newly activated control and the previously active control are sent ChangeActivity messages.

A number of errors can occur. If an error is returned, the resulting handle is NIL.

Possible Errors:

NoCtlError	$1004
NoSuperCtlError	$1005
NoCtlToBeActiveError	$100B

MakeThisCtlActive Call $2810

Input
CtlToBeActive : CtlHandle (LONG)

Output
none

This routine makes the indicated control, the active control.

Both the newly activated control and the previously active control are sent ChangeActivity messages.

This can work if the window is not active (or invisible).

Possible Errors:

NotSuperCtlError	$1007
CanNotBeActiveError	$1008

CallCtlDefProc

Call $2C10

Input

CtlHandle	: LONG
Message	: WORD
Param	: LONG

Output

Result	: LONG

This routine calls the indicated control with the indicated message and parameter value. Before the defProc is called, the drawing environment is set up so that the defProc can safely draw if necessary.

NotifyCtls

Call $2D10

Input

Mask	: WORD	
Message	: WORD	
Param	: LONG	
Window	: LONG	GrafPort of window

Output

none

This routine calls the defProc for each of the super controls in the specified window whose moreflags field has any of the bits specified in Mask set. The mask field is ANDed with the moreFlags field and if the result is zero, no call to the defProc is made. If the result is not zero, the defProc is sent the indicated message.

Before any defproc is called, the drawing environment is set up. This means that it is safe for the defProc to draw in response to the message.

Possible Errors:

None

SendEventToCtl Call $2910

Input

Space	: WORD	
ActiveOnlyFlag	: WORD	True => send only to active ctl, false => all
WindowPtr	: LONG	Ptr to window with controls/NIL for top window
ExtendedTaskRecPtr	: LONG	Ptr to event or task record containing key info

Output

Boolean Result	: WORD	True if event accepted, False otherwise

This call passes the indicated task record to the apropriate control/controls. NOTE: you may only use this call with the new task record defined in the window manager section.

The way that the event is sent depends on the ActiveOnly flag. If the active only flag is set, the event is sent to the active control. If the active only flag is clear, a more complicated search takes place.

The control manager performs a two part search for a control to take the event. First it looks for controls that want key strokes that are not edit fields. Each control is asked to take the event. If no control accepts the event, the second stage begins: the active edit field is asked to accept the event.

If no control accepts the event, the output is False. If one of the controls accepts the event, the output is True. When the output is true, the handle to the accepting control is put in TaskData2.

If no active control is found, the result is false and the TaskData2 field is undefined.

Possible Errors:

NoWindError	$1001
NoSuperCtlError	$1005

GetCtlID Call $2A10

Input

Space	: LONG
ControlHandle	: LONG

Output

ControlID	: LONG

This call returns the ControlID field from the specified control record. If the specified control is not a super control, the resulting ID is 0 and an error is returned.

Possible Errors:

NoCtlError	$1004
NotSuperCtlError	$1007

SetCtlID **Call $2B10**

Input
- NewID : LONG
- ControlHandle : LONG

Output
- none

This call sets the ControlID field in the specified control record. If the specified control is not a super control, an error is returned.

Possible Errors:

NoCtlError	$1004
NotSuperCtlError	$1007

GetCtlMoreFlags **Call $2E10**

Input
- Space : WORD
- ControlHandle : LONG

Output
- ControlID : WORD

This call returns the ctlFlags field from the specified control record. If the specified control is not a super control, the resulting flags are 0 and an error is returned.

Possible Errors:

NoCtlError	$1004
NotSuperCtlError	$1007

SetCtlMoreFlags **Call $2F10**

Input
- NewID : WORD
- ControlHandle : LONG

Output
- none

This call sets the MoreFlags field in the specified control record. If the specified control is not a super control, an error is returned.

Possible Errors:

NoCtlError	$1004
NotSuperCtlError	$1007

GetCtlHandleFromID **Call $3010**

Input

Space for ResultHandle	: LONG
WindowPtr	: LONG
Control ID	: LONG

Output

Handle	: LONG

This call returns the handle of the control with the specified ID in the active window. If an error occurs, a NIL is returned.

Possible Errors:

NoWindError	$1001
NoCtlError	$1004
NoSuperCtlError	$1005
NoSuchIDError	$1009

SetCtlParamPtr **Call $3410**

Input

SubArrayPtr	: LONG

Output

none

This call sets the pointer to the control manager's substitution array. The control manager uses this pointer when displaying static text controls that allow for text subsitution.

Desk Accessories should be careful when using this feature to save and restore the value around its use.

The substituion array has the same format as the one used in AlertWindow. It is an array of up to 9 pointers:

PtrToString1 LONG
PtrToString2 LONG
...
PtrToString9 LONG

GetCtlParamPtr **Call $3510**

Input

Space for ptr	: LONG

Output

Ptr to Sub Array	: LONG

This call returns the pointer to the control manager's substitution array. The control manager uses this pointer when displaying static text controls that allow for text subsitution.

Desk Accessories should be careful when using this feature to save and restore the value around its use.

CMLoadResource **Call $3210**

Input

Space handle	: LONG
Res Type	: WORD
Res ID	: LONG

Output

ResourceHandle	: LONG

This is an entry point to the internal control manager routine to load resources. This entry point is provided so that all defProcs can use the same error handling routine. Any errors that occur during resource load lead to system death.

Possible Errors:

None

CMReleaseResource **Call $3310**

Input

Res Type	: WORD
Res ID	: LONG

Output

none

This is an entry point to the internal control manager routine to release resources. This entry point is provided so that all defProcs can use the same error handling routine. Any errors that occur during resource release leads to system death. The resource is released by making it purgable.

Possible Errors:

None

InvalCtls **Call $3710**

Input

WindowPtr	: LONG

Output

none

This call invalidates all the rectangles for all the controls in the specified window.

Possible Errors:

None

Summary of Errors

NoWindError	$1001	There is no front window
NoCtlError	$1004	no controls in window
NoSuperCtlError	$1005	no super controls in window
NoCtlActiveError	$1006	no active super control
NotSuperCtlError	$1007	action can only be done on super control
CanNotBeActiveError	$1008	This control cannot be made active
NoSuchIDError	$1009	The specified ID cannot be found
TooFewParmsError	$100A	There were not enough parameters specified.
NoCtlToBeActiveError	$100B	Tried to make next ctl active, but no ctls could be active.

Control Templates

customControl

pCount	WORD	At least 6
ID	LONG	Application Assigned ID
Rect	RECT	Bounding rect for control
ProcRef	LONG	Reference to custom control defProc
Flag	WORD	See below.
MoreFlags	WORD	
RefCon	LONG	Programmer Defined RefCon
defProcSpecific	DATA	

Flag

ctlInvis	bit7	1 = Invisible, 0 = visible
	bits 0-6	defined by control

MoreFlags

Bits 0 through 7: defined by control

simpleButtonControl

pCount	WORD	valid values= 7, 8, 9
ID	LONG	Application Assigned ID
Rect	RECT	Bounding rect for control
ProcRef	LONG	simpleButtonControl = $80000000
Flag	WORD	See below
MoreFlags	WORD	See below
RefCon	LONG	Programmer Defined RefCon
TitleRef	LONG	Pointer, Handle, or ResourceID to title
*ColorTableRef	LONG	Optional Ptr, Handle or ResID to color table
*KeyEquivalent	block 6	Optional Key equivalent information

Flag

ctlInvis	Bit7	1=invisible, 0=visible
Reserved	Bits2-6	Must be zero
ButtonType	Bits0-1	Describes button type: 0 single-outlined round-cornered button 1 bold-outlined round-cornered button 2 single-outlined square-cornered button 3 single-outlined square-cornered, and drop shadowed button

MoreFlags

Bits 0 and 1 describe title reference:

TitleIsPtr	equ 0
TitleIsHandle	equ 1
TitleIsResource	equ 2

Bits 2 and 3 describe the color table reference:

ColorTableIsPtr	equ %00000000
ColorTableIsHandle	equ %00000100
ColorTableIsResource	equ %00001000

Bits 4 through 7 must be zero.

Bits 8 through 10	Must be zero.
Bit 11 is FctlTellAboutSize	Not used by simpleButtonControl
Bit 12 is FctlProcNotPtr	Must be set
Bit 13 is FctlWantsEvents	Set if button has keystroke equivalent.
Bit 14 is FctlCanBeActive	Must be clear
Bit 15 is FctlActive	Must be clear

Key Equivalent Information

Key1	byte	upper or lower case value of char
Key2	byte	lower or upper case value of char
Mods	word	Modifier word required. This is the same word used in the event manager for modifiers. Only bits 8-15 are used; the others must be zero. The key modifier from the event record must match the upper byte of the modifier word.
ModsCharBits	word	Modifiers that must match. Set the bits corresponding to modifiers that must match what user types.

checkControl

pCount	WORD	valid values= 8, 9, 10
ID	LONG	Application Assigned ID
Rect	RECT	Bounding rect for control
ProcRef	LONG	checkControl = $82000000
Flag	WORD	See below
MoreFlags	WORD	See below
RefCon	LONG	Programmer Defined RefCon
TitleRef	LONG	Pointer, Handle, or ResourceID to title
InitialValue	WORD	0 for clear, 1 for checked
*ColorTableRef	LONG	Optional Ptr, Handle, or ResID to color table
*KeyEquivalent	block 6	Optional Key equivalent information

Flag

ctlInvis	bit7	1 = Invisible, 0 = visible
Reserved	bits 0-6	must be zero.

MoreFlags

Bits 0 and 1 describe title reference:

TitleIsPtr	equ 0
TitleIsHandle	equ 1
TitleIsResource	equ 2

Bits 2 and 3 describe the color table reference:

ColorTableIsPtr	equ %00000000
ColorTableIsHandle	equ %00000100
ColorTableIsResource	equ %00001000

Bits 4 through 7	Must be zero.
Bits 8 through 10	Must be zero.
Bit 11 is FctlTellAboutSize	Not used by checkControl
Bit 12 is FctlProcNotPtr	Must be set
Bit 13 is FctlWantsEvents	Set if button has keystroke equivalent.
Bit 14 is FctlCanBeActive	Must be clear
Bit 15 is FctlActive	Must be clear

Key Equivalent Information

Key1	byte	upper or lower case value of char
Key2	byte	lower or upper case value of char
Mods	word	Modifier word required. This is the same word used in the event manager for modifiers. Only bits 8-15 are used; the others must be zero. The key modifier from the event record must match the upper byte of the modifier word.
ModsCharBits	word	Modifiers that must match. Set the bits corresponding to modifiers that must match what user types.

radioControl

pCount	WORD	valid values= 8, 9, 10
ID	LONG	Application Assigned ID
Rect	RECT	Bounding rect for control
ProcRef	LONG	radioControl = $84000000
Flag	WORD	See below
MoreFlags	WORD	See below
RefCon	LONG	Programmer Defined RefCon
TitleRef	LONG	Pointer, Handle, or ResourceID to title
InitialValue	WORD	0 for clear, 1 for set
*ColorTableRef	LONG	Optional Ptr, Handle or ResID to color table
*KeyEquivInfo	block 6	Optional Key equivalent information

Flag

ctlInvis	bit7	1 = Invisible, 0 = visible
FamilyNumber	bits 0-6	radio buttons with the same family number are set and cleared together. That is, setting one radio button in a family clears the others in the same family.

MoreFlags

Bits 0 and 1 describe title reference:

TitleIsPtr	equ 0
TitleIsHandle	equ 1
TitleIsResource	equ 2

Bits 2 and 3 describe the color table reference:

ColorTableIsPtr	equ %00000000
ColorTableIsHandle	equ %00000100
ColorTableIsResource	equ %00001000

Bits 4 through 7	Must be zero.
Bits 8 through 10	Must be zero.
Bit 11 is FctlTellAboutSize	Not used by radioControl
Bit 12 is FctlProcNotPtr	Must be set
Bit 13 is FctlWantsEvents	Set if button has keystroke equivalent.
Bit 14 is FctlCanBeActive	Must be clear
Bit 15 is FctlActive	Must be clear

Key Equivalent Information

Key1	byte	upper or lower case value of char
Key2	byte	lower or upper case value of char
Mods	word	Modifier word required. This is the same word used in the event manager for modifiers. Only bits 8-15 are used; the others must be zero. The key modifier from the event record must match the upper byte of the modifier word.
ModsCharBits	word	Modifiers that must match. Set the bits corresponding to modifiers that must match what user types.

scrollControl

pCount	WORD	valid values= 9, 10
ID	LONG	Application Assigned ID
Rect	RECT	Bounding rect for control
ProcRef	LONG	scrollControl = $86000000
Flag	WORD	See below.
MoreFlags	WORD	See below.
RefCon	LONG	Programmer Defined RefCon
MaxSize	WORD	initial size of "document"
ViewSize	WORD	amount initially visible
InitialValue	WORD	initial thumb position
*ColorTableRef	LONG	optional Ptr, Handle, or ResID to color table

Flag

ctlInvis	bit7	1=invisible, 0=visible
Reserved	bit5-6	Must be zero
horScroll	bit4	1=horizontal scroll bar, 0=veritical scroll bar
rightFlag	bit3	1=bar has right arrow, 0=bar has no right arrow
leftFlag	bit2	1=bar has left arrow, 0=bar has no left arrow
downFlag	bit1	1=bar has down arrow, 0=bar has no down arrow
upFlag	bit0	1=bar has up arrow, 0=bar has no up arrow

Note: Depending on the state of the horScroll bit, two of the other bits are ignored.

MoreFlags

Bits 0 and 1 must be zero.

Bits 2 and 3 describe the color table reference:

ColorTableIsPtr	equ %00000000
ColorTableIsHandle	equ %00000100
ColorTableIsResource	equ %00001000

Bits 4 through 7	Must be zero.
Bits 8 through 10	Must be zero.
Bit 11 is FctlTellAboutSize	Not used by scrollControl
Bit 12 is FctlProcNotPtr	Must be set
Bit 13 is FctlWantsEvents	Must be clear
Bit 14 is FctlCanBeActive	Must be clear
Bit 15 is FctlActive	Must be clear

statTextControl

pCount	WORD	valid values= 7, 8
ID	LONG	Application Assigned ID
Rect	RECT	Bounding rect for control
ProcRef	LONG	statTextControl = $81000000
Flag	WORD	See below
MoreFlags	WORD	See below
RefCon	LONG	Programmer Defined RefCon
TextRef	LONG	Pointer, Handle or ResourceID to text
*TextSize	WORD	optional text size field

Flag

ctlInvis	bit7	1 = Invisible, 0 = visible
Reserved	bits 2-6	must be zero.
FSubstituteText	bit 1	0=no substitution. 1=There are substitutions
FSubTextType	bit 0	0=Cstrings. 1=PStrings.

MoreFlags

Bits 0 and 1 describe title reference:

TextIsPtr	equ 0
TextIsHandle	equ 1
TextIsResource	equ 2

Bits 2 through 7	Must be zero.
Bits 8 through 10	Must be zero.
Bit 11 is FctlTellAboutSize	Not used by statText
Bit 12 is FctlProcNotPtr	Must be set
Bit 13 is FctlWantsEvents	Must be clear
Bit 14 is FctlCanBeActive	Must be clear
Bit 15 is FctlActive	Must be clear

If the text is referenced by pointer, the TextSize field must exist. If the text is referenced by handle or resource, the TextSize field is ignored (the size is obtained from the handle).

editLineControl

pCount	WORD	valid values= 8, 9
ID	LONG	Application Assigned ID
Rect	RECT	Bounding rect for control
ProcRef	LONG	editLineControl = $83000000
Flag	WORD	See below
MoreFlags	WORD	See below
RefCon	LONG	Programmer Defined RefCon
MaxSize	WORD	Max length of input line
DefaultRef	LONG	Pointer, Handle, or ResourceID to default text
*TextSize	WORD	optional text size field

Flag

ctlInvis	bit7	1 = Invisible, 0 = visible
Reserved	bits 0-6	must be zero.

MoreFlags

Bits 0 and 1 describe the default text reference:

DefaultIsPtr	equ 0
DefaultIsHandle	equ 1
DefaultIsResource	equ 2

Bits 2 through 7	Must be zero.
Bits 8 through 10	Must be zero.
Bit 11 is FctlTellAboutSize	Not used by editLineControl
Bit 12 is FctlProcNotPtr	Must be set
Bit 13 is FctlWantsEvents	Must be Set
Bit 14 is FctlCanBeActive	Must be Set
Bit 15 is FctlActive	Must be Clear

editTextControl:

pCount	WORD	the minimum parameter count is 7, the maximum is 24
ID	LONG	application assigned ID
boundsRect	Rect	Bounding rectangle for control
procRef	LONG	editTextControl = $85000000
flags	WORD	see below
moreFlags	WORD	see below
refCon	LONG	programmer defined refCon
textFlags	WORD	see below
indentRect	Rect	each coordinate of this rectangle specifies the amount of white space to leave between that edge of the boundsRect and the text itself. To use the default value for a particular coordinate, use $FFFF. The default values are (2,6,2,4) in 640 mode and (2,4,2,2) in 320 mode.
vertBar;	Handle	this is a handle to the vertical scroll bar to use. NULL => the record doesn't have a vertical scroll bar. $FFFFFFFF => create a scroll bar just inside the right edge of the *boundsRect*
vertAmount;	WORD	this is the number of pixels to scroll whenever the up or down arrow on the vertical scroll bar is pressed. $0000 => default of 9 pixels.
horzBar;	Handle	reserved. This currently must be NULL.
horzAmount;	WORD	reserved. This currently must be $0000.
styleRef;	LONG	this is a reference to the initial style information. NULL => use the default style and ruler information. Bits 0 and 1 of the *moreFlags* field are used to specify how this is used.
textDescripter;	WORD	this is an input text descripter that specifies how the initial text is referenced and what format it is in.
textRef;	LONG	this is the reference to the initial text. If this is NULL, then there is no initial text.
textLength;	LONG	this is the initial text length for all descripters that require one.
maxChars;	LONG	this is the maximum number of characters allowed in the text. NULL => no limit on characters
maxLines;	LONG	this is the maximum number of lines allowed in the text. NULL => no limit on lines.
maxHeight;	WORD	this is the maximum height of the document in pixels. NULL => no limit on the total height of the text. Note: all these *max* fields are cumulative; they will all be respected.
pageHeight;	WORD	this is the height of a page in pixels. If this parameter is $0000, then page breaks will not be kept track of and the user will not be allowed to insert explicit page breaks.
headerHeight;	WORD	this is the number of pixels to allocate from the top of the page for the header. This must be less than (*pageHeight - footerHeight*).
footerHeight;	WORD	this is the number of pixels to allocate from the bottom of the page for the footer. This must be less than (*pageHeight - headerHeight*).
pageBoundary;	WORD	this is the number of pixels between the footer of one page and the header of the next. It is used to visibly indicate page breaks. $FFFF means use the default value of 4 pixels.
colorRef;	LONG	see the description of the colorTable record. Bits 2 and 3 of the *moreFlags* field are used to specify the way the color table is referenced.

Flag:

bit #	name	if SET
bit 15 - 8:	-----	Reserved. Must be CLEAR.
bit 7:	ctlInvis	Reserved. Must be CLEAR.
bit 6 - 0:	-----	Reserved. Must be CLEAR.

moreFlags:

bit #	name	if SET
bit 15:	fCtlActive	Must be CLEAR.
bit 14:	fCtlCanBeActive	Must be SET.
bit 13:	fCtlWantsEvents	Must be SET.
bit 12:	fCtlProcRefNotPtr	Must be SET.
bit 11:	fTellAboutGrow	The record will resize itself when the window changes size.
bit 10 - 4:	-----	Reserved. Must be CLEAR.
bit 3 - 2:	colorDescripter	00 = ptr, 01 = handle, 10 = resource ID, 11 is invalid.
bit 1 - 0:	styleDescripter	00 = ptr, 01 = handle, 10 = resource ID, 11 is invalid.

textFlags:

bit #	name	if SET
bit 31:	fNotControl	Don't allocate a custom control for the TextEdit record.
bit 30:	fNoWordWrap	Do not word wrap the text; only break lines on CR's.
bit 29:	fNoScroll	Do not allow any manual or auto scrolling.
bit 28:	fReadOnly	Do not allow editing of the text. Note that copy operations may still be performed.
bit 27:	fSmartCutPaste	Intelligent cut and paste will be supported as described in the initial featur. list for GS TextEdit.
bit 26:	fTabSwitch	When the user types a TAB, switch to the next control in the control list that can be activated. (see the initial feature list for more details)
bit 25:	fDrawBounds	Draw a box around the TextEdit control, just inside the *boundsRect*. The pen size for the box is (2,1)(h,v)
bit 24:	fColorHilight	Use the color table to do color hilighting. This is slower than the default which is to just invert the hilighted text.***This bit must be clear for v1.0.***
bit 23:	fTransparentText	The background color for the text will be ignored and the text will be drawn in foreCopy mode.
bit 22 - 0:	-----	Reserved. Must be CLEAR.

popUpControl

pCount	WORD	valid values= 9 or 10
ID	LONG	Application Assigned ID
Rect	RECT	Bounding rect for control
ProcRef	LONG	popUpControl = $87000000
Flag	WORD	See below
MoreFlags	WORD	See below
RefCon	LONG	Programmer Defined RefCon
TitleWidth	WORD	Width in pixels of the portion of the Rect to use for the title *See below
MenuRef	LONG	Ptr, Handle, or ResourceID to menu definition
InitialValue	WORD	Item ID of the currently selected menu item
*ColorRef	LONG	Reference to color table (optional field, depends on pCount)

Flag

ctlInvis	Bit 7	1 = Invisible, 0 = visible
FType2PopUp	Bit 6	0 = Draw normal pop-up, 1 = draw pop-up with white space
FDontHiliteTitle	Bit 5	1 = Don't hilite title when menu is popped up, 0 = Hilite in the normal way.
FDontDrawTitle	Bit 4	0 = Draw title in noraml way, 1 = no title is drawn.
FDontDrawResult	Bit 3	0 = Draw result in result area after a selection. 1= don't draw result after selection!
FInWindowOnly	Bit 2	0 = Let the pop up use the whole screen. 1 = Keep the pop up in the window.
FRightJustifyTitle	Bit 1	0 = Left justify the title in the bounding rect. 1 = Right justify the title at TitleWidth pixels from the left of the bounding rect.
FRightJustifyResult	Bit 0	0 = Left justify the result drawn after a selection TitleWidth pixels from the left of the bounding box. 1= right justify the result drawn after a selection at the right edge of the bounding box.

MoreFlags

Bits 0 and 1

MenuRefIsPtr	equ 00
MenuRefIshandle	equ 01
MenuRefIsResource	equ 10

Bit 2 is FMenuDefIsText	Set this bit if MenuRef is a pointer to a text stream like you would pass to NewMenu. Clear this bit if the menu is defined as a MenuTemplate. Note: if the bit is set, the menu manager assumes that the MenuRef is a pointer.
Bits 8 through 10	Must be zero.
Bit 11 is FctlTellAboutSize	Not used by popupControl
Bit 12 is FctlProcNotPtr	Must be set
Bit 13 is FctlWantsEvents	Set if popUp can have keystroke equivalent.
Bit 14 is FctlCanBeActive	Must be clear
Bit 15 is FctlActive	Must be clear

How to use all this shit.

Control Record:

control rectangle:

This rect defines the bounding rect for the pop-up rectangle and the pop-up's title. Given that the programmer has defined the bottom right point(x2,y2) for this rect, the length of the title string determines the length of the pop-up rect. (i.e. pop-up rect = (y1, x1+TitleWidth , y2, x2)). If the programmer has defined the bottom right point as (0,0) the control manager will calculate this coordinate for you (so all you have to worry about is where you want the control to start). The control manager calculates the right edge with the following formula: x2 = x1 + TitleWidth or actual width of title if TitleWidth=0 + the width of the widest item in the pop-up menu, bottom edge: y2 = y1 + height of tallest item in the pop-up menu.

TitleWidth:

The TitleWidth field is provided mainly to make it easier for the programmer to position the pop-up in the window. As you can see in Figure 2, the TitleWidth is used as an offset from the upper left point of the control rect. (x1,y1), to the left edge of the pop-up rectangle. When creating a series of pop-up controls, using the same value for the TitleWidth field will create all the pop-ups with the left edge of the pop-up rectangle at the exact same horizontal position as all the other pop-ups (see Figure 1. of the menu manager chapter). Since it is difficult for the programmer to predict what the exact length of the pop-up's title will be, it is difficult then to determine exactly where the pop-up rectangle will be drawn. Having the titleWidth field makes this easier. Having TitleWidth=0 causes automatic calculation of the title's width, with the pop-up rect immediately following the title.
Note: Setting TitleWidth can also be used even if there is no title to be drawn. The programmer must be careful that the actual width of the title does not exceed the value given for TitleWidth or unexpected results may occurr.

Flags:

FRightJustifyResult (bit 0):

"Result" is defined as the currently selected menu item in the pop-up. Right justifying the result just means the result displayed is right justified relative to the left edge of the pop-up rectangle. Setting this bit has no effect if bit 3 is also set (FDontDrawResult). When this bit is cleared, the title is drawn starting at the left edge of the control rect specified.

FRigthJustifyTitle (bit 1):

Figure 2. shows a pop-up control with its title left justified and TitleWidth=100. Figure 3. shows the exact same pop-up if bit 1 were set to right justify the title. This bit has no effect if the TitleWidth field is not used or if bit 4 is set (FDontDrawTitle). Right justification is done relative to the right edge defined by (x1 + TitleWidth). When the title is right justified as in Figure 3., the top left point of the control rect is adjusted accordingly.

FInWindowOnly (bit 2):

Setting this bit contrains the pop-up to the port rect of the window port. In other words the pop-up menu will not "pop" outside of the window's content region. It is advisable that bit 6 (FType2PopUp) is also set in conjunction with this bit. See explanation of Type2 pop-ups in the Menu Manager chapter.

FDontDrawResult (bit 3):

Don't draw the currently selected menu item in the pop-up rectangle.

FDontDrawTitle (bit 4):

Pop-up control is drawn without a title. It should be noted that when writing the menu definition a "bogus" title is still needed even if it is not going to be used. If TitleWidth is defined then the control rect is offset by TitleWidth and the control rectangle equals the pop-up rectangle.

FDontHiliteTitle (bit 5):

Setting this bit has no effect if bit 4 (FDontDrawTitle) is also set. When this bit is set the pop-up's title is not highlighted when the pop-up control is selected.

FType2PopUp (bit 6):

Setting this bit causes a different behavior in pop-ups when scrolling needs to take place. See explanation on Type2 pop-ups in the Menu Manaager section.

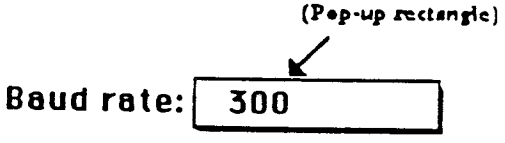

Figure 1.

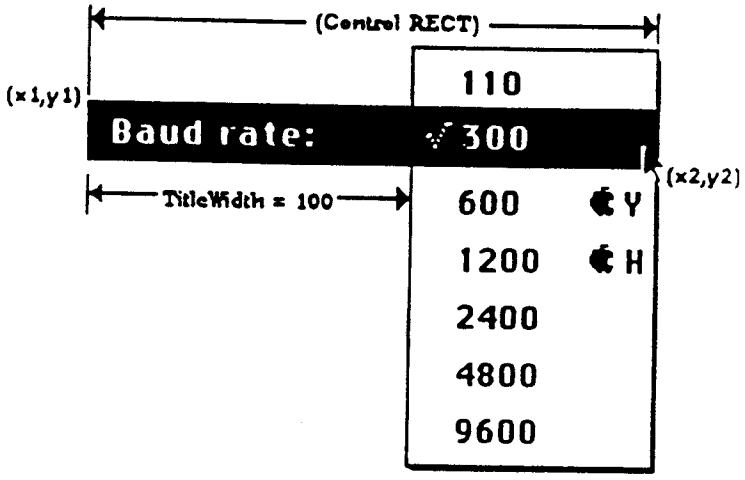

Figure 2. (title is left justified)

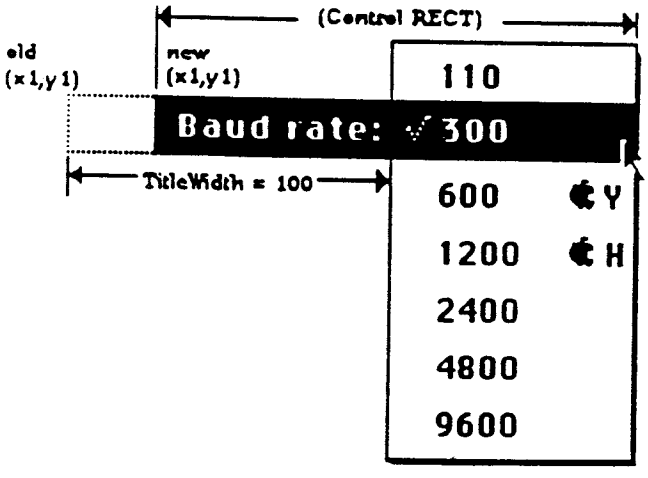

Figure 3. (title is right justified)

listControl

pCount	WORD	valid values= 14 or 15
ID	LONG	Application Assigned ID
Rect	RECT	Bounding rect for control
ProcRef	LONG	listControl = S89000000
Flag	WORD	Bits 7-15 standard use. Bits 0-6 must be zero.
MoreFlags	WORD	
RefCon	LONG	Programmer Defined RefCon
ListSize	WORD	num members in list
ListView	WORD	num members seen at one time
ListType	WORD	multi-selections, pascal strings etc
ListStart	WORD	First member to be seen
ListDraw	LONG	member drawing routine
ListMemHeight	WORD	height of each item
ListMemSize	WORD	num bytes in list record
ListRef	LONG	ref to list of member records
ColorRef	LONG	Reference to color table

Flag

ctlInvis	bit7	1 = Invisible, 0 = visible
Reserved	bits 0-6	must be zero.

MoreFlags

Bits 0 through 7	Not yet defined
Bits 8 through 10	Must be zero.
Bit 11 is FctlTellAboutSize	Not used by listControl
Bit 12 is FctlProcNotPtr	Must be set
Bit 13 is FctlWantsEvents	Must be clear
Bit 14 is FctlCanBeActive	Must be clear
Bit 15 is FctlActive	Must be clear

growControl

pCount	WORD	valid values= 6 or 7
ID	LONG	Application Assigned ID
Rect	RECT	Bounding rect for control
ProcRef	LONG	growControl = $88000000
Flag	WORD	See below
MoreFlags	WORD	See below
RefCon	LONG	Programmer Defined RefCon
*ColorTableRef	LONG	Optional reference to color table.

Flag

ctlInvis	bit7	1 = Invisible, 0 = visible
Reserved	bits 1-6	must be zero.
FCallWindowMgr	bit 0	1 = call GrowWindow and SizeWindow to track this control. 0 = just hilight the control.

MoreFlags

Bits 0 through 1 must be zero.

Bits 2 and 3 describe the color table reference:

ColorTableIsPtr	equ %00000000
ColorTableIsHandle	equ %00000100
ColorTableIsResource	equ %00001000

Bits 4 through 7	Must be zero.
Bits 8 through 10	Must be zero.
Bit 11 is FctlTellAboutSize	Not used by growControl
Bit 12 is FctlProcNotPtr	Must be set
Bit 13 is FctlWantsEvents	Must be clear
Bit 14 is FctlCanBeActive	Must be clear
Bit 15 is FctlActive	Must be clear

pictureControl

pCount	WORD	valid values= 7
ID	LONG	Application Assigned ID
Rect	RECT	Bounding rect for control
ProcRef	LONG	pictureControl = $8D000000
Flag	WORD	Bits 7-15 standard use. Bits 0-6 must be zero.
MoreFlags	WORD	See below
RefCon	LONG	Programmer Defined RefCon
PictureRef	LONG	Handle or ResourceID of picture

Flag

ctlInvis	bit7	1 = Invisible, 0 = visible
Reserved	bits 0-6	must be zero.

MoreFlags

Bit 0 of MoreFlags indicates what the PictureRef is

0	handle
1	resource

Bits 1 through 7	Must be zero.
Bits 8 through 10	Must be zero.
Bit 11 is FctlTellAboutSize	Not used by pictureControl
Bit 12 is FctlProcNotPtr	Must be set
Bit 13 is FctlWantsEvents	Must be clear
Bit 14 is FctlCanBeActive	Must be clear
Bit 15 is FctlActive	Must be clear

IconButtonControl

pCount	WORD	valid values = 7, 8, 9, 10, or 11
ID	LONG	Application Assigned ID
Rect	RECT	Bounding rect for control
ProcRef	LONG	iconButtonControl = $8B000000
Flag	WORD	See Below
MoreFlags	WORD	See Below
RefCon	LONG	Programmer defined RefCon
IconRef	LONG	Ptr, Handle or ResourceID of ICON
*TitleRef	LONG	optional Ptr, Handle or ResourceID to title
*ColorTableRef	LONG	optional Ptr, Handle or ResourceID of color table
*DisplayMode	WORD	Bit flag defining icon's appearance (default = 0)
*KeyEquivalent	block 6	Optional Key equivalent information

Flag

ctlInvis	Bit7	1=invisible, 0=visible
Reserved	Bits3-6	Must be zero
ShowBorder	Bit2	1=No Border, 0=show border
ButtonType	Bits0-1	Describes button type: 0 single-outlined round-cornered button 1 bold-outlined round-cornered button 2 single-outlined square-cornered button 3 single-outlined square-cornered and drop shadowed button

MoreFlags

Bits 0 and 1 describe title reference:

TitleIsPtr	equ 0
TitleIsHandle	equ 1
TitleIsResource	equ 2

Bits 2 and 3 describe the color table reference:

```
ColorTableIsPtr         equ %00000000
ColorTableIsHandle      equ %00000100
ColorTableIsResource    equ %00001000
```

Bits 4 and 5 describe the icon reference:

```
ColorTableIsPtr         equ %00000000
ColorTableIsHandle      equ %00010000
ColorTableIsResource    equ %00100000
```

Bits 6 and 7 must be zero.

Bits 8 through 10	Must be zero
Bit 11 is FctlTellAboutSize	Not used by simpleButtonControl
Bit 12 is FctlProcNotPtr	Must be set
Bit 13 is FctlWantsEvents	Must be zero
Bit 14 is FctlCanBeActive	Must be clear
Bit 15 is FctlActive	Must be clear

TitleRef

This optional field reference the title, which must be a p-string. If no title is to be used, set MoreFlags bits 0 and 1 to 0 and set this field to zero. If the pCount is 7, this field is zeroed. In this case , Moreflags bits 0 and 1 must be 0.

DisplayMode

This field is passed directly to the _DrawIcon routine for the display mode. The bits are defined as follow

Bit 0: selectedIconBit
1 = Invert image before copying
0 = Don't invert image

Bit 1: openIconBit
1 = Copy light-gray pattern instead of image
0 = Don't copy light-gray pattern

Bit 2: offLineBit
1 = AND light-gray pattern to image being copied
0 = Don't AND image

Bits 3 - 7: Reserved
Must be set to zero.

Bits 8-11: Foreground color to apply to black part of black-and-white icons

Bits 12-15: Background color to apply to white part of black-and-white icons

Key Equivalent Information

Key1	byte	upper or lower case value of char
Key2	byte	lower or upper case value of char
Mods	word	Modifier word required. This is the same word used in the event manager for modifiers. Only bits 8-15 are used; the others must be zero. The key modifier from the event record must match the upper byte of the modifier word.
ModsCharBits	word	Modifiers that must match. Set the bits corresponding to modifiers that must match what user types.

NOTE: The border of an icon button control is inset slightly from the control rectangle. The active region of the button is the inset version of the rectangle, and not the control rectangle itself. The reason for this is so that an outlined round button (usually representing the default button) does not draw outside the control rectangle.

Control Template Examples

This section contains an example of how to create a list of controls and create them all with one call to NewControl2. If you wish to try these out in your own program you will need a window that is about 160 lines tall and 600 pixels wide.

```
; Equates for the new control manager features
; ctlMoreFlags
FctlActive              equ $8000
FctlCanBeActive         equ $4000
FctlWantsEvents         equ $2000
FctlProcRefNotPtr       equ $1000
FctlTellAboutSize       equ $0800
TitleIsPtr              equ $0000
TitleIsHandle           equ $0001
TitleIsResource         equ $0002
ColorTableIsPtr         equ $0000
ColorTableIsHandle      equ $0004
ColorTableIsResource    equ $0008

; NewControl2 ProcRef values for standard control types
simpleButtonControl     equ $80000000
checkControl            equ $82000000
radioControl            equ $84000000
scrollBarControl        equ $86000000
growControl             equ $88000000
statTextControl         equ $81000000
editLineControl         equ $83000000
editTextControl         equ $85000000
popUpControl            equ $87000000
listControl             equ $89000000
iconButtonControl       equ $8B000000
pictureControl          equ $8D000000
```

; Here is the definition of my control list, note it is simply a list of pointers, these do not have to be in ; any special order. This list should always be terminated with a zero.

```
MyControls              dc.L theButton,theScroll,theCheck,Radio1,Radio2,StatControl
                        dc.L LEditControl,PopUp,IconButton,0

; Scroll bar color table as defined by the original control manager. The structure
; of these tables has not changed for the existing control types.
MyColorTable
                        dc.W 0                  ; outline color
                        dc.W $00F0              ; arrow unhilited black on white
                        dc.W $0005              ; arrow hilite blue on black
                        dc.W $00F0              ; arrow Background color
                        dc.W $00F0              ; Thumb unhilited
                        dc.W $0000              ; Thumb hilited
                        dc.W $0030              ; Page region solid black/white
                        dc.W $00F0              ; Inactive bar color

; Definition of a simple verticle scroll bar
theScroll               dc.W 10                 ; number of parms
                        dc.L 1                  ; application ID
                        dc.W 10,10,110,36       ; rectangle
                        dc.L scrollBarControl   ; scrollbar def proc
                        dc.W 3                  ; verticle scroll bar w/ arrows
```

```
                        dc.W FctlProcRefNotPtr  ; set procnotptr flag
                        dc.L 0                  ; refcon
                        dc.W 100                ; max size
                        dc.W 10                 ; size of view
                        dc.W 5                  ; initial value
                        dc.L MyColorTable       ; color table to use

SimpTitle               str 'Button'
theButton               dc.W 7                  ; num params
                        dc.L 2                  ; app ID
                        dc.W 10,40,0,0          ; a 25x30 button
                        dc.L simpleButtonControl        ; simple button
                        dc.W 0                  ; visible, round corner
                        dc.W FctlProcRefNotPtr+FctlWantsEvents
                        dc.L 0
                        dc.L simpTitle          ; button Title
                        dc.L 0                  ; no color table

CheckTitle              str 'CheckBox'
theCheck                dc.W 8                  ; num params
                        dc.L 3                  ; app ID
                        dc.W 25,40,0,0          ; bounding rect
                        dc.L checkControl       ; control type
                        dc.W 0                  ; flags
                        dc.W FctlProcRefNotPtr  ; MoreFlags
                        dc.L 0                  ; RefCon
                        dc.L CheckTitle         ; TitlePointer
                        dc.W 0

Radio1Title             str 'Radio1'
Radio1                  dc.W 8
                        dc.L 4
                        dc.W 45,40,0,0
                        dc.L radioControl
                        dc.W 1
                        dc.W FctlProcRefNotPtr
                        dc.L 0
                        dc.L Radio1Title
                        dc.W 1

Radio2Title             str 'Radio2'
Radio2                  dc.W 8
                        dc.L 5
                        dc.W 65,40,0,0
                        dc.L radioControl
                        dc.W 1
                        dc.W FctlProcRefNotPtr
                        dc.L 0
                        dc.L Radio2Title
                        dc.W 0

StatTitle               dc.B 'This is Stat Text'
StatControl             dc.W 8
                        dc.L 6
                        dc.W 120,10,135,210
                        dc.L statTextControl
                        dc.W 0
                        dc.W FctlProcRefNotPtr
                        dc.L 0
                        dc.L StatTitle
                        dc.W 17
```

```
EditDefault             str 'DefaultText'
LEditControl            dc.W 9
                        dc.L 7
                        dc.W 120,240,135,440
                        dc.L editLineControl
                        dc.W 0
                        dc.W FctlProcRefNotPtr
                        dc.L 0
                        dc.W 30
                        dc.L EditDefault
                        dc.W 30

PopUpMenu               dc.B '$$PopUpMenu:\N6',$00
                        dc.B '--Selection 1\N259',$00
                        dc.B '--Selection 2\N260',$00
                        dc.B '--Selection 3\N261',$00
                        dc.B '--Selection 4\N262',$00
                        dc.b '.'

PopUp                   dc.W 9
                        dc.L 8
                        dc.W 25,140,40,380
                        dc.L popUpControl
                        dc.W 0
                        dc.W FctlProcRefNotPtr  ; menu def is text
                        dc.L 0
                        dc.W 100
                        dc.L PopUpMenu
                        dc.W 259                ; initial value

IconButtonTitle         str 'Icon Button'

Icon                    dc.w 0                  ;black and white icon
                        dc.w 200
                        dc.w 10                 ;icon height in pixels
                        dc.w 40                 ;icon width in pixels
;
; Data for icon goes here (omitted)
;

IconButton
            pCount      dc.w 10
            ID          ds.l 1
            CtrlBounds  dc.w 40,40,80,100       ;button rectangle
            ProcRef     dc.l iconButtonControl  ;icon button control
            Flag        dc.w 0                  ;single outline, round-cornered
            MoreFlags   dc.w FctlProcRefNotPtr  ;get defproc from resource
            RefCon      dc.l 0
            IconRef     dc.l Icon               ;pointer to icon
            TitleRef    dc.l IconButtonTitle    ;pointer to p-string title
            ColorTab    dc.l MyColorTable       ;pointer to color table
            DisplayMode dc.w 0                  ;standard drawing of icon
```

To create the above new controls in a window you can use the NewControl2 call like this:

```
                        pha                     ; room for result
                        pha
                        PushLong WindPointer    ; Pointer to Owner window
                        PushWord #ptrToPtr      ; Input verb for ptr to table
                        PushLong #MyControls    ; pointer to table of templates
```

```
_NewControl2
pla                     ; discard these bytes, only verb
pla                     ; single ptr returns a value here
```

Change History

17 Jan 89 Steven Glass

Added error information for new calls.
Added summary of error codes.
Described pathological behavior of new calls.
Removed call GetCodeResConverter (It is now part of Misc. Tools).
Described the drawing environment established for CallCtlDefProc

18 Jan 89 Konstantin Othmer

Added icon button control descriptions.

27 Jan 89 Konstantin

Updated icon button control description.

01 Feb 89 Harry Yee

Added descriptions of the various bit settings of the pop-up control flag and some of the fields in the control record itself. Changed the pop-up control record to reflect that the PCount can be either 9 or 10 depending on whether there is a color table field or not.